

# *Data Engineering* *Interview Series - 4* **SQL**

Anju Mercian

Data Engineering Consultant  
<https://amalphonse.github.io/>

February 2023

# Agenda

- About Me
- Portfolio Projects
- Data Engineering (DE)
- Technical Skills
- Data Engineering Steps
- SQL Fundamentals
- SQL Commands
- Q&A

# About Anju Mercian

- Graduated from Syracuse University, NY with a Master's degree in Computer Engineering
- Worked at Intel for 5 years as an Infrastructure Engineer
- Worked at VMware for one year as a Project Manager
- Worked at Intuitive surgical for one year as a Business System Analyst
- Worked at Omdena as a Platform Engineer
- Currently working as a Data Engineering Consultant @ Meta
- Also a Data Engineering blogger [link](#)

# *Goal of the series*

- To help as a cheat sheet for Data Engineering (DE) Interview Prep.
- To give an insight on how DE Interviews are
- Topics planned to be covered in the series:
  - SQL Questions
  - Python Questions
  - Data Modeling/ Data Design
  - Data Pipelines

# ***What Is Data Engineering?***

“Data engineering is the complex task of making raw data usable to data scientists and groups within an organization. Data engineering encompasses numerous specialties of data science.

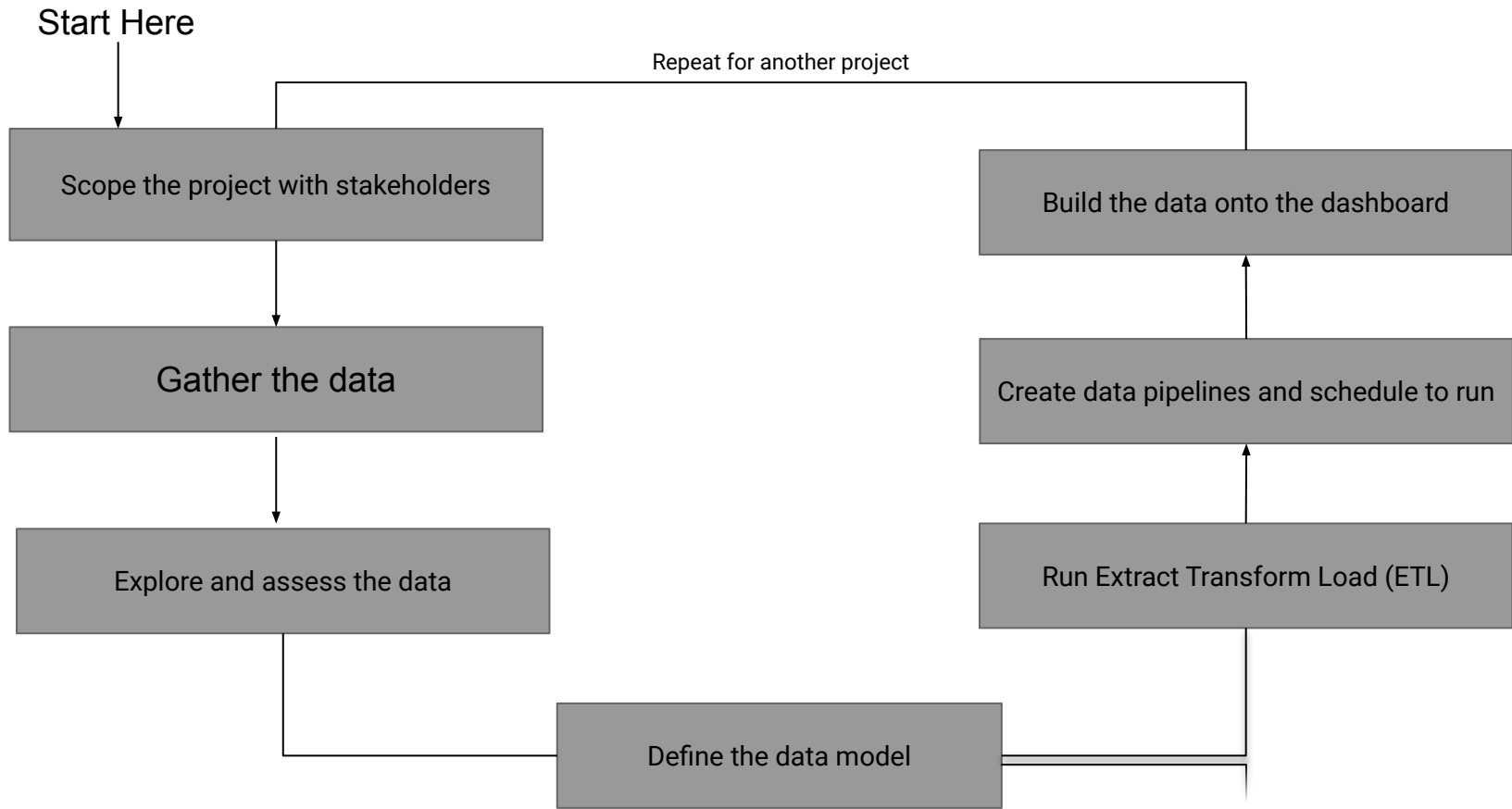
In addition to making data accessible, data engineers create raw data analyses to provide predictive models and show trends for the short- and long-term. Without data engineering, it would be impossible to make sense of the huge amounts of data that are available to businesses.”

[Source](#)

# *Who Is a Data Engineer?*

“Data engineers work in a variety of settings to build systems that collect, manage, and convert raw data into usable information for data scientists and business analysts to interpret. Their ultimate goal is to make data accessible so that organizations can use it to evaluate and optimize their performance.”

[Source](#)



# *Technical Skills*



EMR  
RedShift  
S3





# *Data Engineering Interview*

- There are usually around 3-4 rounds of interviews.
- First will be a technical round with SQL and Python questions
  - Here how quick you are to solve problems are important
- Second and third rounds will be for Data Modeling/ Data Design and some Data Pipelines.

# ***SQL Fundamentals***

# SQL Fundamentals

- *If Accounting is the language of money and biology is the language of life then SQL is the language of data.*
- *Language to manipulate relational database like data sets.*
- *SQL has been widely used in the industry.*
- *Business Analyst, Data Steward, Database admin, BI Analyst and then the Data Engineer.*
- **Being primarily a set-oriented query language, it focuses on the WHAT you want to do instead of the HOW in procedural languages. SQL allows us to get the desired result through a sequence of well-defined operations.**

# SQL Fundamentals contd...

## Fundamental Difference between Set and Procedural

In the example below, both developers want to read enrollment information. One is being read from a predefined Student table. Notice the focus is on WHAT data is wanted -- the name in this case.

Contrast this with the procedural example on the bottom, wherein the desired data is the same. However, there is a heavy focus on HOW to get the data by opening the file and looping through each line.

## **Set Examples in SQL**

```
SELECT name FROM Student;
```

## **Procedural Examples in Python**

with open(Sudent.txt', 'r') as reader:

```
    line = reader.readline()
```

```
    while line != ":
```

```
        print(line, end="")
```

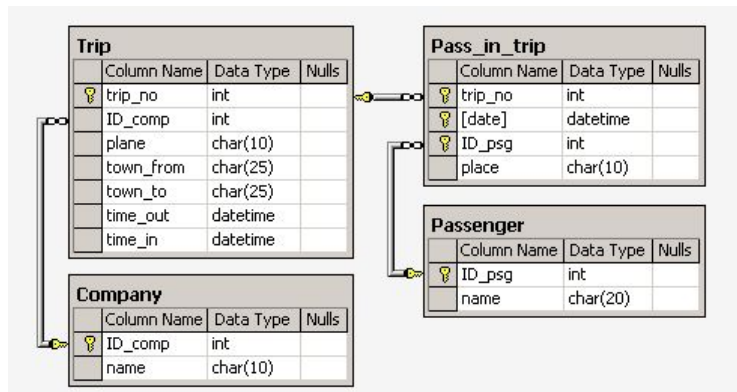
```
        line = reader.readline()
```

# SQL Fundamentals

A relational database is a collection of related tables, each assigned a unique name. The tables are designed to keep track of some aspects of the real world and their relationship.

Here is a sample schematic diagram of a database:

Column headers as attributes. Domain of corresponding attributes.



[Source](#)

# SQL

**Categorization of SQL Commands.** We can categorize different SQL commands based on the functions they perform as follows:

a) Data Definition Language (DDL):

Includes commands for defining schema, deleting relations, modifying the schema.

It consists of commands for specific integrity constraints

examples: CREATE, DROP, TRUNCATE, ALTER, COMMENT, RENAME

b) Data Query Language (DQL):

includes commands to perform operations on data within the defined schema.

example: Select

# SQL

## c) Data Manipulation Language (DML):

includes commands to manipulate data in the database

examples: Insert, Update, Delete

## d) Data Control Language (DCL):

includes the command to control rights and permission of the database.

example: Grant, Revoke

# ***SQL Commands***



# SQL Create

[Source](#)

```
SQL> CREATE TABLE CUSTOMERS(  
    ID      INT                NOT NULL,  
    NAME    VARCHAR (20)       NOT NULL,  
    AGE     INT                NOT NULL,  
    ADDRESS CHAR (25) ,  
    SALARY  DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

# *SQL Update and Insert*

[Source:](#)

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
```

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

# SQL select

```
SELECT column1,column2 FROM table_name  
column1 , column2: names of the fields of the table  
table_name: from where we want to apply query
```

Select \* from <table\_name>

Select distinct column\_name1, column\_name2 from table\_name

Select \* from table\_name order by column\_name.column\_number LIMIT 3

# *From and Where Clause*

Select \* from table\_name1, table\_name 2 - cartesian product of both tables

## **Where Clause**

It specifies the condition that the result must satisfy.

SQL allows the use of the logical connectives and, or, and not

Comparison operators as <, <=, >, >=, =, and <> can be applied too

For String you can use LIKE and len()

# SQL commands

SYNTAX ORDER	Command	Function
1	<b>SELECT</b>	Returns the specified column(s) and their respective row(s)
2	<b>DISTINCT</b>	Ensures the rows from the selected column are unique.
3	<b>FROM</b>	Identifies the first table
4	<b>JOIN</b>	Signifies two tables are being combined
5	<b>ON</b>	Identifies what column(s) the tables should join on column and second table join column
6	<b>WHERE</b>	Used to create logical conditions on one or many non-aggregated columns. Returns the rows of that respective column that meet the criteria. In other words, filters out the rows of the dataset that do not return true for the condition
7	<b>GROUP BY</b>	Used on non-aggregating columns to return a unique set of rows. If there is one aggregation and column not aggregated. The group by is used on the non-aggregating column.
8	<b>HAVING</b>	Used to create logical conditions on one or many aggregated columns. Returns the rows of that respective column that meet the criteria. In other words, filters out the rows of the dataset that do not return true for the condition .
9	<b>ORDER BY</b>	Sorts the final dataset by the given column(s) ascending by default. Unless specified to be descending.
10	<b>LIMIT</b>	Limits the number of rows returned by the given number

# ORDER of SQL execution

EXECUTE ORDER	Command	Function
1	FROM	Identifies the first table
2	ON	Identifies what column(s) the tables should join on column and second table join column
3	JOIN	Keyword that signifies two tables are being combined
4	WHERE	Used to create logical conditions on one or many non-aggregated columns. Returns the rows of that respective column that meet the criteria. In other words, filters out the rows of the dataset that do not return true for the condition
5	GROUP BY	Used on non-aggregating columns to return a unique set of rows. If there is one aggregation and column not aggregated. The group by is used on the non-aggregating column.
6	HAVING	Used to create logical conditions on one or many non-aggregated columns. Returns the rows of that respective column that meet the criteria. In other words, filters out the rows of the dataset that do not return true for the condition .
7	SELECT	Returns the specified column(s) and their respective row(s)
8	DISTINCT	Ensures the rows from the selected column are unique.
9	ORDER BY	Sorts the final dataset by the given column(s) ascending by default. Unless specified to be descending.
10	LIMIT	Limits the number of rows returned by the given number

# SQL Question

Find the 3 most profitable companies in the entire world.  
Output the result along with the corresponding company name.  
Sort the result based on profits in descending order.

Table: forbes\_global\_2010\_2014

[Link](#)

forbes\_global\_2010\_2014

company:	varchar
sector:	varchar
industry:	varchar
continent:	varchar
country:	varchar
marketvalue:	float
sales:	float
profits:	float
assets:	float
rank:	int
forbeswebpage:	varchar

# *UNION vs UNION ALL*

- A set operator is a keyword that combines or compares two queries or tables without join and returns one dataset.
- The result will be similar to join or different depending on the the keyword
- Number of columns need to match and the datatype need to be compatible
- `SELECT t1.col1 from t1 UNION select t2.col1 from t2`
- UNION ALL return all the columns and UNION doesn't return

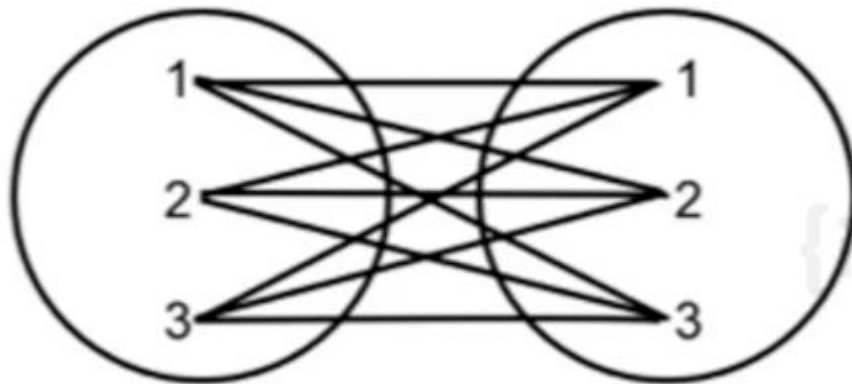


# CROSS JOIN/ CARTESIAN JOIN

A join that return every row from t1 with each row combination in t2

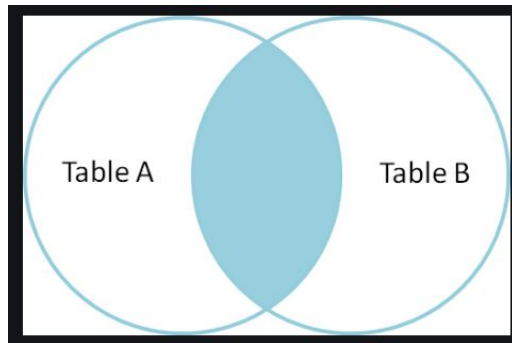
```
SELECT t1.col, t2.col2 from t1,t2
```

FULL OUTER JOIN

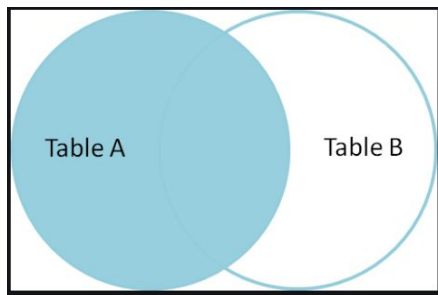


# SQL JOIN

INNER JOIN



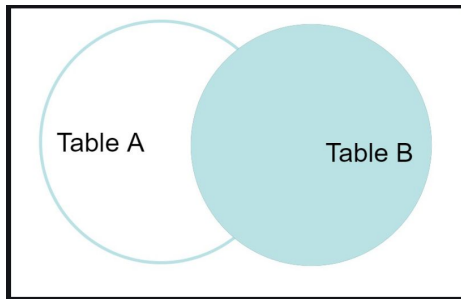
LEFT JOIN



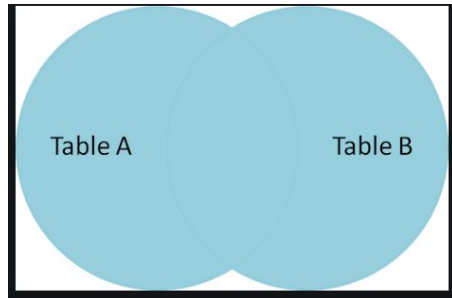
[SOURCE](#)

# SQL JOIN

RIGHT JOIN



FULL JOIN



[SOURCE](#)

# SQL Subqueries

- Subquery is another query embedded within a where clause, will return data that will be used by the main query.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- There are a few rules that subqueries must follow –
  - Subqueries must be enclosed within parentheses.
  - A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
  - An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
  - Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
  - The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
  - A subquery cannot be immediately enclosed in a set function.
  - The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

# Subquery contd..

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
      (SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
[WHERE])
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

# Subquery contd..

```
SQL> SELECT *  
      FROM CUSTOMERS  
      WHERE ID IN (SELECT ID  
                  FROM CUSTOMERS  
                  WHERE SALARY > 4500) ;
```

[Source](#)

# WITH Clause

- The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.
- The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause. Queries that have an associated WITH clause can also be written using nested sub-queries but doing so add more complexity to read/debug the SQL query.
- WITH clause is not supported by all database system.
- The name assigned to the sub-query is treated as though it was an inline view or table
- The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database.

```
WITH temporaryTable (averageValue) as
  (SELECT avg(Attr1)
   FROM Table)
SELECT Attr1
FROM Table, temporaryTable
WHERE Table.Attr1 > temporaryTable.averageValue;
```

[Source](#)



# SQL Question Time...

Calculate each user's average session time. A session is defined as the time difference between a page\_load and page\_exit. For simplicity, assume a user has only 1 session per day and if there are multiple of the same events on that day, consider only the latest page\_load and earliest page\_exit. Output the user\_id and their average session time.

Table: facebook\_web\_log

facebook\_web\_log

user_id:	int
timestamp:	datetime
action:	varchar

[Link](#)

# *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.

[Source](#)

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY depname) FROM empsalary;
```

depname	empno	salary	avg
develop	11	5200	5020.0000000000000000
develop	7	4200	5020.0000000000000000
develop	9	4500	5020.0000000000000000
develop	8	6000	5020.0000000000000000
develop	10	5200	5020.0000000000000000
personnel	5	3500	3700.0000000000000000
personnel	2	3900	3700.0000000000000000
sales	3	4800	4866.6666666666666667
sales	1	5000	4866.6666666666666667
sales	4	4800	4866.6666666666666667

(10 rows)

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY depname) FROM empsalary;
```

depname	empno	salary	avg
develop	11	5200	5020.0000000000000000
develop	7	4200	5020.0000000000000000
develop	9	4500	5020.0000000000000000
develop	8	6000	5020.0000000000000000
develop	10	5200	5020.0000000000000000
personnel	5	3500	3700.0000000000000000
personnel	2	3900	3700.0000000000000000
sales	3	4800	4866.6666666666666667
sales	1	5000	4866.6666666666666667
sales	4	4800	4866.6666666666666667

(10 rows)

# *SQL Window Functions*

There is another important concept associated with window functions: for each row, there is a set of rows within its partition called its window frame. Many (but not all) window functions act only on the rows of the window frame, rather than of the whole partition. By default, if ORDER BY is supplied then the frame consists of all rows from the start of the partition up through the current row, plus any following rows that are equal to the current row according to the ORDER BY clause. When ORDER BY is omitted the default frame consists of all rows in the partition.

# SQL Window Functions

- ROW\_NUMBER()
- RANK()
- DENSE\_RANK()- DENSE\_RANK() function to specify a unique rank number within the partition as per the specified column value. It is similar to the Rank function with a small difference. In the SQL RANK function DENSE\_RANK(), if we have duplicate values, SQL assigns different ranks to those rows as well. Ideally, we should get the same rank for duplicate or similar values
- NTILE(): function to distribute the number of rows in the specified (N) number of groups. Each row group gets its rank as per the specified condition

# Question Time...

Find the titles of workers that earn the highest salary. Output the highest-paid title or multiple titles that share the highest salary.

Tables: worker, title

## worker

worker_id:	int
first_name:	varchar
last_name:	varchar
salary:	int
joining_date:	datetime
department:	varchar

## title

worker_ref_id:	int
worker_title:	varchar

[Link](#)

# ***Data Modeling***



# *Some Definitions..*

## **Online Analytical Processing (OLAP):**

Databases optimized for these workloads allow for complex analytical and ad hoc queries, including aggregations. These type of databases are optimized for reads.

A data warehouse, a decision support database, has extensive history.

## **Online Transactional Processing (OLTP):**

Databases optimized for these workloads allow for less complex queries in large volumes. The types of queries for these databases are read, insert, update, and delete. Fast writes. Highly normalized tables; Limited history; Used for transactional systems.

## **Data Lakes**

A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to first structure the data, and run different types of analytics — from dashboards and visualizations to big data processing, real-time analytics, and machine learning to guide better decisions.

# Cobb's 12 Rules

Codd's twelve rules are a set of 13 rules proposed by Edgar F. Codd, a pioneer of the relational model for databases, designed to define what is required from a database management system in order for it to be considered relational, i.e., a relational database management system (RDBMS).

**Rule 0:** *The foundation rule:*

For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

**Rule 1:** *The information rule:*

All information in a relational data base is represented explicitly at the logical level and in exactly one way – by values in tables.

**Rule 2:** *The guaranteed access rule:*

Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

**Rule 3:** *Systematic treatment of null values:*

Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

**Rule 4:** *Dynamic online catalog based on the relational model:*

The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

**Rule 5:** *The comprehensive data sublanguage rule:*

A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

1. Data definition.
2. View definition.
3. Data manipulation (interactive and by program).
4. Integrity constraints.
5. Authorization.
6. Transaction boundaries (begin, commit and rollback).

**Rule 6:** *The view updating rule:*

All views that are theoretically updatable are also updatable by the system.

**Rule 7:** *Relational Operations Rule / Possible for high-level insert, update, and delete:*

The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.

**Rule 8:** *Physical data independence:*

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

**Rule 9:** *Logical data independence:*

Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

**Rule 10:** *Integrity independence:*

Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

**Rule 11:** *Distribution independence:*

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

**Rule 12:** *The nonsubversion rule:*

If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).

# ***What is a database schema?***

A database schema defines how data is organized within a relational database; this is inclusive of logical constraints, such as table names, fields, data types, and the relationships between these entities.

## **What is a database?**

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

# ER Diagram

The entity-relationship diagram (ERD) is a way to show the relationship between various entities in the data model. By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases.

**Entities-** Are represented by rectangles. Entities are the nouns; are a definable thing like people, objects or concepts that can have data stored about it.

**Relationship-** How entities are associated with one another is represented by relationship. There are four types of relationships: one-to-one, many-to-one, one-to-many, and many-to-many.

**Attributes-** Represented by Ovals. A property or characteristic of an entity.

# Keys

## Primary Key

A primary key (PK) is an attribute or group of attributes that uniquely identify one instance of an entity.

## Foreign Key (FK)

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

Rule 1 Information Rule is important for data modeling.

There are also composite keys.

# Normalization

The process of structuring a relational database in accordance with a series of normal forms, in order to reduce data redundancy (copies of data) and increase data integrity (answer I get back from database, is correct).

## Objects of Normal Forms

To free the database from unwanted insertions, updates and deletion dependencies

To reduce the need for refactoring the database as new types of data are introduced

To make the relational models more informative to users

To make the database neutral to the query statistics

# *Different Normal Forms (NF)*

## **1NF (First Normal Form):**

Each cell must have unique and single values (Atomic Values).

Be able to add data without altering the tables

Separate different relationships into separate tables. For example, sales details in all sales table and customer details in customer tables.

Keep relationship between tables with foreign keys

## **2NF (Second Normal Form)**

1NF has been reached.

All columns in the table must rely on the Primary Key (PK).



# *Different Normal Forms(NF) contd..*

## **3NF (Third Normal Form)**

Must be in 2NF

No transitive dependencies

*Transitive dependency — If  $A \rightarrow B$  and  $B \rightarrow C$  are examples of a functional dependency, then  $A \rightarrow C$  is called a transitive dependency.*

There are 6NF. Only 3NF are used for production.

# *Denormalization*

The process of trying to improve the read performance of a database at the expense of losing some write performance by adding redundant copies of data. Reads will be faster and writes will be slower. Data must be kept consistent.

We denormalize tables when there are many joins that need to happen to get the data. Joins are very flexible, but they slow down the reads. Thus, in order for the reads to be faster, the data can be denormalized. The data would first need to be normalized before denormalizing.

# *Fact and Dimension Tables*

A fact table is a structure that categorized facts and measures in order to enable users to answer business questions.

Dimension tables are people, products, and time. Dimension Tables answers the what, where, and when.

If you consider a Store Sales database, the dimension table would contain details such as:

- Where the product was bought?
- When the product was bought?
- What product was bought?

And Fact table would contain details such as:

- How many units of product was bought?

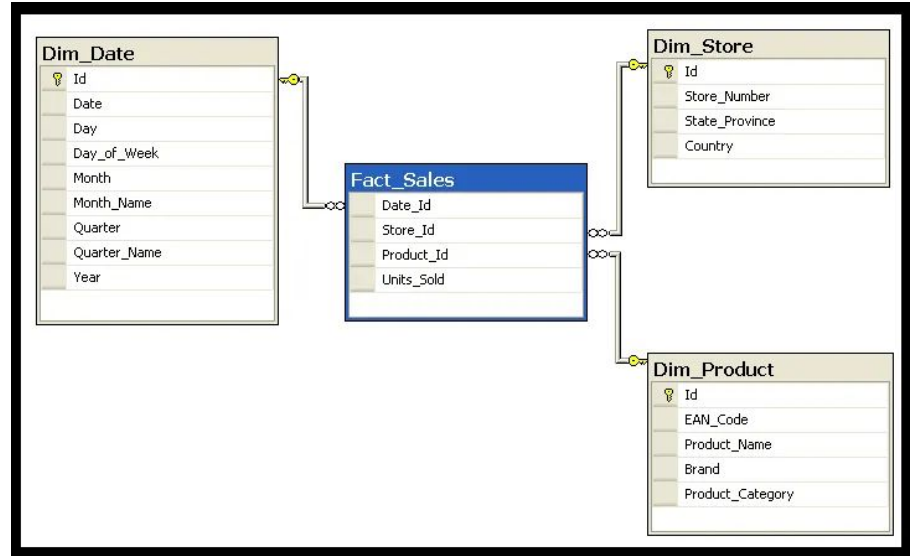
Fact and Dimensional Tables work together to create an organized data model.

# Star Schema and Snowflake Schema

## Star Schema

Simplest style of data mart schemas. The star schema consists of one or more fact tables referencing any number of dimensions tables.

Star schema gets the name from the physical model representing a star.



# *Benefits and drawbacks*

## **Benefits of the Star Schema**

The data can be demoralized

Simplifies queries

Fast aggregations like count, group by

## **Drawbacks of the Star Schema**

Issues that come with data demoralization

Data integrity

Decrease query flexibility

Many-to-many relationship simplified

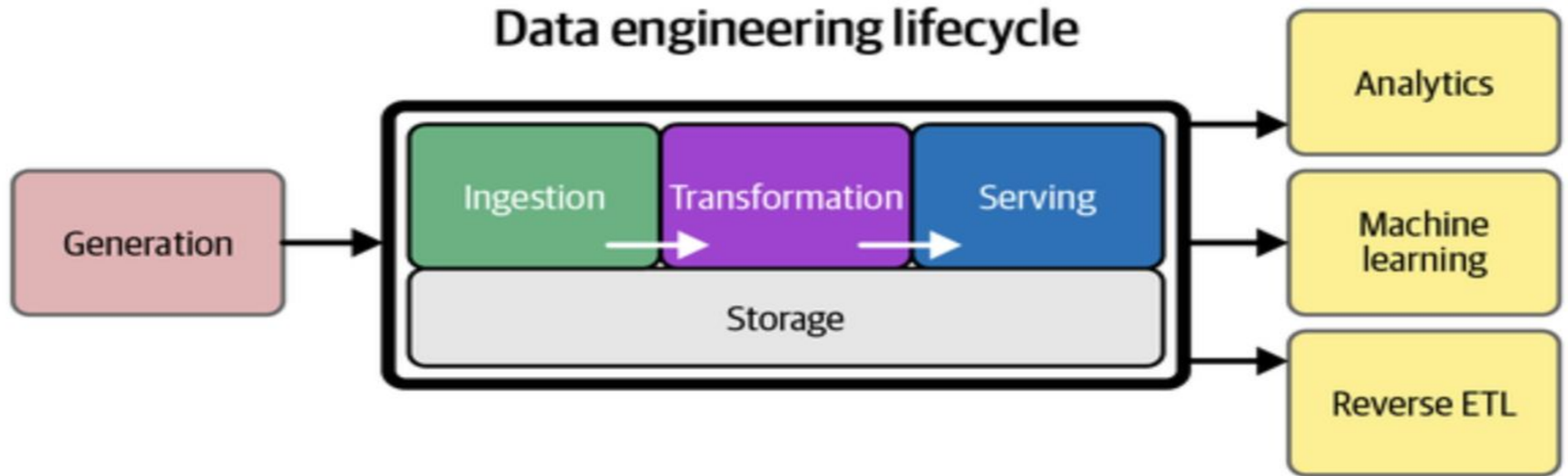
# *Snowflake Schema*

Logical arrangement of tables in a multidimensional database represented by centralized fact tables which are connected to multiple dimensional tables.

A complex snowflake shape emerges when the dimensions of a snowflake schema are elaborated having multiple levels of relationships child tables having multiple parents.

# ***Data Modeling***

# *Data Engineering LifeCycle*



---

## Undercurrents:





# Data Modeling

1. Data Modeling is an abstraction that organizes elements of data and how they relate to each other . — wiki
2. The process for creating data models for an information system. Data modeling is to organize data into a database system to ensure that your data is persisted and easily usable. Support business and user application.
3. Data modeling is about how to structure data to be used by different people within an organization. You can think of data modeling as the process of designing data and making it available to machine learning engineers, data scientists, business analytics, etc., so they can make use of it easily.

# *Why Data Modeling is important*

- Data organization is critical. Organized data can reduce complex queries for reads and can help with faster writes. Enabling efficient retrieval.
- Reduces redundancies
- Reduces storage requirements
- Organized data determines later data use
- Helps add flexibility when more data is to be added

# *Data Modeling Exercise*

You have been asked by a Daycare Management Team to help create a database to analyze, manage and grow their business.



# Conceptual Modeling

This stage is the beginning of the data modeling process. In this stage, things are at a conceptual level, and the identification of various entities is made. This stage defines **WHAT** the data model should look like. The idea is to identify the entities and the primary keys/foreign so that links between the entities can be established. It doesn't contain any attributes.

This stage defines what is the business we are interested in. It is the starting point carried out by the business stakeholders that helps organize and define the business rules. We should understand the business process here.

# Conceptual Modeling contd..

The conceptual data model has 3 main things:

- Entities are tangible and intangible properties that the business wants to capture. These are generally the tables of a database and can be identified in a model by a rectangle or square with a noun for its name.
- Attributes represent the characteristics of the entities. It can be understood as columns of a table. The attributes for the entity StudioWorker can be Identification Number and Name.
- A relationship is an association among several entities.

# *Conceptual Modeling*

The goal at this stage is to create a conceptual model that will capture the different entities and metrics that will make the business a success.

Here, we try to understand the business process, the different entities that would be needed to effectively design the data model, the relationship between the entities, and the primary key and foreign key constraints.

We need to first understand the business process.

# What is a business process?

A business process is a collection of business tasks and activities that when performed by people or systems in a structured course, produces an outcome that contributes to business goals. For example, if you are creating a ticketing system, and the questions you want answered are:

Questions	Business processes would be:
How many tickets were sold by event?	— Customer purchases tickets
How many customers purchased the tickets?	— Returning tickets
Different vendors and the metrics data for the event	— Canceling tickets
	Vendor lists tickets



# *Conceptual Modeling*

- What is the Teacher:Student ratio?
- What is the curriculum compared to other daycare?
- What are the different ages in the daycare?
- Are the students grouped by age?
- What is the Study and play schedule?
- How do the rates compare with the other daycare? - Taya

# *The business process..*

The business process will be:

- Teacher:Student ratio based on the age of the kids.
- Age-based schedule and curriculum for the kids
- Materials available to the kids.
- Rates based age

# *The tables we will need..*

Student
Id
Name
Age
Food allergy
Teacher id
Parent id
Assessment id
Syllabus id
Rate id

Teacher
Id
Name
Qualification
Student id
Syllabus id

Assessments
Id
Student id
Grades
performance

Parent
Id
Name
Student id
Address
Consent

Syllabus
Id
Subject
Content
Teacher id

Materials
Id
Name
Teacher id
Content

Grain of the data: the lowest level the data can be boiled down to.

# *Logical Data Model*

In this stage, we define HOW the data model implementation will be done from the design perspective. The logical stage is carried out by enhancing the conceptual stage by including all the columns and their data types. Other constraints, if they exist, are also identified in this stage.

Defines how the data model will be implemented irrespective of the DBMS used. Add attributes constraints until this point the data model is business focused. Logical data model is irrespective on RDMS.

Database design requires that we find a “good” collection of relation schemas.

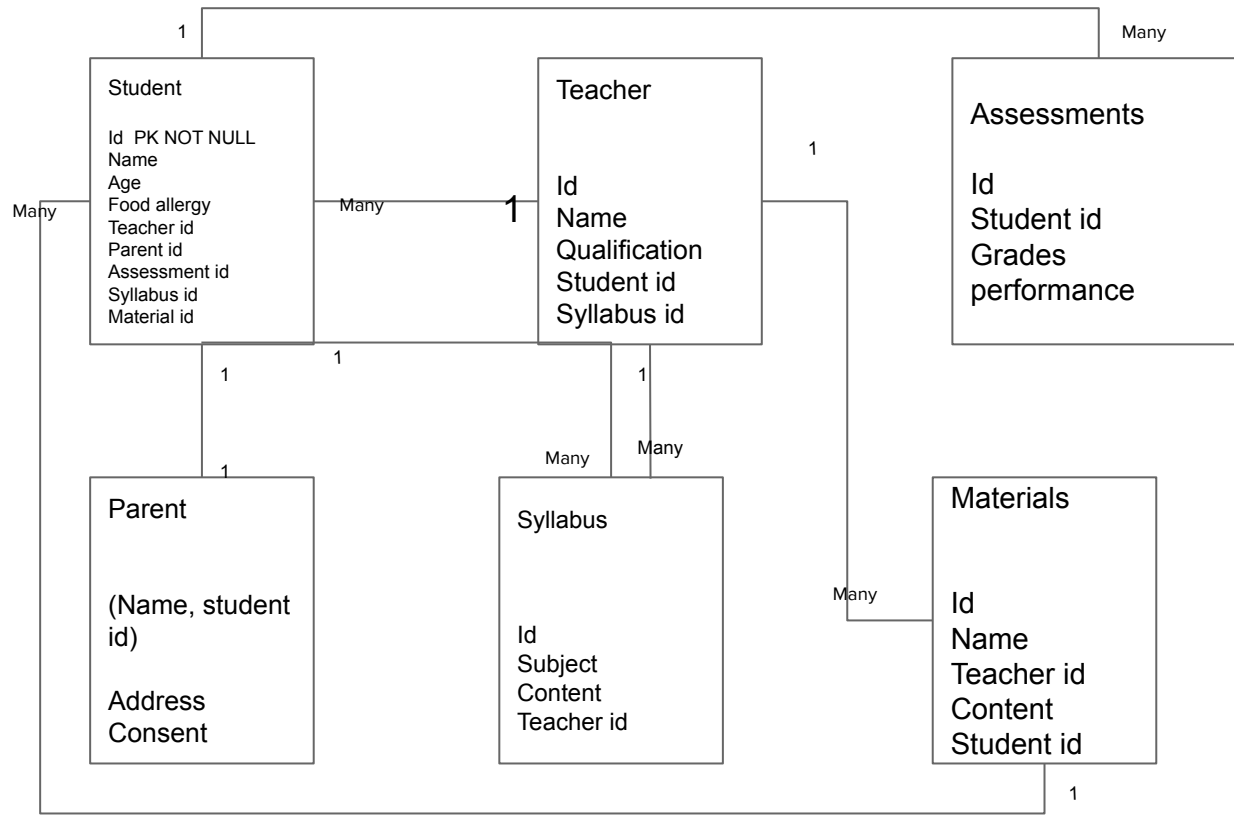
# *Logical Data Model*

Business decision — What attributes should we record in the database?

Computer Science decision — What relation schemas should we have, and how should the attributes be distributed among the various relation schemas?

Here we decided on the column names, the primary key, foreign key constraints, NULL value constraints.

Also, here we decide on the fact and dimension tables and the schema — star or snowflake.



# *Physical Data Model*

In this stage, we implement the data model on the specific DBMS that is used. All the constraints created in the logical stage are implemented as well. While doing this, one might see some errors, and there is a need to debug those. Some of the errors could be — the DBMS does not support the data type, etc.

# ***Python for Data Engineering***



# Introduction..

There are other languages that are used like Java and Scala.

Python is a dynamically typed language, it not as fast as Java and Scala. It's an interpreted object-oriented programming language and is considered a high-level language, in IDLE.

Easy readable code. Python is free open source and has a big community for support.

**Big Data Tools:** PyTorch, PySpark, Seaborn, TensorFlow, scikit-learn, matplotlib, NumPy, pandas.

Compared to Scala and Java.

# Introduction

## ***How is Python used for Data Engineering?***

- Python is used mostly for working with tools like Apache Spark, Apache Airflow etc

## ***What do data engineers use Python for?***

- Data Ingestion
- Data acquisition
- Data Manipulation
- Building ELT/ETL pipelines.

# Python Commands

- **List:** Most widely used, Grow and shrink size as needed, sequence type, sortable.
- **Tuple:** Immutable, Useful for fixed data, Sequence type, faster than lists, Ordered, Allows duplicated because they are indexed.
- **Set:** Store non duplicate items, very fast access, Unordered, Meth Set ops(Union, intersect)
- **Dict:** Key/value pairs, Associative array, Unordered

# Python Commands

**Dict:** Key/value pairs, Associative array, Unordered

Python

```
d = {  
    <key>: <value>,  
    <key>: <value>,  
    .  
    .  
    .  
    <key>: <value>  
}
```


Python

```
d = dict([  
    (<key>, <value>),  
    (<key>, <value>),  
    .  
    .  
    .  
    (<key>, <value>)  
])
```

**List:**

```
list1 = ['physics', 'chemistry', 1997, 2000];  
list2 = [1, 2, 3, 4, 5 ];  
list3 = ["a", "b", "c", "d"]
```

# For and While Loop

For Loop	While Loop
The for loop is used for definite loops when the number of iterations is known.	The while loop is used when the number of iterations is not known.
For loops can have their counter variables declared in the declaration itself.	There is no built-in loop control variable with a while loop.
This is preferable when we know exactly how many times the loop will be repeated.	The while loop will continue to run infinite number of times until the condition is met.
The loop iterates infinite number of times if the condition is not specified.	If the condition is not specified, it shows a compilation error. 

# Lambda Function

It is an Anonymous Function. A function without a name. It takes any number of arguments and then returns a single expression. They are typically used in order to take up less space for an argument. The syntax for a lambda function consists of an argument and an expression separated by a single ':' in the middle.

lambda argument(s) : expression

def f(x):                f(3) >> 6

return x \* 2

lambda x: x \* 3

There can be any number of arguments, but only one expression. For a lambda function, a return statement is not necessary, as it is able to return an expression on its own.

# Filter()

When you want to focus on specific values in an iterable, you can use the filter function. The following is the syntax of a filter function:

```
filter(function, iterable)
```

```
lambda x: x % 2 == 0
```

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
filter(lambda x: x % 2 == 0, list1)
```

```
>> <filter at 0x1e3f212ad60> # The result is always filter object so I will need to convert it to list using list()
```

```
list(filter(lambda x: x % 2 == 0, list1))
```

```
>> [2, 4, 6, 8, 10]
```

# Map()

When you want to modify every value in an iterable.

```
map(function, iterable)
```

```
list1 = [2, 3, 4, 5]
```

```
list(map(lambda x: pow(x, 2), list1))
```

```
>> [4, 9, 16, 25]
```

[Source](#)



# Apply()

```
df1 = df.apply(lambda x: x * x)
```

```
import pandas as pd

df = pd.DataFrame({'A': [1, 2], 'B': [10, 20]})

def square(x):
    return x * x

df1 = df.apply(square)

print(df)
print(df1)
```

Output:

```
   A  B
0  1 10
1  2 20

   A  B
0  1 100
1  4 400
```

[Source](#)

# Shallow Copy vs Deep Copy

With a **Shallow Copy**, only the reference address is copied.

A shallow copy means constructing a new collection object and then populating it with references to the child objects found in the original. In essence, a shallow copy is only one level deep. The copying process does not recurse and therefore won't create copies of the child objects themselves.

With a **Deep Copy**, both the original object and all repetitive copies are copied and stored.

A deep copy makes the copying process recursive. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. Copying an object this way walks the whole object tree to create a fully independent clone of the original object and all of its children.

# Shallow Copy vs Deep Copy

Shallow Copy:

Python

>>>

```
>>> xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
>>> ys = list(xs) # Make a shallow copy
```

Deep Copy:

Python

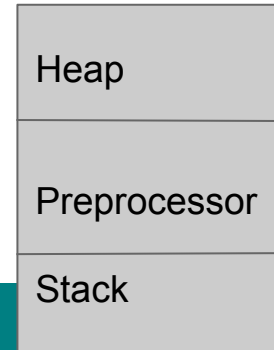
>>>

```
>>> import copy  
>>> xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
>>> zs = copy.deepcopy(xs)
```

# Memory Managed in Python

Python's memory manager automatically allocates memory. Python contains a garbage collector that helps automate the reuse of all unused memory within a program. Objects are deleted from the heap memory as soon as they have no reference point. This allows memory to be freed up and allows for more efficient memory management.

[Source](#)



# *Python Argument passing*

Parameters in Python are passed by reference. What this means is that if a parameter reference is changed, it reflects back to the calling function. As all Python data types are objects, a parameter passing is passed by assignment. Both the new variable and the original variables in Python will end up pointing to the same object.

# Python Decorators

In Python, a decorator is the implementation of a pattern that allows adding a behavior to a function or a class. It is usually expressed with the @decorator syntax prefixing a function. Here's a contrived example:

Python

```
def some_decorator(f):  
    def wraps(*args):  
        print(f"Calling function '{f.__name__}'")  
        return f(args)  
    return wraps  
  
@some_decorator  
def decorated_function(x):  
    print(f"With argument '{x}'")
```

Shell

```
Calling function 'decorated_function'  
With argument 'Python'
```

# ***\*\*args and \*\*kwargs***

```
def function_name(*args, *kwargs):
```

```
    # body
```

\*args allows you to pass the desired number of arguments to the function. Args generally means arguments in this case. regardless of the number of arguments passed, \*args is showing you the result.

what about \*\*kwargs? Well, they are not much different. The term Kwargs generally represents keyword arguments, suggesting that this format uses keyword-based Python dictionaries.

\*\*kwargs stands for keyword arguments. The only difference from args is that it uses keywords and returns the values in the form of a dictionary.

[Source](#)

# Python Packages

Python has many packages for data science, ML and data engineering.

- Caffe, PyTorch, Theano, TensorFlow, Keras, Pandas, Numpy, Matplotlib, Scipy
- Most extensively used are Pandas and Numpy.
- **Pandas**: The Pandas library is the go-to Python library for data analysis. It can be used for many different operations within the data science and data analytics specialty and is a popular option for large data sets. The Pandas library is one of the most popular options for data manipulation within Python. In order to import Pandas, the import statement “import pandas as pd” is used.
- **Numpy**: NumPy is a package in Python that is used for scientific computing. As it was once part of SciPy, it shares many features. However, the two are optimized for different uses and were separated into two different packages in order to accommodate these differences. Through NumPy, you can get access to many scientific computing features that include linear algebra, basic statistics, and other mathematical functions. Universally named “np” when imported into a project, this tool is an easily accessible package. In order to import NumPy, use the import statement “import NumPy as np” at the beginning of a Python program.

[Source](#)



# Pandas

Pandas DataFrame: DataFrame is the most important and widely used data structure and is a standard way to store data. DataFrame has data aligned in rows and columns like the SQL table or a spreadsheet database.

We can either hard code data into a DataFrame or import a CSV file, tsv file, Excel file, SQL table, etc. We can use the below constructor for creating a DataFrame object.

*`pandas.DataFrame(data, index, columns, dtype, copy)`*

data - create a DataFrame object from the input data. It can be list, dict, series, Numpy ndarrays or even, any other DataFrame.

index - has the row labels

columns - used to create column labels

dtype - used to specify the data type of each column, optional parameter

copy - used for copying data, if any

# Pandas Commands

Import pandas as pd

```
#import from csv
```

```
data = pd.read_csv('cities.csv')
```

```
print(data)
```

```
print(df.head(2))
```

```
print(df.tail(1))
```

```
Df1 = df[df['Literacy %']>90]
```

```
print(df.dtypes)
```

```
print(df.index)
```

```
print(df[df['State'].isin(['Karnataka', 'Tamil Nadu'])])
```

```
df.columns
```

```
df.values
```

```
df.describe()
```

```
df.sort_values()
```

```
df[1:3,step] #slicing df[start, stop, step]
```

```
print(df[df['Literacy %']>90])
```

```
df.rename(columns = {}, inplace=True)
```

```
merge(), groupby(), sort_values() and concat()
```

Question 1: [Link](#)

Question 2: [Link](#)

Question 3 : [Link](#)





**Q & A**

WOMEN WHO  
**CODE**®

***Upcoming for Next Week***

WOMEN WHO  
**CODE**®