Data Collection and Modeling

Course 1 - Intro (Course & Concepts)

Why Would You Be Interested?

Stored data will reach 175 zettabytes in 2025

Data can be repurposed, reused, related, new insights can be generated

Useful in all domains

What Does DCM Mean?

It consists of two parts:

- Collection
- Modeling

Course/Lab Info

Course

- Theoretical concepts, case studies, demos
- 50% of final grade
- Exam (most likely quiz using Moodle)

Lab

- Hands-on
- 50% of final grade
- Most likely project based

Topics

- Introduction
- File types; data sharing
- Local vs remote acquisition
- Data transport formats
- Data cleaning
- Web crawling
- APIs: REST vs SOAP
- SQL
- Data types and structures

Introduction

Data Science Lifecycle

Source: https://www.sudeep.co/data-science/2018/02/09/Understanding-the-Data-Science-Lifecycle.html

Data Collection

“Data collection is the process of gathering data for use in business decision-making, strategic planning, research and other purposes. It is an important part of data analytics applications and research projects”

(In Data Science) “Data collection is the process of accumulating data that's required to solve a problem statement”

Data Collection Steps

- Formulate the problem
- Determine data type to collect
- Identify data sources
- Set a time frame for data
- Collect data

Data Collection Types and Methods

- Primary Data
  - Surveys
  - Interviews/Focus Groups
  - Observations
- Secondary Data
  - Track Transactional Data
  - Social Media
  - Online Activity Tracking
  - Forms (Subscription, Registration)

Quantitative Data

Quantitative data relates to information that can be quantified (as numbers). It can be measured or counted, thus given a numerical value.

Quantitative data can be used to answer "how many", "how much" or "how often" type of questions.

Examples:

- How many students received offers after internship?
- How much will be the annual income for such a student?
- How often does a company organize internships?

Some of quantitative data collection methods are: surveys, experiments, polls.

Qualitative Data

Qualitative data has a descriptive nature, it can't be expressed as numerical values.

It is used to describe information that can't be measured or counted. It usualy refers to certain characteristics.

Quantitative data can be used to answer "why" or "how" type of questions.

Examples:

- The opinion of a student about its internship period (good, bad, worthy)

Some of qualitative data collection methods are in-depth interviews, focus groups, observations.

Qualitative vs Quantitative Data

Qualitative (Categorical) Data

- Nationality
- Gender
- Native Language
- High-School Specialization

Quantitative (Numerical) Data

- Age
- Height
- Weight
- Admission Grade

Data Collection Plan

2 main methods:

- diagram: visual, maps the flow of information

- plan table: analytical, list of variables by source

A workflow diagram starts with what data is being collected (quantitative/qualitative) and follows through from how it is collected to where it is stored and how it is shared(reporting presentation/online dashboard).

The plan outline consists in filling out a plan table. It helps organizing variables by source, method of acquisition, timeline, storage, and how it is analyzed and shared.

Information Workflow Diagram

Provides a graphic overview of the process

Uses symbols and shapes to depict the steps to complete from start to finish

Shows who is responsible for work at each step in the process

Each element of a workflow illustrates the flow between each step. Each step includes one of three parameters:

- Input: information required to complete the step
- Transformation: the changes that create the output
- Output: the result of the transformation

Data Collection Plan Outline

- What questions need to be answered?
- What data is available?
- How much data is needed?
- How to measure data?
- Who is performing data collection?
- Where is data collected from?
- Is a sample enough?
- What is the display format?

Sample Size

Not always is feasible to get results for/from all the instances. In this case a random sample should be taken to represent the population as a whole.

Sample size is important:

- Too small: outliers and/or anomalies distort the results
- Too big: complexity, increase cost, time consuming compared to accuracy gain

Sample size calculator

Data Quality

Criteria for data quality:

- Accuracy: data needs to be accurate
- Relevancy: should be appropriate for intended use
- Completeness: should not have missing values or records
- Timeliness: should be up to date
- Consistency: should be in the expected format
- Compliance: should comply with legal obligations

Data Collection and Modeling

Course 2: Files & File Sharing

Files

Definition

“A computer file is a resource for storing information, which is available to a computer program and is usually based on some kind of durable storage. A file is durable in the sense that it remains available for programs to use after the current program has finished.” (definitions.net)

“A file (...) is a self-contained piece of information available to the operating system and any number of individual programs.” (lifewire.com)

Files

A computer system can distinguish between 3 different file types:

- regular: are used to store data
  - text files: lines of text ended by EOL and a final EOF
  - binary files: bits that can represent custom data
- directory: information containers used to access other files
- special: usually physical devices or communication channels (FIFOs)

Files

Text

```
int main(){
        return 0;
}
```

compile…

Binary

```
00003000  00 00 00 00 00 00 00 00  08 40 00 00 00 00 00 00  |.........@......|
00003010  47 43 43 3a 20 28 55 62  75 6e 74 75 20 39 2e 34  |GCC: (Ubuntu 9.4|
00003020  2e 30 2d 31 75 62 75 6e  74 75 31 7e 32 30 2e 30  |.0-1ubuntu1~20.0|
00003030  34 2e 31 29 20 39 2e 34  2e 30 00 00 00 00 00 00  |4.1) 9.4.0......|
00003040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00003050  00 00 00 00 00 00 00 00  00 00 00 00 03 00 01 00  |................|
00003060  18 03 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00003070  00 00 00 00 03 00 02 00  38 03 00 00 00 00 00 00  |........8.......|
00003080  00 00 00 00 00 00 00 00  00 00 00 00 03 00 03 00  |................|
```

Data Types

- Observational
- Experimental
- Derived (compiled)
- Simulation
- Canonical (reference)

Data Formats

Many formats: text, numeric, multimedia, domain specific, device specific (medical imagistics)

Accessible format characteristics:

- Uncompressed
- Standard character encoding
- Non proprietary
- Open standards
- Popular among community

File Formats

Represents the structure of data stored inside a file

Formats can be grouped in various categories

Usually possible to convert from one format to another within same category

docs.fileformat.com

fileinfo.com/filetypes/

File Formats

Format	High Confidence	Medium Confidence	Low Confidence
Text	Plain text UTF-8 BOM (.txt) XML with schema (.xml)	Plain text ISO 8859 (.txt) HTML (.html, .htm), CSS	Microsoft word (.doc) Wordperfect (.wpd)
Raster	Uncompressed TIFF (.tiff) True color 24bit PNG (.png)	Compressed TIFF (.tiff) GIF (.gif), BMP, 8 bit PNG	Rawfile Photoshop (.psd)
Vector	SVG (.svg)	Computer Graphics Metafile (.cgm)	Macromedia Flash (.swf), Postscript (.eps)
Containers	TAR (.tar) No compressed ZIP (.zip)	Compressed ZIP (.zip)	

File Formats

Format	High Confidence	Medium Confidence	Low Confidence
Spreadsheet / Database	Separated values (.tsv, .csv), Delimited text	dBASE (.dbf), MS Excel (xlsx), SAS (.sas)	Excel (.xls)
Multimedia	Uncompressed AVI, FFV1/Matroska (.mkv)	MPEG-1, MPEG-2, Motion JPEG 2000 (.jp2)	Windows MEdia Video (wmv), RealAudio (.ra)
Digitized documents	Uncompressed TIFF PDF/A-1		
Programs		Source code	Executable

File Sharing

Definition

“the practice of making computer files available to other users of a network, in particular the illicit sharing of music and video via the internet.” (Oxford Languages)

“File sharing is a productivity tool that allows select users to share files with one another remotely. Files can be shared with just one or two individuals or entire organizations. File sharing also works with nearly any file type, so users can exchange text documents, images, audio and video files, PowerPoint slides and more.” (Mitel)

Types of sharing

- Operating system file sharing
  - Sharing files between users using network layer (FTP)
- Internet file sharing
  - P2P
  - File sync and sharing services
  - Portals
  - Rsync
  - Data sync

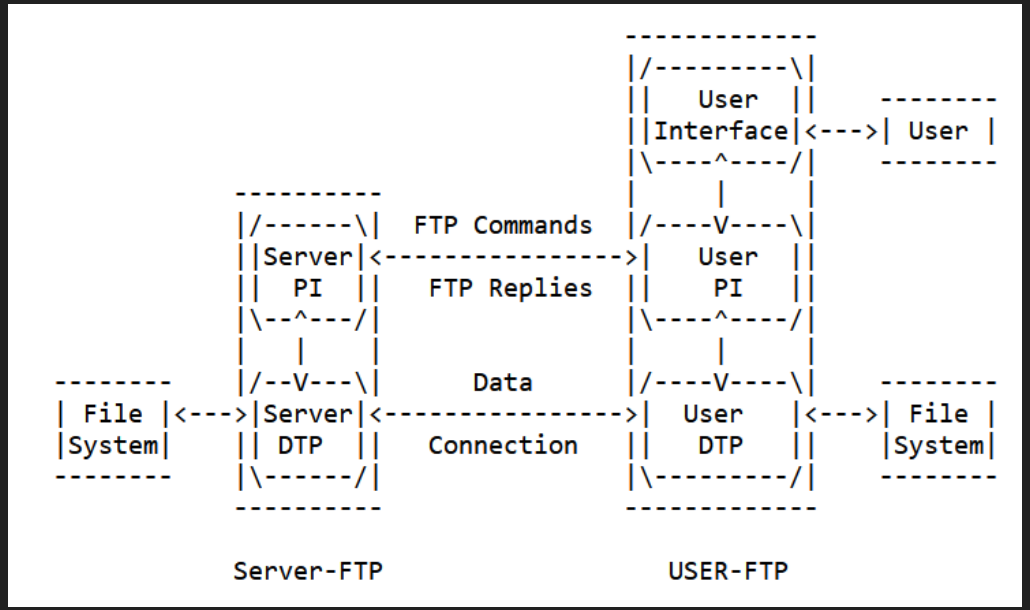
FTP

File Transfer Protocol (RFC 959) is used to communicate and transfer files between computers on a TCP/IP network

Objectives:

- Promote sharing of files
- Encourage use of remote computers
- Prevent against variations in file storage systems between hosts
- Transfer data in a reliable and efficient manner

FTP



The FTP Model

FTP Apps

- FileZilla: open source, multithreaded transfers, SFTP, FTPS; all OSs
- Cyberduck: open source, cloud storage browser, SFTP, WebDav, AWS S3; Windows and MacOS
- FireFTP: open source, Firefox add-on (up to 57, Waterfox after that) , FTP, SFTP, FTPS; all OSs
- WinSCP: popular, many features, FTP, SFTP, SCP, FTPS, WebDAV, AWS S3; Windows

P2P

File-sharing technology that allows the users to access files over a network. Users/computer accounts in this network are called to as peers; they request files from other peers using TCP or UDP connections.

Such a P2P network allows communication without a server; there is no central server for handling requests, the peers interacting without the requirement of a central server.

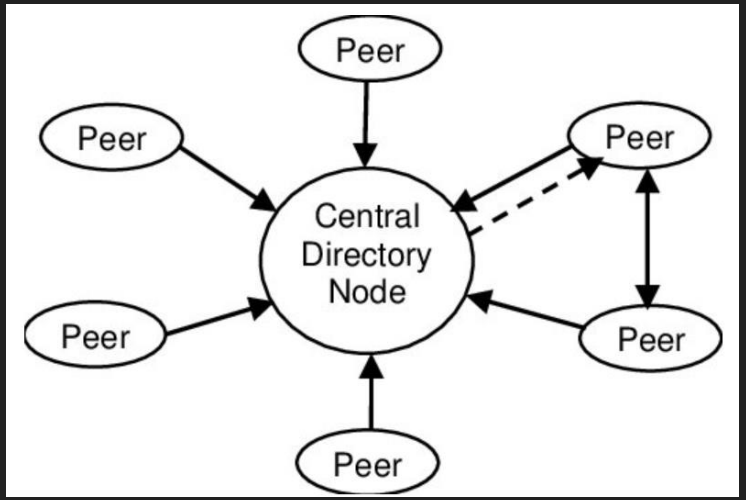
For a peer request there may be multiple peers that have a copy of the file. There are 3 architectures that allow communication with these peers:

- Centralized (Central Directory)
- Pure (Query Flooding)
- Hybrid

P2P - Centralized

- similar to client-server architecture: it maintains a central server for directory services
- the peers inform central server about their IP address and the files available for sharing
- the server queries the peers at regular intervals to check if still connected

[Napster]

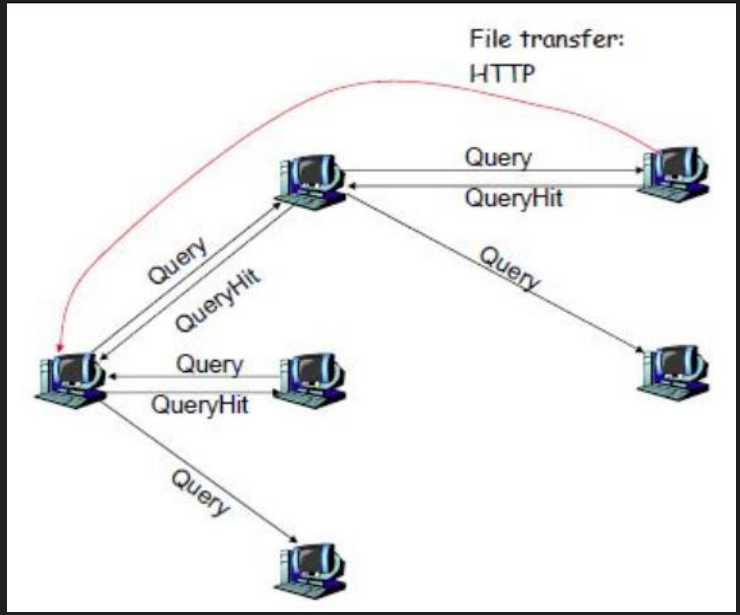


source: researchgate.net

P2P - Pure

- a node searching for a file contacts all neighbors in the system, they contact their own neighbors and so on until a "hit" is obtained (file is located)
- this process assumes no knowledge about the network topology
- if there are multiple hits, the client can select one from the list of peers

Gnutella

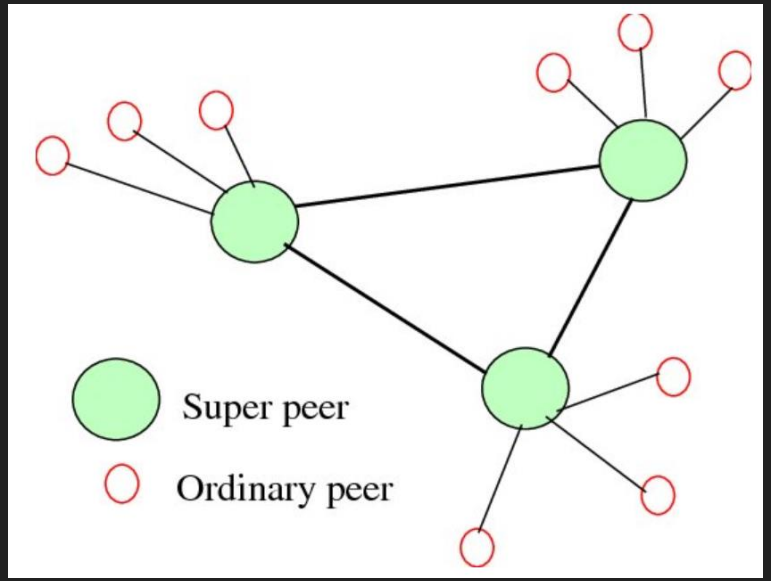


source: Wikipedia

P2P - Hybrid

- makes use of both centralized and pure
- there is no centralized server for query processing
- not all pears are equal; ones with high bandwidth and stable connectivity are called leaders/supernodes
- each supernode has a subset of the rest of th peers and indexes their IPs and shared files

KaZaA



source: researchgate.net

File sync / File Sharing

- file syncing is the process of file updating (in real-time) across multiple devices. It allows concurrent work from more than one person on the same file
- there are two types of syncing:
  - one-way sync: files are updated from one single source to multiple target locations; no data is sent back for update to the source
  - two-way sync: there are several locations that work together using two-way communication between every pair in the system
- usually cloud based
- Google Drive, IDrive, Sync.com, Microsoft Onedrive

Portals

Client portals represent a secure online location that clients can access at any time to view and manage files (upload, download)

Allows:

- easy share files in a secure manner by creating a link to a public or private share
- limit the maximum number of downloads for a shared file
- automatically expire a share link after a certain time
- anonymous file uploads

Files are always in sync and protected by regular backups

Portals

- Huddle: cloud based, audit trails (timestamps for files), enhanced (government grade) security, HIPAA and GDPR compliant, branded client portals
- Citrix Sharefile: enhanced security using password and device lock, flexible storage (on cloud or on-premise storage)
- Dropbox for Business: third party app integrations (Zoom, Slack), top-tier security, fast file transferring, task management systems
- FileCloud: flexible hosting (public, private), syncs across all devices

rsync

stands for remote sync

is a remote and local file synchronization tool

uses an algorithm to minimize the amount of data copied by only moving the portions of files that have changed

in order to use rsync to sync with a remote system, SSH access need to be configured between local and remote machines and rsync installed on both systems

written in C as a single threaded application; the algorithm is a type of delta encoding, and is used for minimizing network usage

Data Sync

The process of synchronizing data between two or more devices and updating changes automatically between them to maintain consistency, ensures accurate, secure, compliant data

Input data gets cleaned, checked for errors, removed duplication and checked for consistency

We consider to be local synchronization when devices and computers use the same local network, and to be remote synchronization when it takes place over a mobile network

Data changes must upgrade every system in real-time to avoid mistakes, prevent privacy breaches, and ensure that the most up-to-date data is the only information available

Data synchronization ensures that all records are consistent, all the time

Data Sync Types

There are some data sync methods that can update more than one copy of a file at a time, and some may update only one

- File Synchronization: most used for home backups, external hard drives, or updating portable data via flash drive
- Version Control: synchronizing solution for files that can be altered by more than one user at the same time
- Distributed File Systems: multiple file versions must be synced at the same time on different devices, those devices must always be connected for the distributed file system to work
- Mirror Computing: provides different sources with an exact copy of a data set. Useful for backup, provides an exact copy to one other location

Data Sync Challenges

- security: data that is updated in different locations hass to meet regulatory standards and privacy laws
- data quality: updates have to maintaining strict integrity of information within a secure environment
- management: data management has to be done in real-time to ensure accuracy and prevent errors
- performance: usually data synchronization is done using ETL (5 steps)
    - Extraction from the source
    - Transfer
    - Transformation
    - Transfer
    - Load to target

When dealing with a large volume of data, synchronization must be a priority to keep performance

- data complexity: formats may change according to needs and technological modifications; data should be available for both old and new systems operations

ETL - E

Data needs to be managed from various sources and saved in a destination system. In this first step of the ETL process, structured and unstructured data is imported and consolidated into a single repository. Volumes of data can be extracted from a wide range of data sources, including:

- Databases and legacy systems
- Cloud, hybrid, and on-premises environments
- Sales and marketing applications
- Mobile devices and apps
- CRM systems
- Data storage platforms
- Data warehouses
- Analytics tools

ETL -T

Data quality can be assured by applying rules and regulations. Data transformation consists of several sub-processes:

- Cleansing — deal with inconsistencies and missing values
- Standardization — apply formatting rules
- Deduplication — remove redundant data
- Verification — unusable data is removed and anomalies are flagged
- Sorting — organize data based on types
- Other data quality tasks — additional/optional applied rules to improve data quality

Transformation is usually the most important/consistent part of the ETL process

ETL - L

- load the transformed data into a new destination (usually data lake or data warehouse). Data can be loaded in its entirety (full load) or incremental load:
- full loading: everything that comes from the transformation step goes into new, unique records in the destination.
- incremental loading: compares incoming data with what’s already in the destination and loads additional records if new information is found

Data Collection and Modeling

Course 3: Data Acquisition

Definition

“Data acquisition is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer. Data acquisition systems (abbreviated with the acronym DAS or DAQ) typically convert analog waveforms into digital values for processing.” (Dataforth)

“Data acquisition (DAQ) is the process of measuring an electrical or physical phenomenon, such as voltage, current, temperature, pressure, or sound.” (National Instruments)

Other Definitions

- The process that uses a data acquisition system (DAS) to measure and digitize real world measurements/signals data
- The process of collecting data that has already been measured and digitally transformed

Depending on the project both these processes can be involved together with synthetic data generation (synthetic data obtained by simulation and generation to have enough instances for training a ML model)

Related Terms

- **Measurement** - quantitative determination of a physical characteristic. Usually it is the conversion of a physical quantity or observation to a domain where the value can be determined (by human or computer).
- **Instrumentation** – devices for converting a physical quantity to a quantity observable by a human or computer.
- **Data Acquisition** - Gathering information from measurement sources, such as sensors/transducers.
- **Sensor** - device that converts physical phenomena such as temperature, vibration, and acoustics to electrical signals.

Applications of DAQ

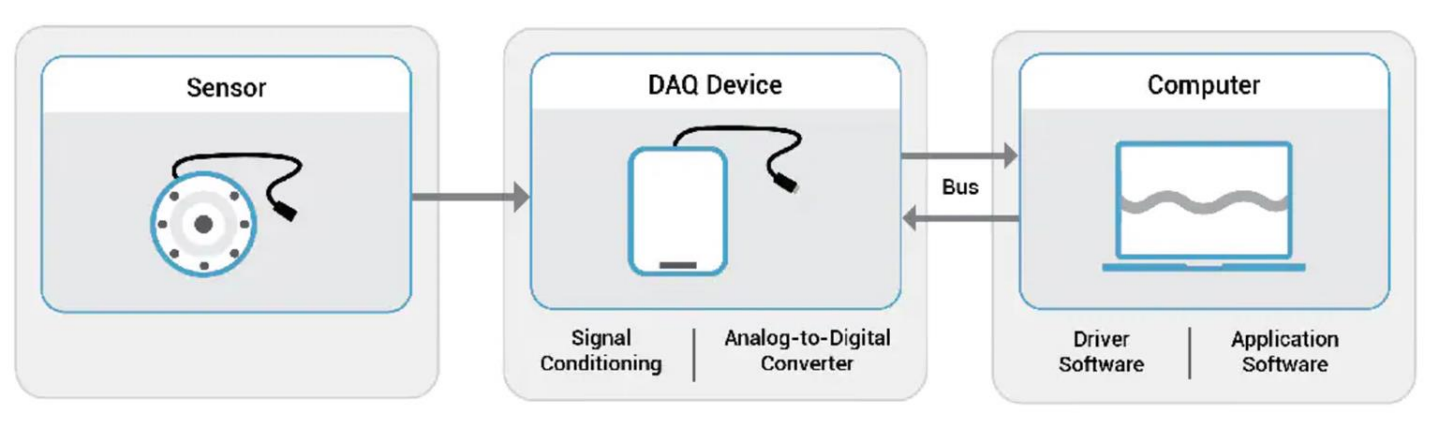
- **Scientific Exploration**: collect information for experiments, simulations, observations
- **Industrial Efficiency**: monitoring manufacturing processes for quality assurance, optimizing overall efficiency
- **Environmental Insights**: tracking critical factors: air quality, water levels, soil conditions; allows effective environmental management and timely disaster prediction
- **Healthcare and Biomedical Studies**: monitor vital signs, acquire physiological data, help in diagnostic accuracy
- **Automotive Evaluation**: testing vehicle performance, safety features, and efficiency across diverse scenarios

Challenges of DAQ

- **Accuracy and Calibration**: sensors and instruments must be accurate and well-calibrated
- **Data Integrity**: avoid signal interference and noise for data integrity
- **Sampling Rates**: crucial to choose it not to miss important data but also not to overload the system
- **Compatibility and Integration**: compatibility issues when integrating sensors and instruments from different manufacturers

DAQ System

consists of sensors, DAQ measurement hardware, and a computer with programmable software



Source: Omega

Sensors

Sometimes called transducers

“a transducer is a device that converts a signal from one physical form to a corresponding signal that has a different physical form. Therefore it is an energy converter” (Sensors and Signal Conditioning)

Sensors or transducers interact with the subject measured, either directly or indirectly (contact or non-contact). They convert the physical values to electric signals. The type of sensors used in a DAQ system varies based on the nature of its application.

Sensors

- **Active sensors**: produce output by means of external excitation supply; the physical properties changes depend on applied external effect
- **Passive sensors**: produce output without external excitation; they don’t need external voltage/power supply

- **Analog sensors**: produce analog output as continuous signal with respect to time
- **Digital sensors**: produce discrete non-continuous with respect to time; output signals converted and transmitted digitally

Sensor Types

Phenomena	Sensors
Force, pressure	Load cells Strain gauges
Fluid flow	Rotational flowmeters Ultrasonic flowmeters
Light	Photoconductive cells Vacuum tube photosensors
Position	Linear voltage differential transformers Optical encoders Potentiometers
Sound	Microphone
Temperature	Thermocouples Thermistors Resistance temperature detectors

source: electricaltechnology.org

Signal Conditioning

The process of optimizing the signal

Sometimes the signal can contain noise, can be only partially measured (weak)

- Calibration
- Linearization/Offsetting
- (External) Excitation
- Amplification
- Filtering

DAQ Hardware

The device connected between sensor and computer; can be connected to computer on various ports/slots (USB, PCI-Express)

Receives (analog) output signals from sensors and transforms/converts them to digital signals that can be processed by computers

Choosing the Right DAQ Hardware

1. What types of signals are used?
2. Is signal conditioning needed?
3. What is the sampling rate?
4. What resolution is needed?
5. What is the accuracy?

What types of signals are used?

Different types of signals have to be measured/generated in different ways; sensors convert physical phenomena into measurable electrical signals (voltage or current). Sensors can also receive a measurable electrical signal to produce physical phenomena.

Functions of DAQ Devices:

- Analog inputs measure analog signals
- Analog outputs generate analog signals
- Digital inputs/outputs measure and generate digital signals
- Counters/timers count digital events or generate digital pulses/signals

There are both dedicated and multifunctional devices that can be used

ADC

Takes environment (analog) data from sensors and converts it to a digital form (discrete levels) usable in computer environment.

Characterized by the number of bits (4, 8, 12, 16, …). The larger this value, the greater number of discrete digital values can be represented (resolution).

- Successive Approximation (SAR)
- Delta-Sigma
- Dual Slope
- Pipelined
- Flash

Is signal conditioning needed?

General-purpose DAQ devices usually measure or generate voltages in the ranges of +/-5 V , +/-10 V.

There are sensors that generate signals that are difficult to measure directly with general purpose DAQ devices; these sensors require signal conditioning: amplification, filtering, linearization.

	Amplification	Attenuation	Isolation	Filtering	Excitation	Linearization
Thermocouple	x			x		x
Thermistor	x			x	x	x
RTD	x			x	x	x
Strain Gage	x			x	x	x
Load, Pressure, Torque (mV/V, 4-20mA)	x			x	x	x
	x			x	x	x
Accelerometer	x			x	x	x
Microphone	x			x	x	x
Proximity Probe	x			x	x	x
LVDT/RVDT	x			x	x	x
High Voltage		x	x			

source : ni.com

What is the sampling rate?

Sampling rate represents the speed at which the ADC takes samples of a signal.

Each application's sampling rate depends on the maximum frequency component of the signal that needs to be measured or generated.

The Nyquist Theorem stipulates that, in order to accurately rebuild the signal, twice the highest component frequency is needed.

The practical experiments show that, to obtain the correct shape of the signal, the sample rate should be at least 10 times the maximum frequency.

What resolution is needed?

The resolution represents the smallest detectable change in the signal. It represents the number of binary levels an ADC can use to represent a signal.

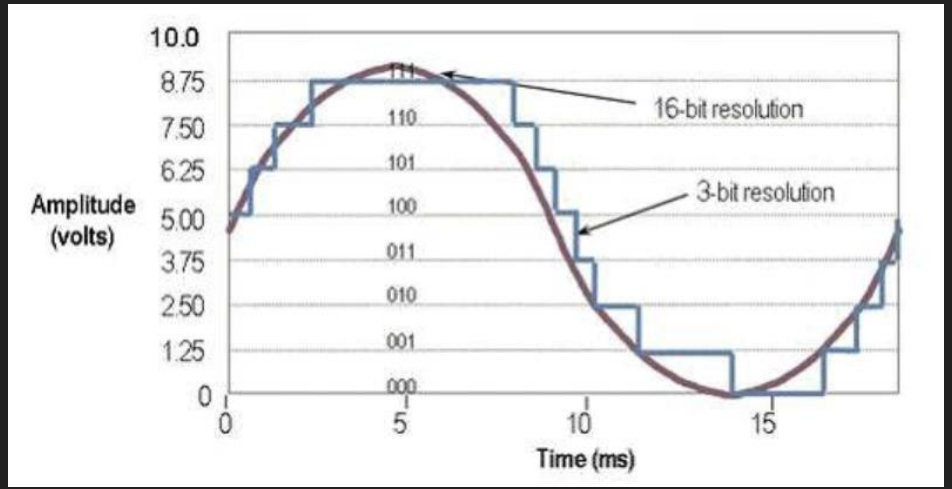
Imagine a 3-bit ADC and a 16-bit ADC:

- 3-bit ADC can represent

  8 (2^3) discrete voltage levels

- 16-bit ADC can represent

  65,536 (2^{16}) discrete levels



source: [ni.com]

What is the accuracy?

Accuracy can be stated as the uncertainty (amount) in a measurement with respect to an absolute standard.

Accuracy specifications usually contain the effect of errors due to gain and offset parameters. Offset errors can be given as a unit of measurement such as volts or ohms and are independent of the magnitude of the input signal being measured.

Gain errors depend on the magnitude of the input signal and are expressed as a percentage of the reading.

Total accuracy is equal to the sum of the two(example ±(0.1% of input +1.0 mV))

DAQ Features

- Digitization
- Multiplexing
- Conversion
- Transmission
- Binary output

Data Acquisition Options

- **Data Loggers**: data recording over a time period; it a self contained DAQ system with an embedded processor and software. Stand-alone, portable devices
- **Data Acquisition Devices**: can be connected on various ports (USB, Ethernet, PCI); contains signal conditioning and ADC but the software is on a connected computer
- **Data Acquisition Systems**: used for complex systems; integrates multiple sensors (different types) and synchronizes them.

Remote Data Acquisition

- Gather data from remote location and send it to a central machine for storage and processing
- Composed of sensors, DAQ hardware, communication hardware
- Allows monitoring and controlling equipment from a distance
- Offers
  - real-time data collection helping decision about remote equipment operation and maintenance/service
  - Monitor conditions in environments where a human operator can’t have access/can’t work securely

Remote Data Acquisition

- Web-based: accessible using a browser; easy to use
- Desktop based: installed on a desktop computer (RDP/SSH)
- Embedded: integrated with other devices (industrial controllers)
- Wireless: collected data from sensors is sent wirelessly to a controller; suitable for non wireable environments

Data can be collected simultaneously from several locations, saving time and money; Security can be a concern, especially with sensitive data; Networking plays a major role (redundant communication channels?)

Can be used for monitoring patients at home, power plants, collecting scientific data

Data Acquisition Methods for ML

Data can be collected to be cleaned and processed as training data for ML. We can identify 3 methos of data sourcing that can be used in any combination:

- Manual collection
- Data Procurement
- Open Source Warehouses

Data Collection and MOdeling

Course 4 - Data Exchange Formats

Definition

“Data exchange is the process of taking data structured under a source schema and transforming it into a target schema, so that the target data is an accurate representation of the source data” (Principles of Data Integration)

“Data exchange is the mutual transfer of information between people and devices. This includes the sending of documents or photos. Data exchange increasingly takes place in electronic form. The term comes from the field of data processing. The exchange of data is a part of data processing and is standardized. This means that certain standards are defined and formats are used. These standards serve to make data exchange efficient and uniform worldwide.” (TeamDrive)

Data Exchange Formats

- API (Application Programming Interface): web services over HTTP; SOAP, REST, GraphQL (a later course deals with this topic)
- ETL (Extract, Transform, Load): data is transferred by direct database connection
- **File Transfer**: files are transferred (using various formats) and then loaded for processing
- RPC (Remote Procedure Call): a program call launches the execution of a procedure on another address space (even remote computer)
- Message Brokers/Events Server: data is transferred by a delivery service
- Data Streaming: several data sources send data continuously to a process that ingests it in sequence

Data Heterogeneity

- Technical: Data exchange format
- Syntactical: Character encoding

Open Standards for Exchange Data

- Formats: describe structure of data
- Data types: how values are expressed
- Data transfers: define rules on exchanging data or providing access
- Rules: describe what data to share, formats, schemas
- Maps: describes how models are expressed using data exchange formats

Data Formats

- Text Based
  - CSV
  - XML
  - JSON
  - YAML
  - RDF
  - Plain Text
  - Specialized formats: SVG, GML, WKT, KML
- Binary Based
  - CORBA (CDR)
  - BSON
  - Protocol Buffers (ProtoBuf)
  - Big Data: Avro, Parquet, Thrift
  - Specialized formats: WKB

Text Based

XML

Simple text-based format for structured data; derived from SGML; extensible

One of most often used format for sharing structured data; self-descriptive

Doesn’t use predefined tags; structure and tags must be defined

Hardware and software independent format

Separates data from presentation

W3C Recommendation

XML

Documents formatted as element trees

A document has a root element

Relationships between elements could be: parent, child, sibling

There is a (optional) prolog that defines the version and character encoding

Elements must have a closing tag; tags are case sensitive; should be properly nested

Elements can have: text, attributes, elements

XML

Well Formed Documents:

- must have a root
- elements must have a closing tag
- tags are case sensitive
- elements must be properly nested
- attribute values must be quoted

Valid Document:

- Well formed
- conform to Document Type Definition
  - DTD: structure, elements and attributes for XML documents
  - XML Schema (alternative to DTD)

XML

Browser usually access XML documents by loading them in an XML DOM object by means of an XML Parser

XML can be displayed using XSLT

XQuery is used to find and extract attributes and elements from XML

XSLT uses XPath to find information in an XML document

XQuery tutorial

XPath tutorial

XML

```
<?xml version="1.0"?>
<!DOCTYPE cars [
  <!ELEMENT car     (make,model,type,price)>
  <!ELEMENT make    (#PCDATA)>
  <!ELEMENT model   (#PCDATA)>
  <!ELEMENT type (sedan, coupe, hatchback)>
  <!ELEMENT price   (#PCDATA)>
]>
<car>
  <make>Dacia</make>
  <model>Spring</model>
  <type>hatchback</type>
  <price>10.000</price>
</car>
```

```
<?xml version="1.0"?>
  <!ELEMENT car     (make,model,type,price)>
  <!ELEMENT make    (#PCDATA)>
  <!ELEMENT model   (#PCDATA)>
  <!ELEMENT type (sedan, coupe, hatchback)>
  <!ELEMENT price   (#PCDATA)>
```

```
<?xml version="1.0"?>
<!DOCTYPE cars SYSTEM "cars.dtd">
<car>
  <make>Dacia</make>
  <model>Spring</model>
  <type>hatchback</type>
  <price>10.000</price>
</car>
```

JSON

JavaScript Object Notation is a self-describing text format for storing and transporting data

Easy conversion between JavaString object to JSON string and vice-versa

Syntax with few rules

Hierarchical structure

Has to be parsed

JSON

- Data represented in name-value pairs
- Data is separated by comma
- Objects are enclosed by curly braces
- Arrays are enclosed within square brackets

Keys must be strings
Values must fall under one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

JSON

JSON is just a string with a specified format; contains only properties, no methods

Double quotes are required for string values and keys

JSON can be validated against errors (if any, will break the application) JSONLint

JSON is mainly used for data exchange and applications minify it to save bandwidth. This doesn't change the data in the JSON file, but it makes the JSON code unreadable. In order to make it readable again a JSON Formatter can be used

JSON

```
{
  "car": {
          "make": "Dacia",
          "model": "Spring",
          "type": "hatchback",
          "price": 10000
  }
}
```

YAML

Is a human-friendly data serialization language for all programming languages, but also often used for writing configuration files.

May stand for "yet another markup language" or "YAML ain’t markup language" (a recursive acronym), emphasizing that YAML is for data, not documents.

Has features that come from Perl, C, XML, HTML, and other programming languages. Is also a superset of JSON, JSON files being valid in YAML.

YAML

Uses Python-style for indentation to indicate nested elements.

Tab characters are not allowed, whitespaces being used instead.

There are no formatting symbols, such as braces, square brackets, closing tags, or quotation marks (except for abbreviated forms of lists[] or dictionaries {}).

YAML files use a .yml or .yaml extension.

YAML

The structure of a YAML file is a map or a list.

A map is formed by key-value pairs. Each key must be unique.

A map needs to be resolved before it can be closed, and a new map is created.

A map can be created by increasing the indentation level or by resolving the previous map.

A list includes values listed in a specific order and may contain any number of items.

A list sequence starts with a dash (-) and a space, indentation separates it from the parent

A list can be embedded into a map.

YAML

YAML also can contain scalars (arbitrary data encoded in Unicode) that can be used as values such as strings, integers, dates, numbers, or booleans.

Null values can be specified by “null” or “~”

Booleans can be defined using “True”, “False”, “On”, “Off”

A file starts with three dashes (-). Dashes indicate the start of a new YAML document.

YAML supports multiple documents, compliant parsers recognizing each row of 3 dashes as the beginning of a new one.

YAML

```
car:
  make: Dacia
  model: Spring
  type: hatchback
  price: 10000
```

RDF

Resource Description Framework is a model to represent data about physical objects and abstract concepts.

Uses a graph format to express relations between entities.

Modeled concepts are considered resources. Information is represented using statements in the format:

```
<subject> <predicate> <object>
```

The expressed relation is between the subject and the object, both being considered resources. RDF statements are also known as triples.

RDF

The same resource could be used in several triples with different roles. In this way, connections between resources can be expressed.

RDFs defined by IRIs and literals

- IRI - URI expressed using UNICODE; can be used in all positions in the triple
- Literal - basic value: string, date, number, etc; can be used only in object part of the triple

RDF

```
/* triples as < subject, predicate, object > */

<root, car, make>

<make, type, Dacia>

<root, car, model>

<model, type, Spring>

<root, car, type>

<type, type, hatchback>

<root, car, price>

<price, type, 10000>
```

SVG

Stands for Scalable Vector Graphics

Is used to define vector-based graphics for the Web

Defines the graphics in XML format

Elements and attributes can be animated

Is a W3C recommendation

Integrates with W3C standards like DOM and XSL

SVG

Images can be created/edited with any text editor

Images can be searched, indexed, scripted, and compressed

Images are scalable, zoomable

Graphics don’t lose any quality if they are zoomed or resized

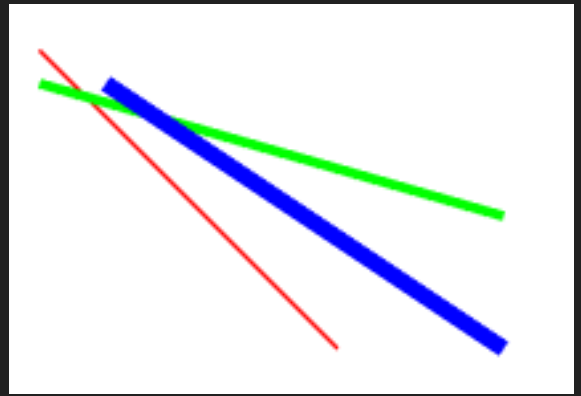
SVG is an open standard

SVG files are pure XML

Can be converted to binary image formats (PNG, JPG, PDF)

SVG

```
<svg width="160" height="160">
  <line x1="10" y1="10" x2="100" y2="100" style="stroke:#f00; stroke-width:1"/>
  <line x1="10" y1="20" x2="150" y2="60"  style="stroke:#0f0; stroke-width:3"/>
  <line x1="30" y1="20" x2="150" y2="100" style="stroke:#00f; stroke-width:5"/>
</svg>
```



Binary Based

BSON

Binary Javascript Object Notation is a binary encoded serialization of JSON documents.

Ads optional non-JSON-native data types, like dates and binary data.

Compared to other binary formats, like Protocol Buffers, BSON is more "Schema-less" than Protocol Buffers, providing the advantage of flexibility at a slight disadvantage of space efficiency.

BSON

Type	Description
byte	1 byte (8 bits)
int32	32 bits signed integer
int64	64 bits signed integer
decimal128	16 bytes decimal floating point
date	64 bits integer
objectId	12 bytes composed field
array	Data type dependent storage

BSON offers additional data types compared to JSON; decimal128 is used for high precision decimal representation (in financial and trading applications)

BSON is lightweight, large documents being stored and transferred easily

It is efficient as the stored data is prefixed with length fields and array indices where is the case

Protocol Buffers

Used (Google) for storing and interchanging structured information.

Serves as a basis for a custom remote procedure call (RPC) system that is used for inter-machine communication at Google.

- language-neutral
- platform-neutral
- extensible

It is smaller, faster, and simpler than XML.

Data structure is defined once, then special generated source code can be used to easily write and read structured data using a variety of languages.

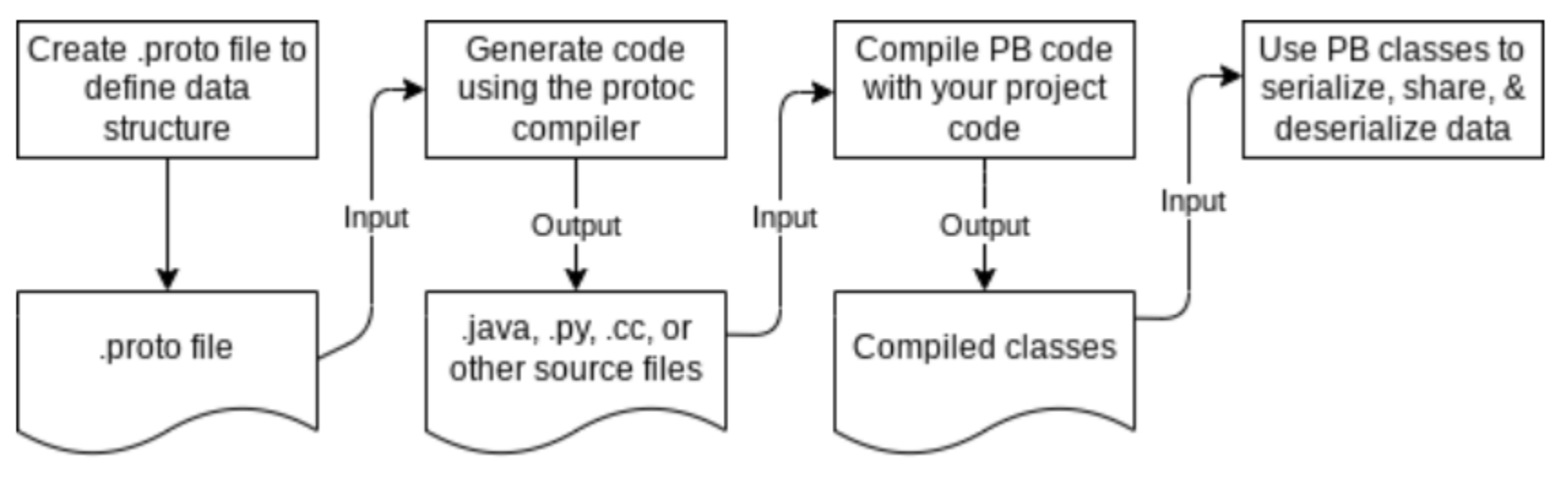
Protocol Buffers

Some advantages of protocol buffers:

- Compact data storage
- Fast parsing
- Availability in many programming languages
- Optimized functionality through automatically-generated classes

A messages can be read by code written in any supported programming language. A Java program on one platform can produce data from one software system, serialize it based on a .proto definition, and then another application in Python, running on a different platform, can extract specific values from that serialized data.

Protocol Buffers



Protobuf workflow

Further reading

- JSON vs XML
- XML

Data Collection and Modeling

Courser 5 - API

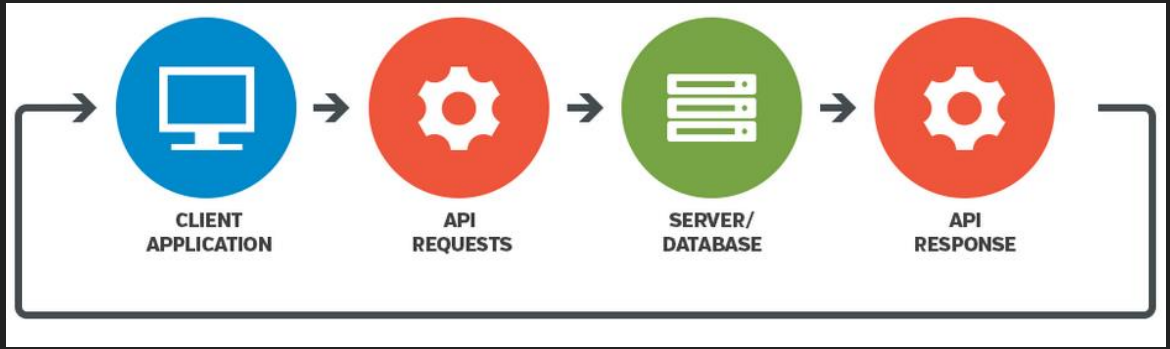
Definition

“An application programming interface, or API, enables companies to open up their applications’ data and functionality to external third-party developers and business partners, or to departments within their companies. This allows services and products to communicate with each other and leverage each other’s data and functionality through a documented interface. Programmers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services.” (IBM)

“API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software.” (RedHat)

How API Works

APIs enable apps or services to communicate with other apps or services without knowing how are they implemented.
An API is a set of rules that state how apps or services communicate with one another (API specification).
APIs sit between an application and a server, acting as an intermediary layer that processes data transfer between systems.



source: Calameo

How API Works

- An app makes an API call (request) to retrieve information.
- Upon receiving a valid request, the API makes a call to the external application or server.
- The server sends a response to the API with the requested information.
- The API transfers the data to the initial requesting application.

APIs enable the abstraction of functionality between interacting systems. An API endpoint decouples the consuming application from the infrastructure that provides a service. As long as the specification for what the service provider is delivering to the endpoint remains unchanged, the alterations to the infrastructure behind the endpoint should not be noticed by the applications that rely on that API.

History of APIs

- Commercial APIs: early 2000s, startups made available products and services but also enabled partners and resellers to extend reach of their platforms. Salesforce, Amazon, eBay
- Social media APIs: mid 2000s; Flickr, Facebook, Twitter
- Cloud APIs: 2006 AWS S3, EC2; enables the use of web APIs to deploy infrastructure
- Mobile apps APIs: after iPhone and Android in 2007; Twilio, Instagram
- API for connected devices: 2010, connect cameras, speakers, sensors to cloud; Fitbit, Nest, Alexa

Types of API

- By management strategies
  - Open API: open source; also known as public APIs
  - Internal API (aka Private API): not accessible to external users; built for company’s internal use to improve integration of internal systems
  - Partner API: exposed only to business partners; usually require some form of authentication
  - Composite API: combine several APIs and allow the access of several endpoints in a call
- By purpose
  - Process API: combine and orchestrate APIs for a specific business purpose
  - System API: provide access to core systems
  - Experience API: exposes services to intended audience

Architectural Styles

- RPC (XML/JSON): Remote Procedure Call; XML or JSON, simple and lightweight
- SOAP: Simple Object Access Protocol; based on XML
- REST: Representational State Transfer; web API architectural principles
- GraphQL: query language for the API; multiple sources in a API call

RPC

XML-RPC (Remote Procedure Call) is a protocol for cross-platform communication. It makes procedure calls by using HTTP as transport and XML as the encoder.

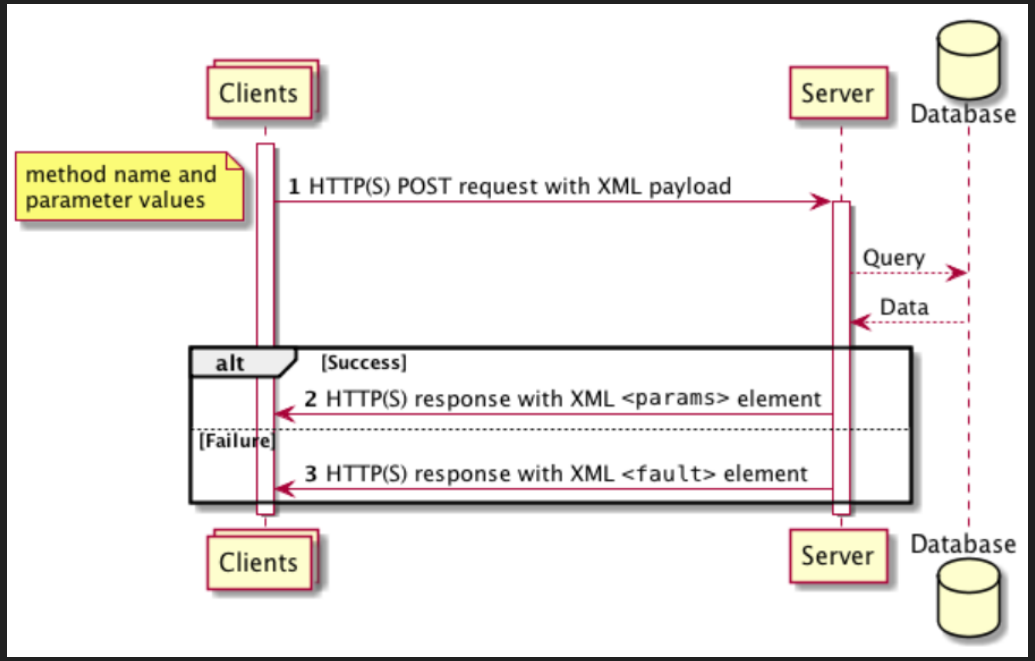
The client makes a call by sending an HTTP request to the server and receives the HTTP response in return.

XML-RPC invokes functions via HTTP request, the functions perform some actions and send hard-coded responses in return.

It does not provide asynchronous model, streaming or security.

RPC

- A client sends an XML document to the server using an HTTP request
- The server either returns
  - HTTP status and XML containing return value
  - HTTP status and XML containing an error



source: Papercut

JSON-RPC

- stateless, lightweight remote procedure call (RPC) protocol.
- defines data structures and the rules for processing them.
- can be used within the same process over sockets, HTTP, or many other message passing environments.
- uses JSON (RFC 4627) as data format.

JSON-RPC

Request
Method: the string that denotes a method. There is a set of reserved names with prefix ‘rpc’, meant for internal RPC calls
Params: can be an object or an array, the value to be passed to method.
Id: A number that identifies the request. In the absence of a response towards a request, the ID will be removed automatically.

```
{"id": 1, "method": "sum", "params": [1,2]}
```

Response
Result: the data returned by invoked method.
Error: the second part, if a problem occurs. It has a code and a message.
Id: refers to the request for which response relates.

```
{"id": 1, "result": 3}
```

SOAP

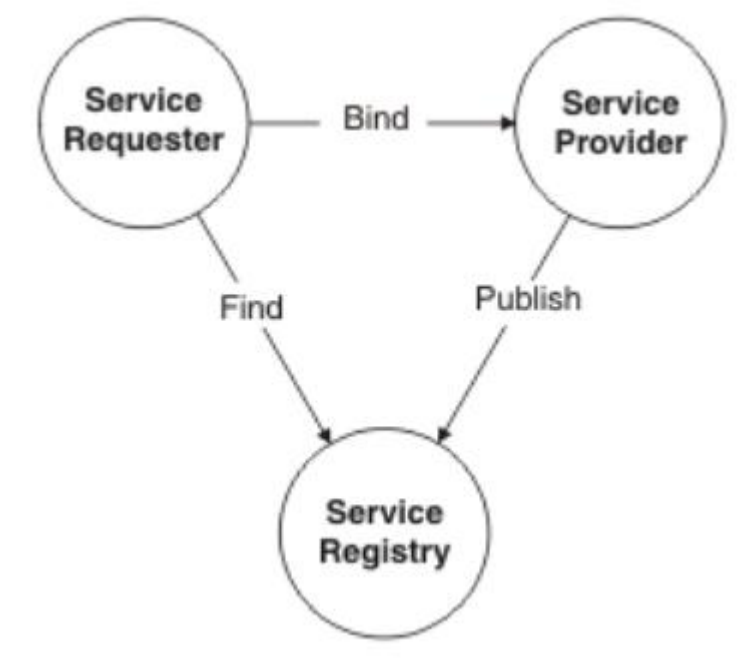
The web service architecture is based on components that play different roles in the interaction between them

Service provider: the collection of software that provides the service
- application
- middleware
- platform

Service requester: the collection of software that requests a service
- application
- middleware
- platform

Service registry: a central location where providers publish service descriptions and requesters can find them
- optional
- allows versioning

SOAP



source: IBM

- Publish: the provider publishes service description in the registry so it can be found
- Find: the requester finds service by description in registry
- Bind: the requester binds with the provider and interacts with the implementation

SOAP

API is a form of agreement between web services on how they are exchanging data.

Service description: a document that service provider uses to communicate the specification to requesters. It is expressed in XML but referred as WSDL (Web Service Description Language). It defines and describes endpoints, data types and actions available.

Strongly follows standards related to message structure, encoding rules, to provide platform independent and language independent communication.

It offers a good level of security.

SOAP Message

- Envelope: the root element in every message; can contain an optional <Header> and a mandatory <Body>
- Header: is used to transmit information related to application to be processed by nodes along the path, extra requirements (authentication)
- Body: contains the request or the response
- Fault: (optional) information about errors

It does not enforce the content of <Header> or <Body>

SOAP Specification

Common web service specifications include:

- Web services security (WS-security): Standardizes how messages are secured and transferred through unique identifiers called tokens.
- WS-ReliableMessaging: Standardizes error handling between messages transferred across unreliable IT infrastructure.
- Web services addressing (WS-addressing): Packages routing information as metadata within SOAP headers.
- Web services description language (WSDL): Describes what a web service does, and where that service begins and ends.

SOAP Example - Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
            <HelloRequest xmlns="http://learnwebservices.com/services/hello">
            <Name>John Doe</Name>
            </HelloRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

https://apps.learnwebservices.com/services/hello

SOAP Example - Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <HelloResponse xmlns="http://learnwebservices.com/services/hello">
         <Message>Hello John Doe!</Message>
      </HelloResponse>
   </soap:Body>
</soap:Envelope>
```

SOAP Example - Error

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
                <soap:Fault>
                <faultcode>soap:Client</faultcode>
                <faultstring>Unmarshalling Error: Unexpected '&lt;' character in element (missing closing '>'?)
 at [row,col {unknown-source}]: [6,13]</faultstring>
                </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

SOAP Extensibility

SOAP has been augmented with standard protocols (WS-*) that enhance it by specifying how things are done:

- WS-Security
- WS-Messaging
- WS-Transactions
- WS-MetadataExchange
- WS-Atomic-Transactions (ACID Compliant)

REST

Representational State Transfer is a software architecture style that uses a stateless communications protocol, usually HTTP.

Most often REST uses JSON to structures data, but also XML, YAML, or any other format that is machine-readable can be used.

It uses the object-oriented programming paradigm of noun-verb and is data-driven

The components of a REST system are:

- client: makes a request for a resource
- server: owner of the requested resources

REST

6 criteria (constraints) for a RESTful API:

- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)

REST

Uniform Interface: only one way of interacting with a server to request a resource so that information is transferred in a standard form.

- Resource-Based: resources are identifiable and separate from the representations sent to the client
- Manipulation of Resources Through Representations: client can manipulate the resource by means of representation of resource as it contains enough information
- Self-descriptive Messages: messages include enough information to describe how the messages should be processed
- Hypermedia as the Engine of Application State (HATEOAS): after accessing a resource the client should be able to use hyperlinks to find all other currently available actions it can take

REST

Stateless: no client information is stored between requests, the necessary state to handle the request is contained within the request itself; the server doesn't store anything related to the session.

The client can include all information for the server as a part of query params, headers or URI.

Cacheable: the response should include information that states if it is cacheable or not and, when yes, the duration it can be cached at the client. For that period, the client will return the data from cache without sending the request again to the server.

A good caching policy can eliminate some client–server interactions, improving availability and performance.

REST

Client-Server: application has a client-server architecture. Client and server can evolve/change independently.

Layered system: the architecture is composed of multiple layers. A layer knows only about the immediate layer; there can be many intermediate servers (invisible to the client) between client and the end server. These servers may be responsible for load-balancing, security shared caches.

Code on demand: server can send executable code to the client, extending client functionality.

REST

Rules for creating REST API endpoints:

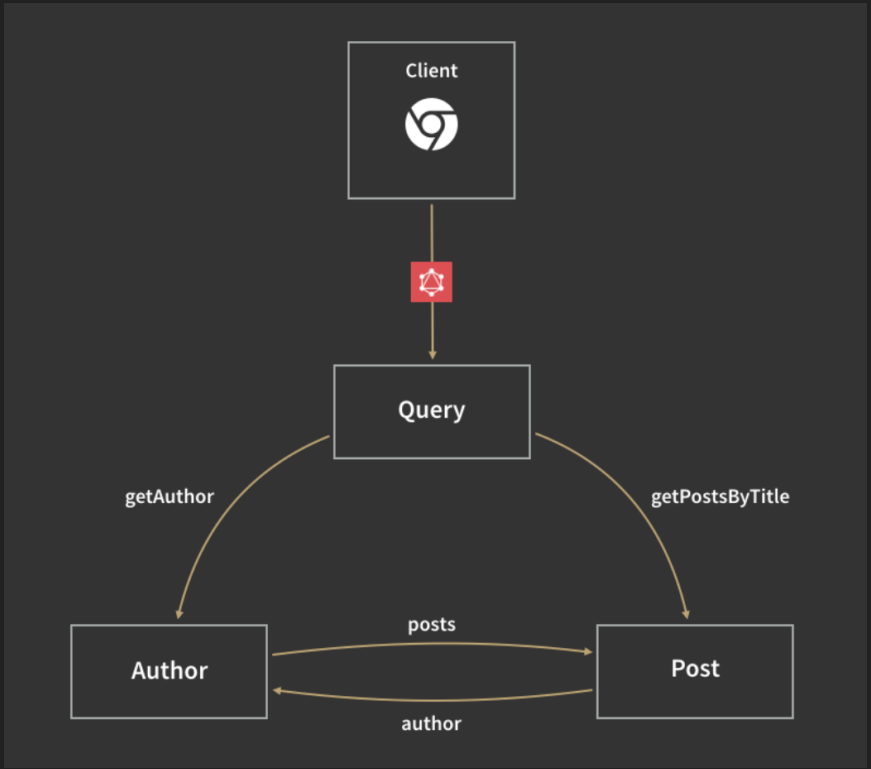
- REST is based on the noun (resource) instead of verb (action). A URI should always end with a noun: /api/users
- HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, GET, PATCH.
- Application should be organized into resources and use HTTP verbs to modify the resources. By looking at the endpoint and HTTP method used one should know what needs to be done.
- Use plurals in URL as naming standard to keep API URI consistent.
- Send a proper HTTP code to indicate a success or error status.

GraphQL

“GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.”

- Is a query language and server-side runtime for APIs; provides complete and understandable description of data
- A GraphQL query defines needed resources and returns predictable results
- Allows retrieval of several resources in one request by following references between them
- GraphQL APIs don't use endpoints but types and fields. All data is accessed using a single endpoint

GraphQL



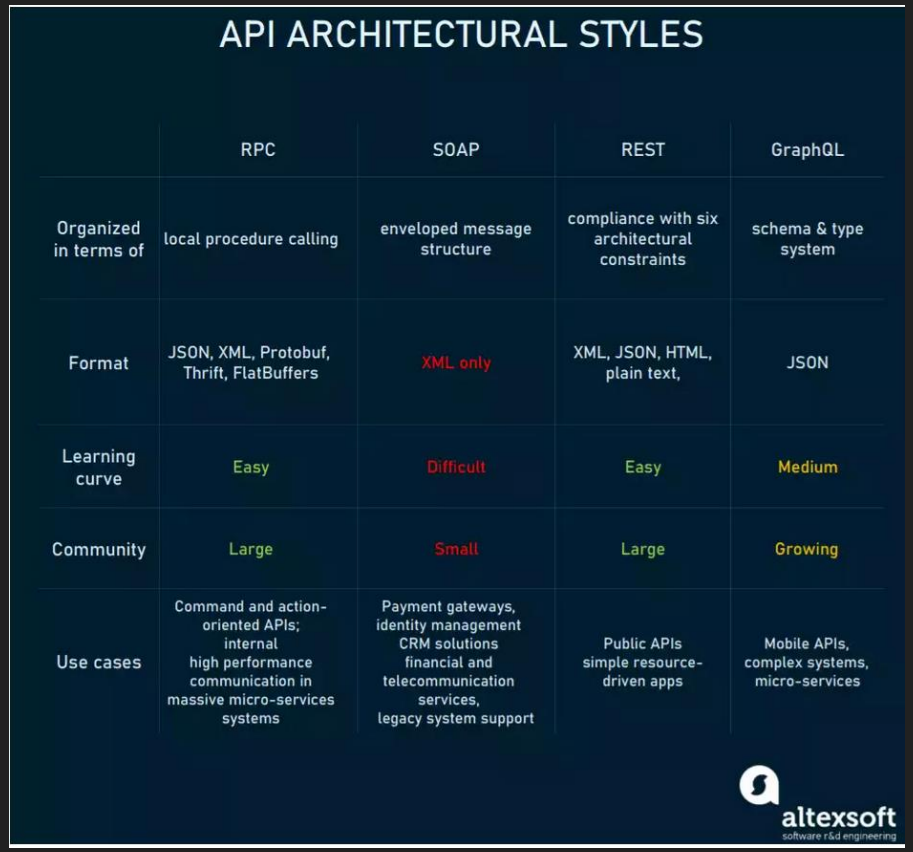

```
type Author {
  id: Int
  name: String
  posts: [Post]
}type Post {
  id: Int
  title: String
  text: String
  author: Author
}type Query {
  getAuthor(id: Int): Author
  getPostsByTitle(titleContains: String): [Post]
}schema {
  query: Query
}
```

source: Apollographql

GraphQL

```
{
  getAuthor(id: 5){
            name
            posts {
            title
            author {
            name # this will be the same as the name above
            }
            }
  }
}
```

RPC vs SOAP vs REST vs GraphQL



Free Web Services

https://dog.ceo/dog-api/

https://swapi.co/

https://graphical.weather.gov/xml/

https://swagger.io/

https://www.predic8.de/public-soap-web-services.htm

Resources

JSON-RPC

SOAP

Services Articles

REST

REST Architecture

GraphQL

GraphQL (Apollo)

Data Collection and Modeling

Course 6 - Web Scraping

Definition

“Web scraping is a term for various methods used to collect information from across the Internet. Generally, this is done with software that simulates human Web surfing to collect specified bits of information from different websites. Those who use web scraping programs may be looking to collect certain data to sell to other users, or to to use for promotional purposes on a website.

Web scraping is also called Web data extraction, screen scraping or Web harvesting.” (Techopedia)

“the extraction and copying of data from a website into a structured format using a computer program” (dictionary.com)

Is it legal?

The action of web scraping isn't illegal if certain rules are followed.

It becomes illegal when extracting non publicly available data.

Data on the internet is considered publicly available if:

- the publisher stated that data is public.
- data access doesn't involve an account creation nor login.
- the robots.txt file for the website doesn't block web scrapers or spiders.

(Legality article)

Is it ethical?

Depends on the purpose of the action; it is considered to be ethical when extracting data from websites that are public and using it without harm and invasion of privacy.

- Request strictly necessary data
- Use a reasonable rate for requests
- Use a User-Agent string
- Use data only to create new data/information

How Web Scrapers Work?

1. Make a HTTP request to a web server: define the URL or set of URLs that the scraper needs to load; the scraper loads the HTML (some advanced scrapers can load also CSS and Javascript elements)
2. Extract and parse web server’s response (website code): it can extract all the data or data selected by the user
3. Save extracted data in desired format: most scrapers can output data in csv or spreadsheet (Excel, OpenOffice) format, but also in JSON

Web Scraper Types

There may be several types of such software; they can be grouped based on 4 main perspectives:

- Pre-built or custom-built
- With or without (comprehensive) user interface
- Software or browser extension
- Local running or cloud based

Pre-built or custom-built

There are a plethora of web scrapers ready to be downloaded (free or commercial ones), with options that would suit all needs. They can allow inexperienced users to just click and select site components they want to scrape, others need some technical knowledge to define how and what to select (Winautomation vs Easy Web Extract).

As the above ones were created, anyone can create its own web scraper provided that has some advanced programming and communication protocols knowledge; more features needed, deeper knowledge is requested.

Custom-built

In order to build a web scraper, there are some generic steps to be followed:

- Define the URLs to be used
- Inspect the page (view page source)
- Identify the components to be extracted
- Write the code (use a library for good productivity)
- Execute the code
- Store the data

User Interface

The range of features offered to the users can vary widely:

- Minimal UI with command line; appropriate for experienced users that use command line regularly
- Powerful UI with entire site rendered so the user can click the components it wants to scrape
- Suggestions and tips to guide the user during the actions to be performed

Software or Browser Extension

Depending on the need for the scraper to be integrated with the browser or not, we can identify:

- Browser extensions; offer features for ad blockers, themes, etc. They can be lighter and simpler to run. They are limited to what a browser can offer.
- Stand alone application: can be downloaded and installed locally, offering advanced features like IP rotation, multithreading, scheduling

Local Running or Cloud Based

The scraping session can run from local computer or from cloud.

- When running locally, it uses the resources of local system and is limited to local network card and connection bandwidth; when having large set of URLs, it may have impact on system’s performance for a while
- Some companies provide scraper that run directly in their cloud environments; the impact on local system is nonexistent; advanced features can be provided (IP rotation, multithreading, etc)

Web Scrapers Use

There are several use cases that span across different industries:

- Price monitoring
- Sentiment analysis
- Real estate listings consolidation
- Lead generation
- Market research
- Email marketing

Scraping and Security

Data scraping can be used to expose sensitive data. The website may not be aware of its data being collected or the scraper may not secure stored data, being accessible by attackers.

- Phishing attacks
- Password cracking

Scraping Techniques

- HTML Parsing: JavaScript is used to extract text and links from HTML pages
- DOM Parsing: it describes the structure and content of an XML; used to access nodes with information of interest
- XPATH: query language for XML documents; can be used to select nodes; can be combined with DOM parsing
- Vertical Aggregation: used to scrape certain business domains (verticals) and aggregate this information; usually they use massive computing power (cloud)

Mitigating Web Scraping

In order to reduce the amount of data that is scrapable, there are few techniques to be used:

- Limit user requests rate: limit the number of requests made from same IP per time interval according to the maximum rate a human is capable to perform (1 page in few seconds vs 100 pages per second)
- Modify HTML regularly: randomized HTML elements modifications
- Apply CAPTCHA: add a task completion easy to perform for humans
- Embed content in media objects: insert content as image instead of text

Robots.txt

A file that is used by crawlers to know what they can access on that specific website.

It manages the crawler traffic based on 3 file types:

- Web page: blocks crawling to specific page, but URL can still be reached via search results; non HTML pages are excluded
- Media file
- Resource file

Robots.txt

It is placed in the root folder of the website.

Consists of at least one rule.

Each defined rule allows or blocks access for specified crawlers to given paths.

All files are implicitly accessible for crawling.

Can be consulted by accessing the website URI and appending /robots.txt to it.

Robots.txt Rules

- Allow access to a single crawler

```
User-agent: My-crawler
Allow: /
User-agent: *
Disallow: /
```

- Allow access to all except one crawler

```
User-agent: My-crawler
Disallow: /
User-agent: *
Allow: /
```

- Disallow crawling for the entire website

```
User-agent: *
Disallow: /
```

- Disallow crawling for an entire folder

```
User-agent: *
Disallow: /events/
Disallow: /personal/work/
```

Robots.txt Rules

- Disallow crawling for a specific web page

```
User-agent: *
Disallow: /myfile.html
```

- Disallow crawling for the entire website except a subfolder

```
User-agent: *
Disallow: /
Allow: /public/
```

- Disallow crawling for files of a specific file type

```
User-agent: *
Disallow: /*.png$
```

- Disallow crawling for URLs that match a string with wildcards

```
User-agent: Googlebot
Disallow: /*.xls$
```

Tutorials

Webscraper.io

Bardeen.ai

Data Collection and Modeling

Course 7 - Data Cleaning

Definition

“Data cleansing or data cleaning is the process of identifying and correcting corrupt, incomplete, duplicated, incorrect, and irrelevant data from a reference set, table, or database.

Data issues typically arise through user entry errors, incomplete data capture, non-standard formats, and data integration issues.” (Precisly)

“Data cleansing, also referred to as data cleaning or data scrubbing, is the process of fixing incorrect, incomplete, duplicate or otherwise erroneous data in a data set. It involves identifying data errors and then changing, updating or removing data to correct them. Data cleansing improves data quality and helps provide more accurate, consistent and reliable information for decision-making in an organization.” (Techtarget)

Motivation

Results of data processing are as good as the data they process.

Studies suggest that 29% of data is inaccurate and leads to loss of revenue and customers. Data cleaning improves accuracy and cand make data more structured and consistent.

- Organized data
- Improved productivity
- Improved mapping
- Avoid mistakes
- Reduce costs

Error Types

There may be various errors in a dataset: invalid, incompatible, inaccurate and corrupt data. The source can be human error, differences in representation, terminology or format.

- Duplicated data
- Typos, invalid, missing data
- Inconsistent data
- Irrelevant data

Overall Process

In a typical scenario, there are at least the following steps:

1. Profiling and inspection: statistics on data set to help identify missing values, erroneous ones, discrepancies
2. Data cleaning: fixing data issues
3. Verification: checks according to quality rules and necessary data standards
4. Reporting: results should be continuously reported with respect to numbers of issues found/fixed and generate data quality trends/visualizations

Data Profiling

Is the process of analyzing data and producing summaries to help identify quality issues and trends.

Typical methodology includes determination of mean, min, max, percentile and frequency distribution; it also includes detection of primary key candidates, functional dependencies and possible foreign keys.

Helps to identify eros like null values (the representation of missing or unknown values), values of of range/scale, values with questionable frequencies.

- Column profiling
- Cross-column profiling
- Cross-table profiling

Data Profiling

Can be:

- Structure discovery: checks for consistency and format (pattern matching)
- Content discovery: relates to data quality; data is processed to adapt to formatting and standardization to be integrated with existing data
- Relationship discovery: detect connections between datasets

Helps by producing:

- Data quality and credibility
- Proactive crisis management
- Predictive decision making
- Organized sorting

Cleaning Process

source: Alteryx

Duplicate removal

- Usually generated during data collection
- When data is combined from multiple sources
- One of the largest area in the cleaning process
- Approach depends on the situation
  - Deal with entire duplicates
  - Deal with partial duplicates

Irrelevant Observations Removal

- Identify relevant data; might need to check correlated values later in the analysis
- Exclude data from analysis, not from source
- Try to use 4 eyes principle when in doubt
- Remove elements like: Personal Identifiable data, URLs, blank spaces, HTML tags, tracking codes
- Dataset becomes more manageable and relevant

Dealing with Incomplete Data

- Can be found as NULL values, “not applicable”, “NA”, “0”, “none”
- Determine if they are plausible or generated by missing information
- Possible approaches:
  - Remove entries associated with it: can induce bias
  - Guess values based on similar data: linear regression, median, etc
  - Flag data as missing: can become information

Filter Outliers

- Data points that are extremely different from the rest in the dataset
- Some can represent true values (natural variation in the population), other may be produced by incorrect data entry or malfunctions/errors
- Can affect the result of analysis
- True outliers should be retained as they represent natural variations
- Can be identified by:
  - numeric techniques: z-scores
  - visual techniques: plots, histograms
  - sorting method
  - Interquartile range (the range of the middle half of the dataset)

Structural Errors

- Typos, capitalization, abbreviations, formatting
- Dependent on data type
- Can be handled using spell-checking, formatting (padding or trimming, adopting a consistent capitalization), using same measurement unit
- Can refer to columns (names) or values in columns

Data Validation

- Use to authenticate data, confirm quality, consistency and (formatting) uniformity
- Is data enough?
- Is uniformly formatted?
- Does data make sense?
- Use a sample to validate or invalidate working theory

Example

#	Id	Name	Birthday	Gender	IsTeacher?	#Students	Country	City
1	111	John	31/12/1990	M	0	0	Ireland	Dublin
2	222	Mery	15/10/1978	F	1	15	Iceland	
3	333	Alice	19/04/2000	F	0	0	Spain	Madrid
4	444	Mark	01/11/1997	M	0	0	France	Paris
5	555	Alex	15/03/2000	A	1	23	Germany	Berlin
6	555	Peter	1983-12-01	M	1	10	Italy	Rome
7	777	Calvin	05/05/1995	M	0	0	Italy	Italy
8	888	Roxane	03/08/1948	F	0	0	Portugal	Lisbon
9	999	Anne	05/09/1992	F	0	5	Switzerland	Geneva
10	101010	Paul	14/11/1992	M	1	26	Ytali	Rome

Missing values

Invalid values

Misfielded values

Misspellings

Uniqueness

Formats

Attribute dependencies

source: Quantdare

Data Quality Report

Based on data exploration and checks performed, a quality report can be produced by answering several questions:

- Are there missing attributes or values?
- Are there deviations? Do they represent outliers or not?
- Are there spelling inconsistencies?
- Was a plausibility check for values performed?
- Was there irrelevant data to be excluded?

Data Cleaning Report

Steps to be performed for data cleaning should be documented for future iterations or future projects:

- What types of noise occurred in data
- What techniques/methods have been used to remove the noise
- Was data removed? Note excluded data

Characteristics of Cleaned Data

The quality and cleanliness of the data set can be measured with respect to:

- Accuracy
- Validity
- Consistency
- Completeness
- Integrity
- Uniformity
- Timeliness

Benefits

Even though data cleaning takes time and effort, it brings benefits both for short and long term:

- Better decision making
- Minimise compliance risk
- Boost results and revenue
- Save time and increase productivity
- Protect reputation

Tools/Vendors

- Open source: DataCleaner, OpenRefine
- Data preparation: Altair, DataRobot, Tableau
- Data management: Ataccama, Informatica, SAP, SAs, Talend
- Data cleaning: Data Ladder, WinPure
- Data quality: Datactics, Experian, Precisely

Data Collection and Modeling

Course 8 - Data Modeling

Definition

“Data modeling is the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures. The goal is to illustrate the types of data used and stored within the system, the relationships among these data types, the ways the data can be grouped and organized and its formats and attributes.

Data models are built around business needs. Rules and requirements are defined upfront through feedback from business stakeholders so they can be incorporated into the design of a new system or adapted in the iteration of an existing one.” (IBM)

Data Models

Is a specification of data structures and rules together with a visual representation of data and relations between elements. It answers to:

- Who
- What
- Where
- Why
- When

Data model should evolve with changes and should be shared

Data Models Types

Can be defined at different levels of abstraction and start at higher level with an overview, becoming more specific, in a top-down approach; usually the following data models are used during the 3 phases of data modeling:

- Conceptual data model
- Logical data model
- Physical data model

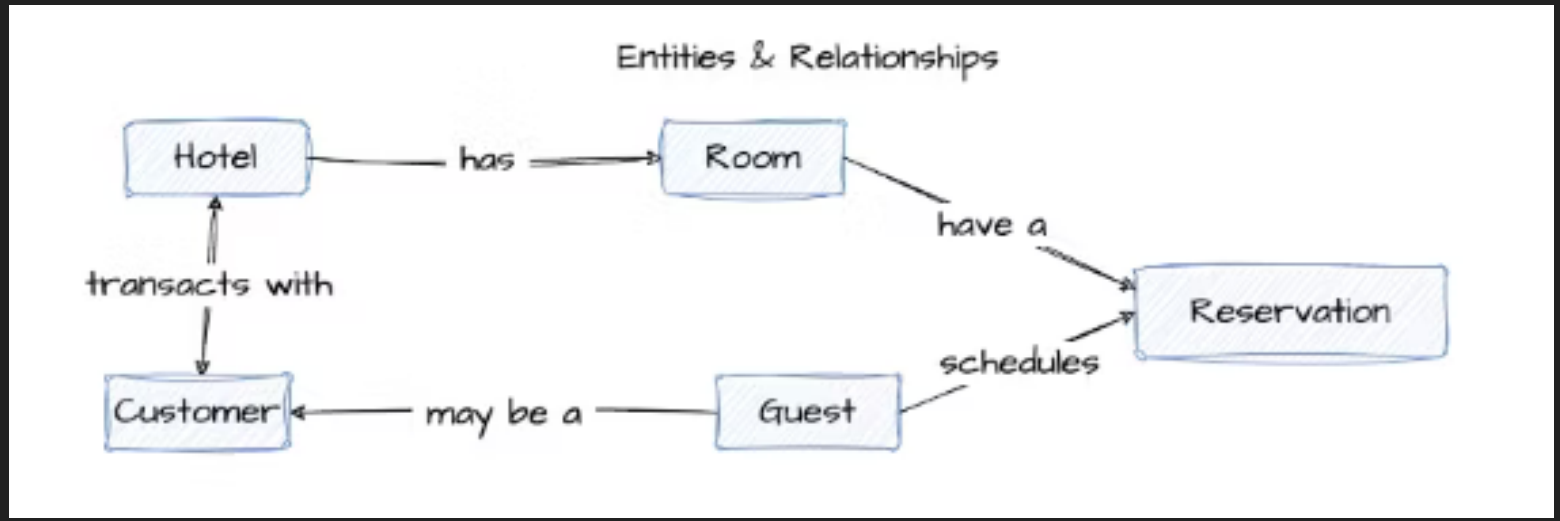
Conceptual Model

identifies data needed in processes and analytics/reporting applications, with business rules and concepts. It doesn't define the data processing flow or physical characteristics.

it is also referred to as domain model and provides an overview of what the system contains, how it is organized, and which business rules are used.

is created during gathering initial project requirements. It includes entity classes, characteristics and constraints, relationships between entities and security and data integrity requirements.

Conceptual Model



Source: www.thoughtspot.com/data-trends/data-modeling

Logical Model

is less abstract; provides more details about data structures (tables), attributes (columns), their data types and lengths, and show the relationships among entities (foreign keys).

the model is independent of database technology or file structure.

can be implemented as relational, multidimensional or NoSQL system; logical data models don’t specify any technical system requirements

Physical Model

provides a schema for the physical storage within a database.

offers a design that can be directly implemented as a relational database, with tables that implement the relationships between entities by means of primary keys and foreign keys.

physical data models can include database management system specific properties, including performance tuning (constraints, indexes, triggers, tablespaces and partitions).

Modeling Process

1. Identify entities: each should be cohesive and discrete from others
2. Identify properties for each entity: unique properties (attributes)
3. Identify relationships between entities: nature of relationship with others; may be specified using UML
4. Map attributes to entities: using some design patterns (OOP design patterns, business patterns)
5. Decide degree of normalization (and assign keys): obtain a good balance between redundancy and performance
6. Validate model: in an iterative process with the business

Types of Modeling

- Hierarchical: tree structure with parent-child relationship, with links and types
- Network: extends previous model to allow more parents for a child record
- Relational: data stored in tables with columns, having specified relations between entities
- ER: formal diagrams to represent entities and their relationships
- OO: derive from OOP, abstractions of domain entities (data and behavior); uses inheritance
- Dimensional: facts to represent measurements and dimensions to represent the context
- Graph: uses nodes to represent each entity instance and edges to depict relationships (can have direction and label); can be property or semantic

Database Design

- Requirements analysis: identifies needed data ,supported operations, applications that will access data
- Conceptual design: high level description of data
- Logical design: translation of conceptual design to DB schema (and model)
- Schema refinement: normalization, redundancy elimination
- Physical design: performance considerations (indexes, partitioning, MV) and possibly some schema redesign

Database Conceptual Modeling

Is a structured business view of data; focused on identifying data used in business processes, not the processing flow

Represents the structure of data independent of software or storage

Defines:

- Non technical data concepts/entities
- High level relationships between entities, without their cardinalities

Consider a database to be a collection of objects/entities

Database Logical Modeling

Contains more details than the conceptual model, with specifications of entity attributes and relationships between these entities.

It is still independent from the DBMS

Defines:

- Entities (tables)
- Attributes (columns)
- Relationships (keys)

Uses business names for entities and attributes

Logical Model

There are 2 popular high-level design models:

- Entity-Relationship (ER)
- Unified Modeling Language (UML)

Both are visual (graphical) models.

UML became more popular with the adoption of OOP

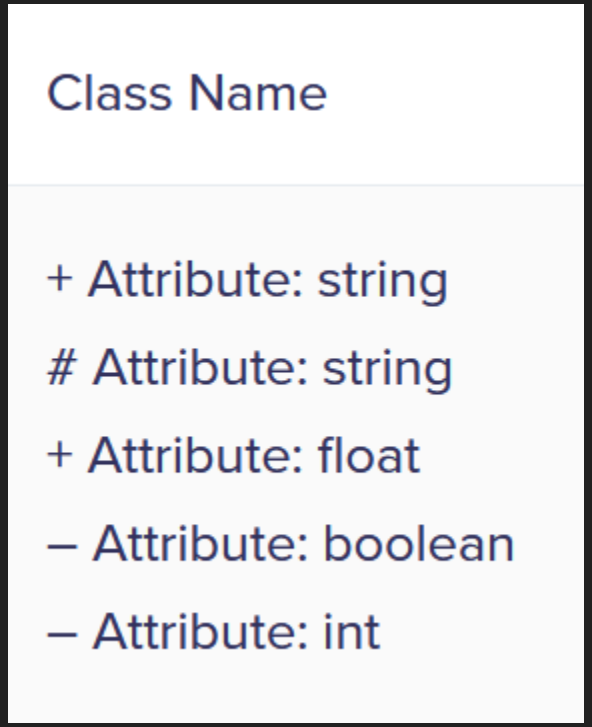
UML Database Modeling

UML class diagrams describe the type of objects and the relationships between them

It contains:

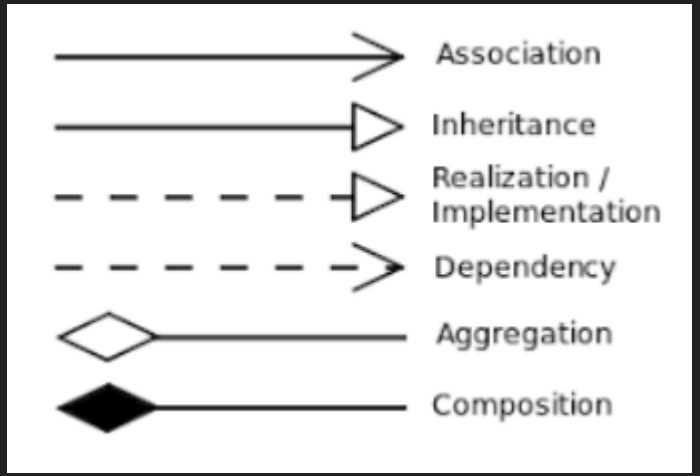
- Classes: have a name, attributes and behaviors/methods
- Relationships: different types represented by different styles of arrows

Classes


Class Name
+ Attribute: string
# Attribute: string
+ Attribute: float
– Attribute: boolean
– Attribute: int

Relationships

- Association: relationship between 2 different classes; can also have cardinality
- Inheritance: relationship between parent and children/descendants
- Realization: relationship between interface and objects that implement it
- Dependency: an object of a class uses an object of another class in its methods and is not stored in any field
- Aggregation: a class is part of another
- Composition: same like aggregate but it gets destroyed when the aggregate is deleted



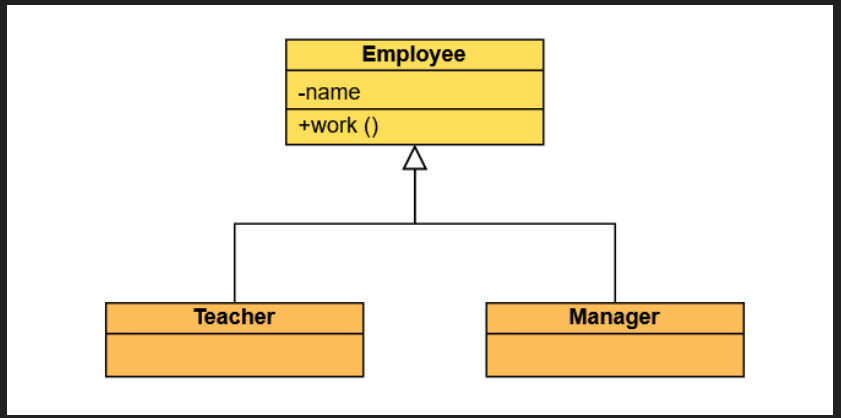
source: visual-paradigm

Relationships

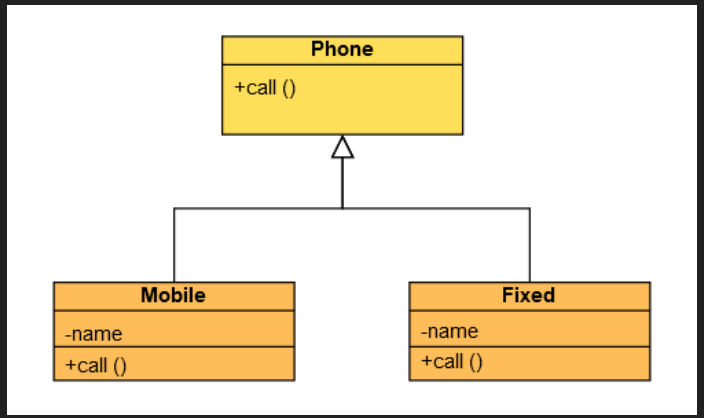
For association relationship there are several possible cardinalities:

Expression	Description
1	Exactly 1
0..1	0 or 1
*	0 or more
0..*	0 or more
1..*	1 or more
5..10	Value in the range specified
1,3,5,7	Exactly one of the enumerated values

Relationships

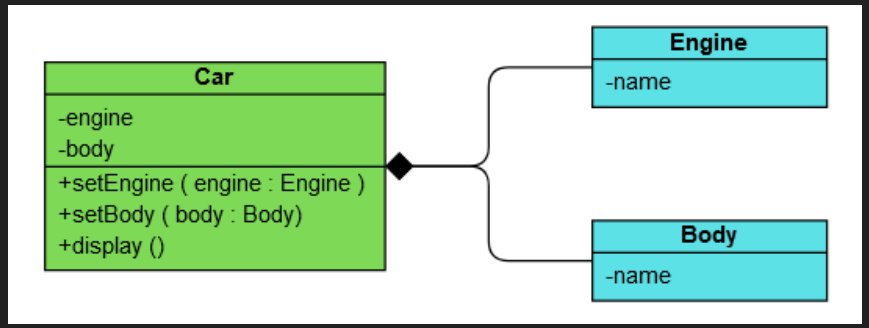


Inheritance

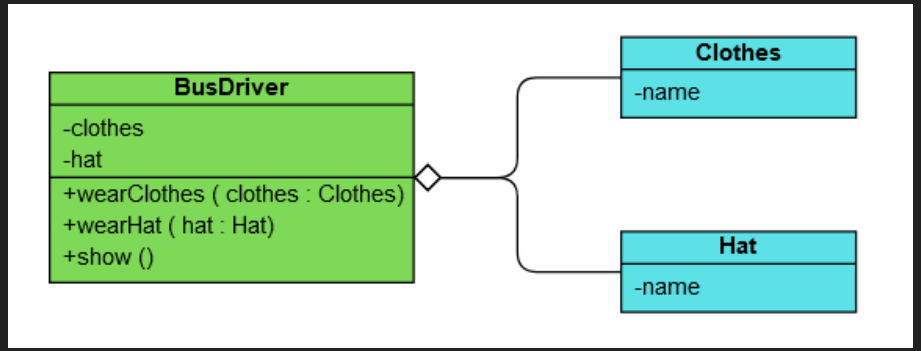


Realization

Relationships



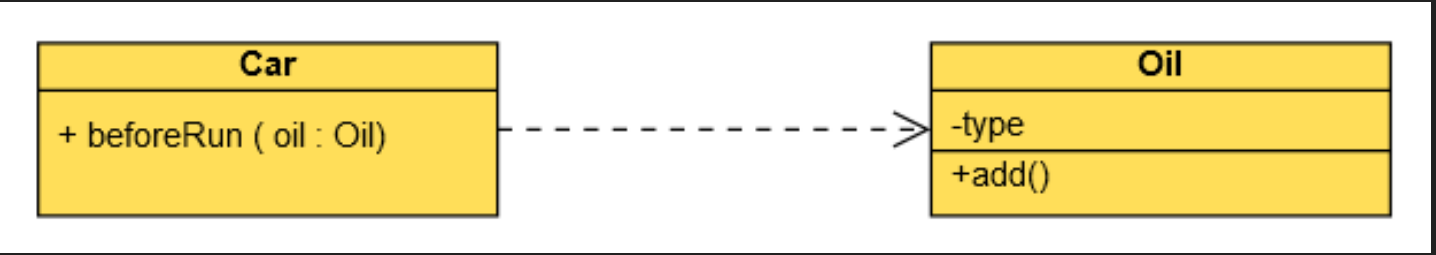
Composition



Aggregation

Relationships

Association



Dependency

Model to Database

Classes:

- become tables
- by convention tables have plural names (as opposed to classes)
- simple attributes are mapped into fields
- composite attributes are mapped into new tables
- derived attributes are not mapped

Model to Database

Associations:

- 1 to 0..1 : 1 table for each class (using FK)
- 1 to many: 1 table for each class (using FK)
- 1 to 1 : 1 table with reunion of the attributes
- many to many : 1 table for each class and a mapping table between them

Inheritance options:

1. A view is created for each subclass
2. A table for each subclass and superclass attributes are found in each one
3. A single table with reunion of superclass and subclass fields and an attribute to indicate the subclass

Principles of Well Design Data Architecture

1. Share data across enterprise: avoid creating silos and supply a transparent view of correlated information from all departments/business functions
2. Provide access to data: build proper interfaces that allow different levels of access
3. Analytics should be close to data: it is more efficient to have analytics apps close to data source than to ship large amounts of data to tools
4. Assure data compliance and governance: comply with all legal requirements but also take into consideration to follow best practices and comply with policies and procedures defined
5. Integrate multiple platforms in data architecture: different services could run on different platforms and be exposed in various ways

Advantages

- Reduce errors in database development (and software)
- Assist the design process at conceptual, logical and physical levels
- Better consistency in documentation and design
- Better communication between teams
- Improved data mapping in the enterprise
- Better app and database performance

Data Modeling Tools

- ErWin
- Enterprise Architect
- ER/Studio
- Lucidchart
- Draw.io
- Visual Paradigm

Data Collection and Modeling

Course 9 - Databases

Definition

“A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.” (Oracle)

“database, also called electronic database, any collection of data, or information, that is specially organized for rapid search and retrieval by a computer. Databases are structured to facilitate the storage, retrieval, modification, and deletion of data in conjunction with various data-processing operations. A database management system (DBMS) extracts information from the database in response to queries.” (Britannica)

DBMS

- Is a software system used to store, retrieve, and query data. It can be seen as an interface between users and databases that facilitates operations to create, read, update, and delete data in the database.
- Provides data security, data integrity, concurrency, and data administration processes.
- Provides a centralized view of data that can be accessed by multiple users from different locations. It can offer several views of a single database schema by limiting what a user can see and how that data can be seen.

DBMS

Components:

- Storage engine: used to interact with the file system at the OS level
- Data dictionary: metadata for all objects that exist in the database
- Access language: API for data access (usually SQL)
- Optimization engine: parses query and translates it into commands for data access
- Query processor: used for running the (optimized) query and returning results
- Lock manager: manages concurrent access
- Log manager: ensures the record of changes (log) is accurate and efficient
- Data utilities: statistics, backup, recovery, integrity, load, download, repair…

DB Types

- Relational
- Object oriented
- Distributed
- Data warehouses
- NoSQL
- Cloud
- Multimodel

Relational DBMS

- also known as SQL DBMS, it stores/represents data as rows in tables with a fixed schema and enforces relationships defined by values in key columns.
- adaptable to most use cases.
- simple data representation
- queries expressed using a simple high level language
- suitable also for non technical users

Relational Model

- based on relations: data collection is organized in tables (that represent relations)
- a relation has a schema and an instance
- the schema describes the relation name, the header (table columns/fields) and the domain for each field; the domain of a field has a name and a set of associated values
- the instance is the table itself

Relational Model

- the attribute values are atomic and scalar
- the records in the instance are distinct
- the records are not ordered !!!
- relations must be normalized to avoid certain anomalies

Relational Model

- Integrity constraints: a set of predefined rules that are used to ensure integrity, validity and consistency of data in the database table
- Rules are evaluated every time a data modification operation is applied (insert, update, delete or alter)
- Types of constraints:
  - Domain: restricts the values for a column
  - Entity integrity: ensures that there are no null primary keys
  - Referential integrity: maintains a valid relationship between 2 tables
  - Key: identify uniquely a row inside a table; there can be several keys, one being primary key (PK)

Relational Database

- Set of relations having distinct names
- Contains also schemas: schemas for the relations in the database
- There is a database instance: collection of relation instances that are valid by conforming with the integrity constraints

SQL DDL

Data Definition Language is a subset of SQL that is used to define the database schema. It is used to create, modify, and delete database objects such as tables, indexes, and views.

DDL statements are typically executed using a database management system, such as PostgreSQL, which provides both a command-line interface or graphical user interface for running statements.

DDL statements include CREATE, ALTER, and DROP. The CREATE statement is used to create new database objects, the ALTER statement is used to modify existing objects, and the DROP statement is used to delete objects.

SQL DDL

```
CREATE TABLE table_name(
column_definition
[, column_definition]
...
...
[, table_restrictions]
)
```

where

```
Column_definition = column_name data_type[(length)] [DEFAULT
value] [column_restriction]
```

SQL DDL

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL
);
```

SQL DDL

In SQL can be defined several restrictions; some of them are:

- NOT NULL - the column cannot have a NULL value
- UNIQUE - all values in the column are unique (one can be NULL)
- PRIMARY KEY - the combination of NOT NULL and UNIQUE. It is used to uniquely identify each row in a table
- FOREIGN KEY - Used to preserve links between tables
- CHECK - the values in the column satisfy the condition
- DEFAULT - Used as default value for the column if no value is specified

SQL DDL

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

SQL DDL

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
        PRIMARY KEY (id)
);
```

SQL DDL

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    age int CHECK (age>=18),
        PRIMARY KEY (id)
);
```

SQL DDL

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    is_admin VARCHAR(3) DEFAULT ‘no’,
        PRIMARY KEY (id)
);
```

SQL DDL

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
     star_date date,
     end_date date,
         PRIMARY KEY (id),
         CONSTRAINT check_dates (start_date < end_date)
);
```

SQL DDL

```
CREATE TABLE user_posts (
    id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    post VARCHAR(255) NOT NULL,
    post_date date,
        PRIMARY KEY (id),
        FOREIGN KEY (user_id) REFERENCES users(id)
);
```

SQL DDL

`ALTER TABLE` is used to add, delete, modify columns in an existing table or to add and drop various constraints on an existing table.

```
ALTER TABLE table_name ADD column_name datatype;
ALTER TABLE users ADD city varchar(100);
```

```
ALTER TABLE table_name DROP COLUMN column_name;
ALTER TABLE users DROP COLUMN city;
```

SQL DDL

```
ALTER TABLE table_name ALTER|MODIFY COLUMN column_name datatype;

ALTER TABLE users ALTER COLUMN city TYPE varchar(150);


ALTER TABLE table_name ALTER COLUMN column_name [SET DEFAULT
value | DROP DEFAULT];

ALTER TABLE table_name ALTER COLUMN column_name [SET NOT NULL|
DROP NOT NULL];

ALTER TABLE table_name ADD CHECK expression;

ALTER TABLE table_name ADD CONSTRAINT constraint_name
constraint_definition;
```

SQL DDL

`DROP TABLE` is used to drop an existing table in a database.

```
DROP TABLE table_name;
```

```
DROP TABLE users;
```

`TRUNCATE TABLE` is used to delete the data inside a table, but not the table.

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE users;
```

Exercise

Model the following entities:

- Customers: name, email, address
- Orders: order_date, products
- Products: name, description, category, price
- Categories: name, description

SQL DML

`INSERT INTO` is used to insert records in a table.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO users (id, name, email)
VALUES (1, 'John Doe', 'jon.doe@email.com')
```

If the values are specified in the order the columns were created in the table, the list of columns can be omitted.

SQL DML

`INSERT INTO` is used to insert records in a table.

```
INSERT INTO table_name (column1, column2, column3, ...)
SUBQUERY;
```

Where `SUBQUERY` is a `SELECT` statement that generates a set of records.

```
INSERT INTO users (id, name, email)
SELECT id, name, email FROM other_users;
```

SQL DML

UPDATE is used to modify existing records in the table.

```
UPDATE table_name

SET column1 = value1, column2 = value2, ...

[WHERE condition];

UPDATE users

SET name = ‘Jane Doe’ WHERE id = 1;
```

SQL DML

UPDATE is used to modify existing records in the table.

`Condition`, if used, specifies which records in the table are changed; if omitted, all records in the table will have the specified columns changed to the specified value.

SQL DML

`DELETE` is used to remove existing records in the table.

```
DELETE FROM table_name [WHERE condition];
```

```
DELETE FROM users WHERE id = 1;
```

`Condition`, if used, specifies which records in the table are removed; if omitted, all records in the table will be erased.

SQL DML

Conditions are based on SQL filters; they are text strings that specify a set of comparisons for data items that are returned when condition holds true.

The syntax is:

```
Field  Operator      Value

[AND | OR | NOT (Field      Operator      Value) ...]
```

where `Field` is the name of a table field, `Operator` is a (comparative) operator, and `Value` is the field value.

SQL DML

Filter Conditions:

```
expression comparison_operator expression

expression [NOT] BETWEEN valmin AND valmax

expression [NOT] LIKE pattern ("%" any substring, "_" one character)

expression IS [NOT] NULL

expression [NOT] IN (value [, value] ...)

expression [NOT] IN (subquery)

expression comparison_operator {ALL | ANY} (

[NOT] EXISTS (subquery)
```

Data Collection and Modeling

Course 10 - Normal Forms; DML

Database Normalization

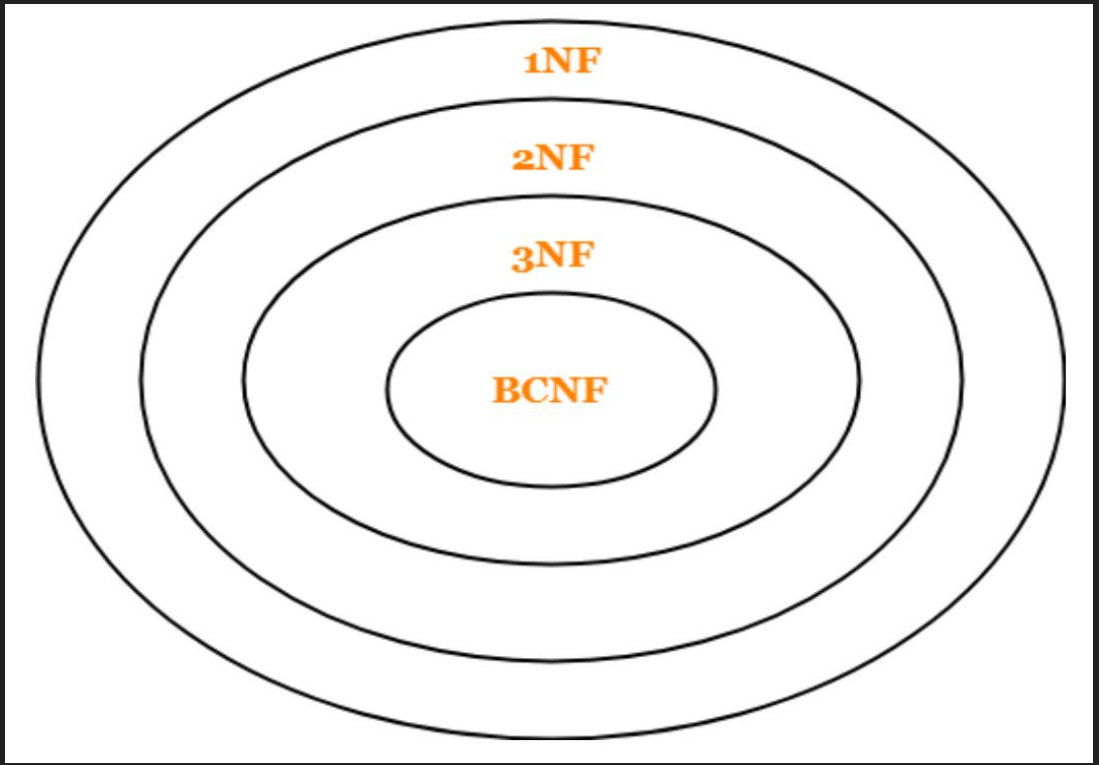
Database Normalization is a method for data organization in the database. It is based on decomposing tables to eliminate data redundancy and anomalies (insert, update and deletion).

It consists of several steps that organize data into tabular form and remove duplicate data from the tables according to rules designed to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency. Each rule is called a normal form.

Anomalies

- Insert: inability to add data to the database due to the absence of other data. For example, if we have the following relation: `Students[id, name, section]` and section has a `NOT NULL` restriction, if the student is not given a section, can’t be saved in the database.
- Update: results from data redundancy and a partial update. If a section is recorded erroneously for several students, if it is updated only for a portion of them, there will be inconsistency.
- Delete: unintended loss of data due to deletion of other data. If a section is deleted, all students associated with it will be deleted (will no longer exist in database).

Normal Forms



Source: codingninjas

1st Normal Form

The following rules have to be satisfied:

1. There are only Single Valued Attributes.
2. Attribute Domain does not change.
3. There is a unique name for every Attribute/Column.
4. The order in which data is stored does not matter.

```
Teachers[id, name, courses]
```

2nd Normal Form

The following rules have to be satisfied:

1. It should be in the First Normal form.
2. it should not have Partial Dependency.

`Grades[student_id, course_id, grade, teacher]`

Partial Dependency is when an attribute in a table depends on a part of the primary key and not on the whole key.

3rd Normal Form

The following rules have to be satisfied:

1. It is in the Second Normal form.
2. It doesn't have Transitive Dependency.

`Students[id, name, city, country, age ]`

Transitive Dependency is when a non-prime attribute depends on other non-prime attributes instead of depending on the prime attributes or primary key.

BCNF

Boyce and Codd Normal Form is a stronger version of 3NF.

A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

1. R must be in 3rd Normal Form
2. For each functional dependency (X → Y), X should be a super Key.

`Classes [student_id, course, teacher]`

A

DML - Select

`SELECT` statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

```
SELECT column1, column2, ...
```

```
FROM table_name;
```

`column1, column2, ...` are the field names (attributes) of the table data is selected from. When selecting all the fields available in the table, the simplified syntax is:

```
SELECT * FROM table_name;
```

DML - Select

```
SELECT [DISTINCT] select_list FROM source
[ WHERE condition(s) ]
[ GROUP BY expression ]
[ HAVING condition ]
[ ORDER BY expression ] ;
```

- `select_list` specifies the fields (or column names) to retrieve.
- `DISTINCT` is used to discard duplicate records and retrieve only the unique records for the specified columns.
- `FROM` clause is the only required clause in the SELECT statement. It specifies the tables to retrieve data from.
- `WHERE` clause filters the rows such that the result contains records that meet some particular conditions.
- `GROUP BY` clause specifies the column list used to aggregate rows.
- `HAVING` clause specifies the specific conditions to group by.
- `ORDER BY` clause specifies a column list to order by (ascending or descending).

Select (PostgreSQL)

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
        [ * | expression [ [ AS ] output_name ] [, ...] ]
        [ FROM from_item [, ...] ]
        [ WHERE condition ]
        [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
        [ HAVING condition ]
        [ WINDOW window_name AS ( window_definition ) [, ...] ]
        [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
        [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST |
LAST } ] [, ...] ]
        [ LIMIT { count | ALL } ]
        [ OFFSET start [ ROW | ROWS ] ]
        [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES }
]
        [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [,
...] ] [ NOWAIT | SKIP LOCKED ] [...] ]
```

Select DISTINCT

```
Students[id, name, group, country_origin, age]

-- return all groups that have students, each group once
SELECT DISTINCT group FROM Students;

-- return all countries of origin there are students from
SELECT DISTINCT country_origin FROM Students;

-- return all countries for students in each group
SELECT DISTINCT group, country_origin FROM Students;
```

Select - WHERE

```
Students[id, name, group, country_origin, age]

-- return all Romanian students
SELECT * FROM Students WHERE country_origin='Romania';

-- return all Romanian students that are younger than 20
SELECT * FROM Students WHERE country_origin='Romania' AND
age<20;

-- return all groups that have Romanian students
SELECT DISTINCT group FROM Students WHERE
country_origin='Romania' OR age>21;
```

Select - WHERE

```
Students[id, name, group, country_origin, age]

-- return all Romanian students that have age specified
SELECT * FROM Students WHERE country_origin='Romania' AND age IS NOT NULL;

-- return all Romanian students that are younger than 20 and older than --
17
SELECT * FROM Students WHERE country_origin='Romania' AND age<20 AND
age>17;

-- return all Romanian students that are younger than 20 and older than --
17
SELECT * FROM Students WHERE country_origin='Romania' AND age BETWEEN 18
AND 19;
```

Select - WHERE

```
Students[id, name, group, country_origin, age]

-- return all students that have name starting with 'AB'
SELECT * FROM Students WHERE name LIKE 'AB%';

-- return all students that have name containing 'AB'
SELECT * FROM Students WHERE name LIKE '%AB%';

-- return all students that have name starting with any
-- letter followed 'AB'
SELECT * FROM Students WHERE name LIKE '_AB%';
```

Select - ORDER BY

```
Students[id, name, group, country_origin, age]

-- return all Romanian students ordered by age
SELECT * FROM Students WHERE country_origin='Romania' ORDER BY
age;

-- return all students that are older than 20 order by group
and, -- within each group by name
SELECT * FROM Students WHERE age>20 ORDER BY group ASC, age ASC;

-- return all distinct student names ordered lexicographically
in -- descending order
SELECT DISTINCT name FROM Students ORDER BY name DESC;
```

Select - IN

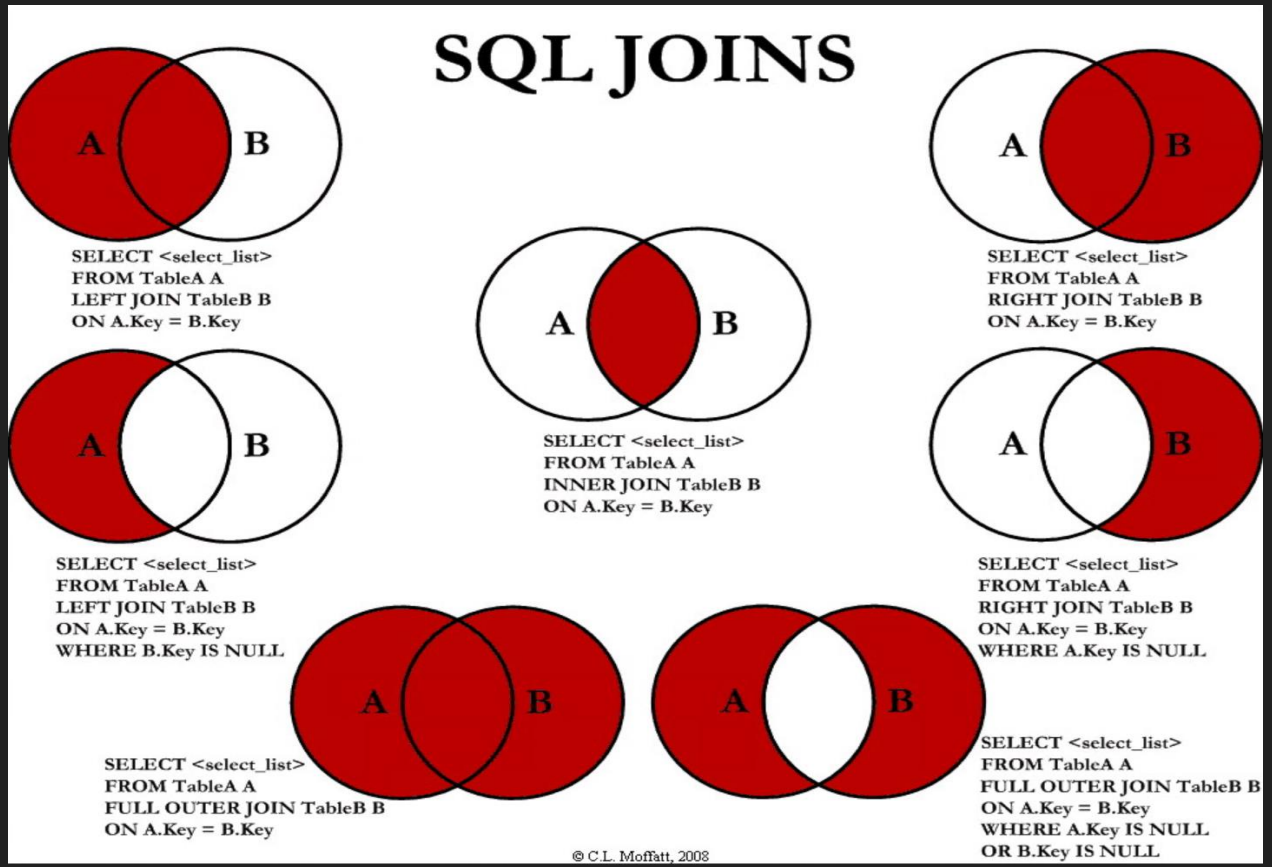
```
Students[id, name, group, country_origin, age]

-- return all Romanian and French students
SELECT * FROM Students WHERE country_origin IN ('Romania',
'France');

-- return all students that are 20 and there have the same
-- name with students being 18 years old
SELECT * FROM Students WHERE age=20 AND name IN (SELECT name
FROM Students WHERE age=18);
```

SQL - JOINS

`JOIN` is a command clause that combines records from two or more tables in a database based on related columns/column expressions between them.

- INNER JOIN: records that have matching values for join condition in both tables
- LEFT [OUTER] JOIN: all records from the left table and matching values from the right table
- RIGHT [OUTER] JOIN: all records from the right table and matching values from the left table
- [FULL | OUTER] JOIN: all records from the right table and left table, once for the matching values

SQL - JOINS



source : codeproject.com

INNER JOIN

```
Students[id, name, group_id, country_origin, age]
Groups[id, code, description]

-- return student names, their age and group codes for
-- Romanian students
SELECT Students.name, Students.age, Groups.code FROM Students, Groups
WHERE Students.group_id=Groups.id AND Student.country_origin='Romania';

SELECT s.name, s.age, g.code FROM Students s, Groups g where
s.group_id=g.id AND s.country_origin='Romania';

SELECT s.name, s.age, g.code FROM Students s INNER JOIN Groups g ON
s.group_id=g.id AND s.country_origin='Romania';

SELECT s.name, s.age, g.code FROM Students s INNER JOIN Groups g ON
s.group_id=g.id WHERE s.country_origin='Romania';
```

LEFT JOIN

```
Students[id, name, group_id, country_origin, age]
Groups[id, code, description]

-- return student names, their age and group codes (if
-- present) for Romanian students, even if they are not
-- registered in a group
SELECT s.name, s.age, g.code FROM Students s LEFT JOIN Groups
g ON s.group_id=g.id AND s.country_origin='Romania';
```

RIGHT JOIN

```
Students[id, name, group_id, country_origin, age]
Groups[id, code, description]

-- return student names, their age and group codes (if
-- present) for Romanian students, even if they are not
-- registered in a group
SELECT s.name, s.age, g.code FROM Groups g RIGHT JOIN
Students s ON s.group_id=g.id AND s.country_origin='Romania';
```

FULL JOIN

```
Students[id, name, group_id, country_origin, age]
Groups[id, code, description]

-- return student names, their age and group codes (if
-- present) for Romanian students, even if they are not
-- registered in a group and also group codes with no
-- students registered
SELECT s.name, s.age, g.code FROM Groups g FULL JOIN Students
s ON s.group_id=g.id AND s.country_origin='Romania';
```

Self JOIN

A (regular) join where a table is joined with itself

```
Students[id, name, group, country_origin, age]

-- return all students with same name and different age
SELECT * FROM Students s1, Students s2 WHERE s1.name=s2.name
AND s1.age<>s2.age;

-- return all students with same name and different age
SELECT * FROM Students s1 INNER JOIN Students s2 ON
s1.name=s2.name AND s1.age<>s2.age;
```

SQL - UNION

Operator that combines two or more result sets into one; some conditions have to be met:

- Every result set must have the same number of columns
- Corresponding columns must have compatible data types
- Each result set has to have same column names in same order

There are two versions: `UNION` and `UNION ALL`

- Simple form of the operator selects distinct rows
- The version with `ALL` selects all rows

SQL - UNION

```
Students[id, name, group, country_origin, age]
Teachers[id, name, department, age]

-- return persons
SELECT name, age FROM Students
UNION
SELECT name, age FROM Teachers;

-- return all persons
SELECT name, age FROM Students
UNION ALL
SELECT name, age FROM Teachers;
```

SQL - GROUP BY

A clause that is used to generate summary rows based on grouping rows using identical values one or several columns; the result will contain one row per group (set of distinct values). The restriction is that any column in the SELECT list that is not part of an aggregate expression must be included in the GROUP BY list.

Usual aggregate functions used:

- COUNT()
- MIN(), MAX()
- SUM(), AVG()

SQL - GROUP BY

```
Students[id, name, group, country_origin, age]

-- return total number of students
SELECT count(id) FROM Students;
SELECT count(*) as total_no FROM Students;

-- return number of students from each group
SELECT group, count(id) FROM Students GROUP BY group;

-- return number of students from each group, by country of
origin
SELECT group, country_origin, count(id) FROM Students GROUP
BY group, country_origin;
```

SQL - GROUP BY … HAVING

Is a clause that is similar to where but applied on the grouping result; WHERE clause can be used only before GROUP BY

```
Students[id, name, group, country_origin, age]

-- return number of non Romanian students from each group
SELECT group, count(id) FROM Students WHERE
country_origin<>'Romania' GROUP BY group;

-- return number of non Romanian students from each group, by
country of origin
SELECT group, country_origin, count(id) FROM Students GROUP
BY group, country_origin HAVING country_origin<>'Romania';
```

Self JOIN GROUP BY - Exercise

Students[id, name, group, final_grade]

Using Self JOIN and GROUP BY, determine the ranking for each group of students based on their final_grade; the ranking should start with highest final grade, and, when the same grade, the order should be alphabetical.

Self JOIN GROUP BY - Exercise

`Students[id, name, group, final_grade]`

Using `Self JOIN` and `GROUP BY`, determine the ranking for each group of students based on their `final_grade`; the ranking should start with highest final grade, and, when the same grade, the order should be alphabetical.

Self JOIN GROUP BY - Exercise

```
Students[id, name, group, final_grade]

-- obtaining ranking of grades by group
SELECT s1.group, s1.final_grade, count(*)
FROM Students s1 INNER JOIN Students s2
on s1.group=s2.group and s1.final_grade <= s2.final_grade
group by s1.group, s1.final_grade
order by 1,3
-- one more time Students table is needed for final result
```

Self JOIN GROUP BY - Exercise

```
Students[id, name, group, final_grade]

-- obtaining final ranking
SELECT s4.group, s4.name, s4.final_grade, s3.pos FROM
(SELECT s1.group, s1.final_grade, count(*) as pos
FROM Students s1 INNER JOIN Students s2
on s1.group=s2.group and s1.final_grade <= s2.final_grade
group by s1.group, s1.final_grade
) s3
INNER JOIN Students s4 on s3.group=s4.group and
s3.final_grade=s4.final_grade
order by 1,4,2
```

Data Collection and Modeling

Course 11 - SQL DML Part 2

SQL - GROUP BY (From previous course)

A clause that is used to generate summary rows based on grouping rows using identical values one or several columns; the result will contain one row per group (set of distinct values). The restriction is that any column in the SELECT list that is not part of an aggregate expression must be included in the GROUP BY list.

Usual aggregate functions used:

- COUNT()
- MIN(), MAX()
- SUM(), AVG()

SQL Aggregate Extensions

`GROUP BY` clause can be used to obtain aggregation based on a set of columns; if we need to compose a result set with similar columns from the same data source based on different lists, the easiest way is to use `UNION` to reunite all distinct result sets:

```
Students[id, name, group, country_origin, age]

-- return number of students from each group
SELECT group, count(id) FROM Students GROUP BY group;

-- return number of students from each country of origin
SELECT country_origin, count(id) FROM Students GROUP BY 1;

-- return number of students group and country of origin
SELECT group, country_origin, count(id) FROM Students GROUP BY
1,2;
```

SQL Aggregate Extensions

The previous result sets are not compatible as they have different number of columns and may contain columns with different data types; in order to make them compatible:

```
-- return number of students from each group
SELECT group, NULL, count(id) FROM Students GROUP BY 1
UNION ALL
-- return number of students from each country of origin
SELECT NULL, country_origin, count(id) FROM Students GROUP BY 2
UNION ALL
-- return number of students group and country of origin
SELECT group, country_origin, count(id) FROM Students GROUP BY
1,2;
```

SQL Aggregate Extensions

SQL `GROUPING SETS` is an extension of the `GROUP BY` clause that allows to specify multiple grouping sets within a single query. This feature enables performing aggregated calculations at different levels of granularity within the same result set. It is particularly useful when generating subtotals and totals for different combinations of columns.
The previous union of result sets can be solved in a more performant way with much less code using the `GROUPING SETS` clause:

```
SELECT group, country_origin, count(id) FROM Students GROUP BY GROUPING
SETS(
        (group, country_id),
        (group),
        (country_origin)
-- ,() if we want also a grand total
);
```

SQL Aggregate Extensions

In order to see if a row is produced as a result of a specific grouping set, there is a function, `GROUPING`, that returns 0 if the argument is member of the current grouping set and 1 otherwise:

```
SELECT
GROUPING(group) uses_group,
group, country_origin, count(id) FROM Students GROUP BY
GROUPING SETS(
       (group, country_id),
       (group),
       (country_origin)
);
```

SQL Aggregate Extensions

The `GROUPING` function can be also used in the `HAVING` clause to filter the result set:

```
SELECT group, country_origin, count(id) FROM Students GROUP
BY GROUPING SETS(
       (group, country_id),
       (group),
       (country_origin)
HAVING GROUPING(group)=0
);
```

SQL Aggregate Extensions

The `GROUP BY` clause has another additional clause that can define several grouping sets as a hierarchy (that makes sense) between lists of columns.
`GROUP BY ROLLUP` is a clause used to generate subtotals and grand totals for a set of columns in a result set. It is an extension of the `GROUP BY` clause useful for creating summary reports with hierarchical aggregates. The `ROLLUP` operation generates a result set that includes not only the usual grouping, but also additional rows that represent various levels of subtotal and grand total.

```
SELECT group, country_origin, count(id) FROM Students GROUP BY
ROLLUP(group, country_id);
```

It will generate groups for the next sets:

```
(group, country_origin)
(group)
()
```

SQL Aggregate Extensions

`CUBE` is a subclause of `GROUP BY` clause that allows the generation of multiple grouping sets based on all possible grouping sets specified by columns list:

```
SELECT group, country_origin, count(id) FROM Students GROUP BY CUBE(group, country_id);
```

It will generate groups for the next sets:

```
(group, country_origin)
(group)
(country_origin)
()
```

SQL Window Functions

“A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.” (PostgreSQL documentation)

Unlike aggregate functions (e.g., `SUM`, `AVG`, `COUNT`), which operate on entire groups of rows, window functions operate on a "window" of rows defined by an `OVER` clause.

SQL Window Functions

Aggregate functions aggregate data from each group of distinct values into a single row using `GROUP BY`.

Window functions operate in a similar fashion on each group of distinct values but preserves the initial number of rows from the group.

```
window_function(arg1, arg2,..) OVER (
   [PARTITION BY partition_expression]
   [ORDER BY sort_expression [ASC|DESC] [NULLS {FIRST|LAST}])
```

SQL Window Functions

Components:

`window_function`: The window function itself (e.g., `SUM`, `AVG`, `ROW_NUMBER`, etc.).

`PARTITION BY`: Divides the result set into partitions to which the window function is applied separately. It's an optional clause, and if omitted, the window function is applied to the entire result set.

`ORDER BY`: Specifies the order of rows within each partition.

`ROWS BETWEEN n PRECEDING AND m FOLLOWING`: Defines the window frame, indicating the range of rows to which the window function is applied relative to the current row.

SQL Window Functions

There are aggregate functions that can be used as window functions:

```
AVG(), COUNT(), MIN(), MAX(), SUM()
```

But there are also analytic functions specific to this context:

```
ROW_NUMBER()

RANK(), DENSE_RANK()

NTILE()

LEAD(), LAG()

FIRST_VALUE(), LAST_VALUE()
```

SQL Window Functions

```
Students[id, name, group, country_origin, finale_grade]

-- return the students with their position in their group
-- based on final grade
SELECT name, group, RANK() OVER (PARTITION BY group ORDER BY
finale_grade desc) as position
FROM Students;

SELECT name, group, DENSE_RANK() OVER (PARTITION BY group
ORDER BY finale_grade desc) as position
FROM Students;
```

SQL Window Functions

```
Students[id, name, group, country_origin, finale_grade]

-- return the students with their position in their group
-- based on final grade and the difference from 1st
SELECT name, group, RANK() OVER (PARTITION BY group ORDER BY
finale_grade desc) as position, FIRST_VALUE(final_grade) OVER
(PARTITION BY group ORDER BY finale_grade desc) -
finale_grade as difference
FROM Students;
```

Data Collection and Modeling

Course 11 - HLPL Data Access

Introduction

Data collections stored in files or databases are not of much use unless they are accessible for various users and applications; up to this point we have seen how to store data in files and databases, and access this data from SQL. When data needs to be accessed by non experienced users, usually we develop applications that offer some user interface(s) for CRUD operations on data, but also generating reports and visualizations. Applications are written using High Level Programming Language(s) (HLPL) and, for data access, we use some drivers/adapters.

Data Access using Python

Many times data is stored and accessed in a combination of:

- Files, local or remote/shared
- APIs
- Databases
- Datasets (publicly available)

Files

There are several scenarios when data is shared using files; from Python there are several ways to work with files:

- using specific libraries/modules:
  - csv: csv library
  - excel: xlrd library, openpyxl library, xlwings library
  - text: standard wrapper over OS API, fileinput library, pickle library,
  - json: json library
  - xml: xml library (xml.dom.minidom)
- using pandas library

Files with pandas

“pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language”

Pandas is a library that offers tools/functions for working with data sets and analyze, clean, manipulate and explore data

Pandas can organize data in:

- Series: one dimension array that stores data of any type (homogeneous), size immutable
- DataFrames: 2 dimensional array (table with rows and columns), size mutable

Files with pandas

In order to work with pandas, it needs to be imported:

```
#importing pandas library

import pandas as pd
```

In what follows next we focus on DataFrames as they are more appropriate for several use cases; a DataFrame can be created by explicitly calling a constructor (`pd.DataFrame`) or as a result of a I/O function:

- `pd.read_excel`
- `pd.read_csv`
- `pd.read_sql`

CSV Files with pandas

When importing a csv file into pandas, there can be specified:

- File path: the path where the file is stored
- Header: the row used for specifying column names in DataFrame (default 0)
- Column delimiter: specify a custom delimiter (default ,)
- Index column: set the columns used as index of the DataFrame(default None)
- Select columns: specify columns to import as names or indices
- Omit rows: number of rows to skip at start, at end, nr rows to read
- Missing values: specify which values to be considered NaN
- Change values: convert column values as specified
- Compress/decompress file: (default infer)

CSV Files with pandas

```
#importing pandas library

import pandas as pd

#reading csv file

df=pd.read_csv('../data.csv',header=None, sep=';',nrrows=100)

df

#writing data to csv file

df.to_csv('new_data.csv')
```

Databases

Python offer several possibilities to connect to databases:

- Generic database interfaces and APIs
  - ODBC
  - ADO
- Database interfaces for RDBMSs
  - General purpose
  - Data warehouses
  - Embedded into apps
- Non-relational databases
- Native Python databases

Databases

In order to connect to a database we might use a database driver built to enable a database connections from other systems, or we might use ORMs.

Drivers:

- Relational: PyMySQL, psycopg2, mysqlclient
- No-SQL: redis-py, cassandra-python-driver, PyMongo
- Embedded: SuperSQLite

ORM:

- SQLAlchemy, PonyORM, The Django ORM

Databases

In order to work with a DBMS using a driver we need to:

- specify connection details: username, password, host, port, database name
- Connect using `connect()` method
- Create a cursor object using `cursor()` method to store records
- Use `execute()` method to run SQL queries and return results
- Extract results in the cursor using one of the methods:
  - `fetchone()`
  - `fetchmany()`
  - `fetchall()`
- Close cursor and connection: `cursor.close()` and `connection.close()`

```
import psycopg2
# Connect to an existing database
connection = psycopg2.connect(user="postgres",
password="123456", host="127.0.0.1", port="5432",
database="dcm_db")

# Create a cursor to perform database operations
cursor = connection.cursor()

# Executing a SQL query
cursor.execute("SELECT * FROM students;")
# Fetch result
record = cursor.fetchone()
print("Student: ", record, "\n")

cursor.close()
connection.close()
```

```
import psycopg2
# Connect to an existing database
connection = psycopg2.connect(user="postgres",
password="123456", host="127.0.0.1", port="5432",
database="dcm_db")

# Create a cursor to perform database operations
cursor = connection.cursor()

# Executing a SQL query
cursor.execute("INSERT INTO students values (2,’Jane
Doe’,222,’USA’, 20);")
# Persist changes
connection.commit()

cursor.close()
connection.close()
```

Databases - ORM

SQLAlchemy is an ORM that allow complex queries to be expressed easier than in other ORMs; it considers the database to be a relational algebra engine instead of just a collection of tables. Rows can be selected from tables, views, joins and other select statements; any of these units can be composed into a larger structure. SQLAlchemy's expression language builds on this concept from its core.

Databases - ORM

```
from sqlalchemy import create_engine

db_conn = "'postgresql://postgres:123456@localhost:5432/dcm_db'"
db = create_engine(db_conn)

# Create
db.execute("CREATE TABLE IF NOT EXISTS students (id int, name varchar(100), group_id int,
country_origin varchar(100), age int)")
db.execute("INSERT INTO students (id, name, group_id, country_origin, age) VALUES (1,
'John Doe', 222, 'Romania', 19)")

# Read
result_set = db.execute("SELECT * FROM students")
for r in result_set:
        print(r)

# Update
db.execute("UPDATE students SET group_id=333 WHERE id=1")

# Delete
db.execute("DELETE FROM students WHERE age=19")
```

Databases - pandas

Pandas is specially built for data preprocessing and offers a more user friendly way of working with data than SQL; when data is stored in relational databases, it has to be retrieved and stored in pandas dataframe:

- Specify a connection string
- Create engine using connection string
- Connect to db
- Load data into a DataFrame using a SELECT query
- Process data from DataFrame
- Close connection

```
# read data from a PostgreSQL table and load into a pandas DataFrame

import psycopg2
import pandas as pds
from sqlalchemy import create_engine

# Create an engine instance
db_conn = "'postgresql://postgres:123456@localhost:5432/dcm_db'"
db = create_engine(db_conn)

# Connect to PostgreSQL server
dbConnection      = db.connect()

# Read data from PostgreSQL database table and load into a DataFrame instance
dataFrame         = pds.read_sql_query("SELECT * FROM students;", dbConnection)

# Profile/summarize DataFrame data
dataFrame.describe()

# Close the database connection
dbConnection.close()
```

```
# read data from a PostgreSQL table and load into a pandas DataFrame

import psycopg2
import pandas as pds
from sqlalchemy import create_engine

# Create an engine instance
db_conn = "'postgresql://postgres:123456@localhost:5432/dcm_db'"
db = create_engine(db_conn)

# Connect to PostgreSQL server
dbConnection      = db.connect()

# Read data from PostgreSQL database table and load into a DataFrame instance
dataFrame         = pds.read_sql_table("students", dbConnection)

# display some DataFrame data
dataFrame.head()

# Close the database connection
dbConnection.close()
```

Databases - pandas

In the previous example we used read_sql_table and read_sql_query methods that can be replaced by read_sql method that is a wrapper around these 2 methods and it actually invokes one or the other based on the first argument supplied, a select query or a table name:

```
# Read data from PostgreSQL database table and load into a DataFrame instance
dataFrame          = pds.read_sql("SELECT * FROM students;", dbConnection)

will call read_sql_query

# Read data from PostgreSQL database table and load into a DataFrame instance
dataFrame          = pds.read_sql("students", dbConnection)

will call read_sql_table
```

Databases - pandas

When using `DataFrame` we usually store all the data in such an object; during the processing steps we apply all necessary changes to `DataFrame` and then we need to persist it into the database. In order to do that we can use `to_sql()` function:

```
# Create an engine instance
db_conn = "'postgresql://postgres:123456@localhost:5432/dcm_db'"
db = create_engine(db_conn)
# Connect to PostgreSQL server
dbConnection    = db.connect()

# prepare dataFrame data ...

dataFrame.to_sql("students", dbConnection, if_exists='replace', index=false)
# Close the database connection
dbConnection.close()
```

Resources

Bulk insert performance