

Introduction to Databases

Zeham Management Technologies BootCamp
by SDAIA
August 4th, 2024



SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority

Introduction to Databases

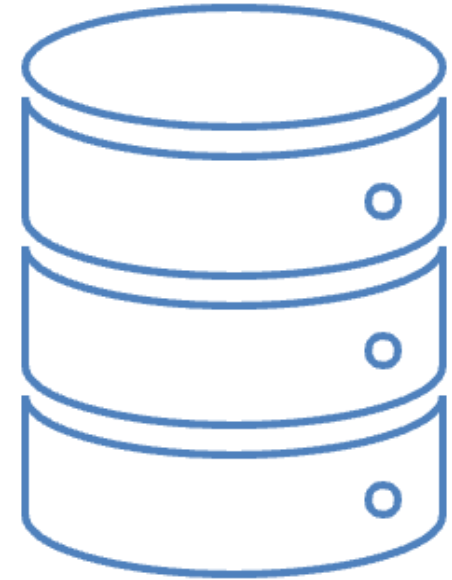


What is Database?

Database is a structured collection of data that can be easily accessed, managed, and updated.

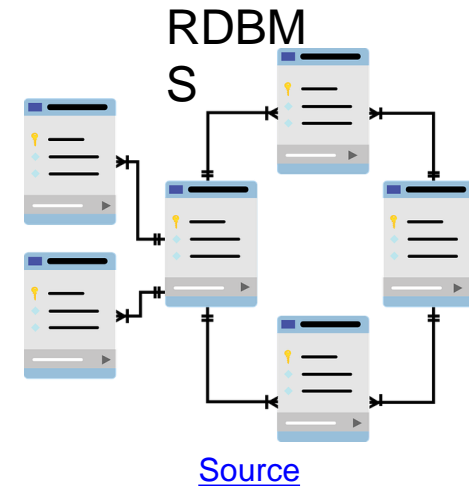
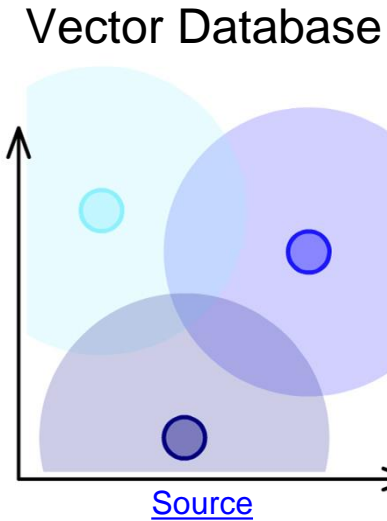
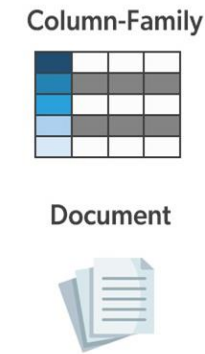
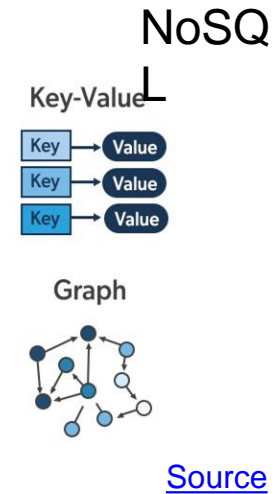
Types of Databases:

- **Relational Databases (RDBMS):** Organized into tables with rows and columns.
- **Non-relational Databases (NoSQL):** Organized differently, such as key-value pairs, documents, or graphs.
- **Vector Databases:** Specialized for storing and querying high-dimensional vectors.



Types of Databases

- **Relational Databases (RDBMS):**
 - **Examples:** MySQL, PostgreSQL, Oracle
 - **Features:** Structured schema, SQL query language, ACID properties.
- **Non-relational Databases (NoSQL):**
 - **Examples:** MongoDB, Cassandra, Redis
 - **Features:** Flexible schema, various data models (document, key-value, graph), eventual consistency.
- **Vector Databases:**
 - **Examples:** Pinecone, FAISS, Milvus
 - **Features:** Optimized for similarity search and nearest neighbor queries on high-dimensional vector data.





Database Components

- **Relational Databases:**
 - **Tables, Rows, Columns:** Basic structures for data storage.
 - **Indexes, Keys, Constraints:** Improve performance and maintain data integrity.
- **Non-relational Databases:**
 - **Collections, Documents:** Organize data in flexible formats.
 - **Indexes, Sharding:** Optimize performance and scalability.
- **Vector Databases:**
 - **Vectors:** Represent data in high-dimensional space.
 - **Indexes:** Specialized indexing techniques for efficient vector search (e.g., HNSW, IVF).
- **Query Languages:**
 - **SQL:** Standard language for relational databases.
 - **NoSQL Queries:** Vary by database type, e.g., MongoDB query language.
 - **Vector Queries:** Similarity search and nearest neighbor queries.





Importance of Databases in Data Science

Data Warehousing and Business Intelligence

- Store and analyze historical data for business insights.

Real-time Data Processing and Analytics

- Process and analyze data as it is generated.

Machine Learning Model Training and Deployment

- Store and access large datasets for model training.

Big Data Analytics

- Handle and analyze massive datasets efficiently.

High-Dimensional Data Search

- Vector databases for applications like recommendation systems, image recognition, and natural language processing.





Data Management Challenges

Handling Large Volumes of Data

- Efficient storage and retrieval of large datasets.

Ensuring Data Quality and Consistency

- Maintain accuracy and integrity of data.

Maintaining Data Security and Privacy

- Protect sensitive data from unauthorized access.

Managing Database Performance and Scalability

- Optimize performance and scale with growing data.

Vector Data Challenges

- Efficient indexing and querying of high-dimensional data.





Database Components

- **Relational Databases:**
 - **Tables, Rows, Columns:** Basic structures for data storage.
 - **Indexes, Keys, Constraints:** Improve performance and maintain data integrity.
- **Non-relational Databases:**
 - **Collections, Documents:** Organize data in flexible formats.
 - **Indexes, Sharding:** Optimize performance and scalability.
- **Vector Databases:**
 - **Vectors:** Represent data in high-dimensional space.
 - **Indexes:** Specialized indexing techniques for efficient vector search (e.g., HNSW, IVF).
- **Query Languages:**
 - **SQL:** Standard language for relational databases.
 - **NoSQL Queries:** Vary by database type, e.g., MongoDB query language.
 - **Vector Queries:** Similarity search and nearest neighbor queries.

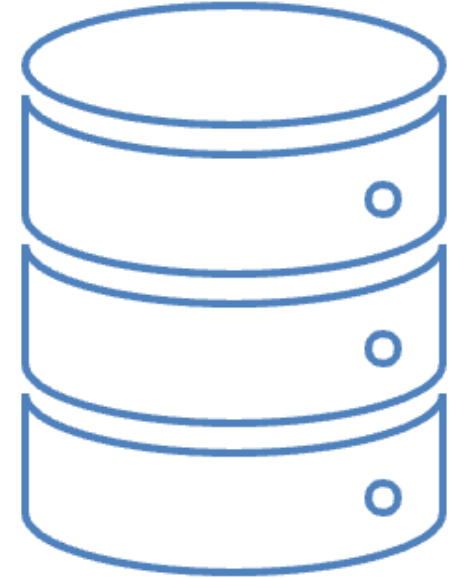


SQL Databases

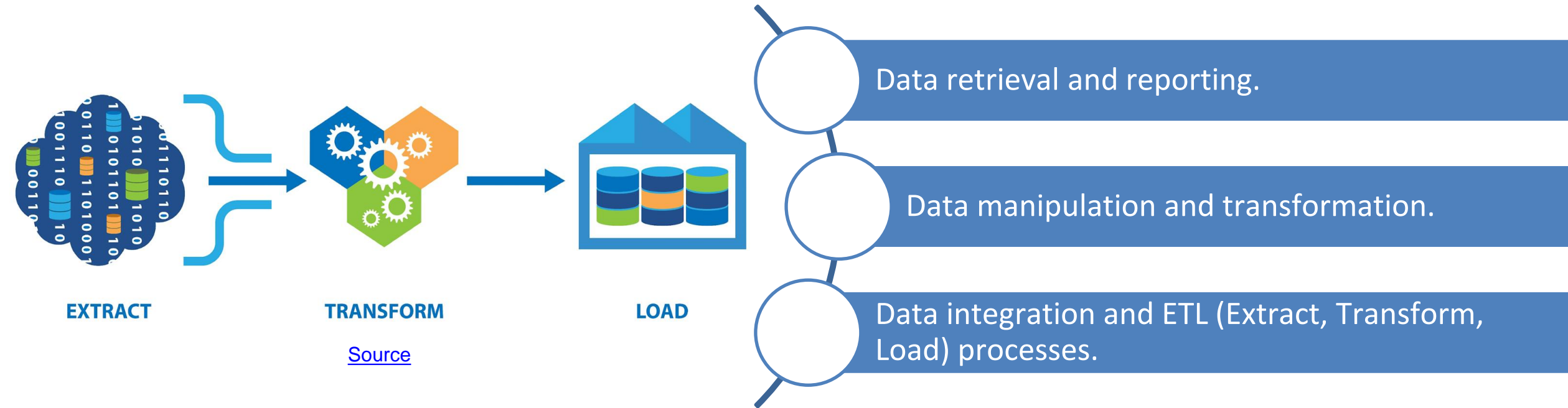


Database Components

Structured Query Language (SQL) is a standardized language used to manage and manipulate relational databases.



Database Components





SQL Concepts

Tables, Rows, and Columns:

- **Tables:** Collections of related data organized in rows and columns.
- **Rows:** Individual records in a table.
- **Columns:** Attributes or fields of the data.

Table

Rows

EmployeeID	FirstName	LastName	BirthDate	HireDate	Position	Salary
1	John	Doe	1985-06-15	2010-01-20	Software Engineer	75000
2	Jane	Smith	1990-07-25	2015-03-15	Data Analyst	68000
3	Michael	Brown	1979-12-30	2005-11-05	Project Manager	90000
4	Emily	Davis	1988-02-18	2012-09-10	HR Manager	85000
5	David	Wilson	1992-04-22	2018-06-01	Marketing Specialist	65000

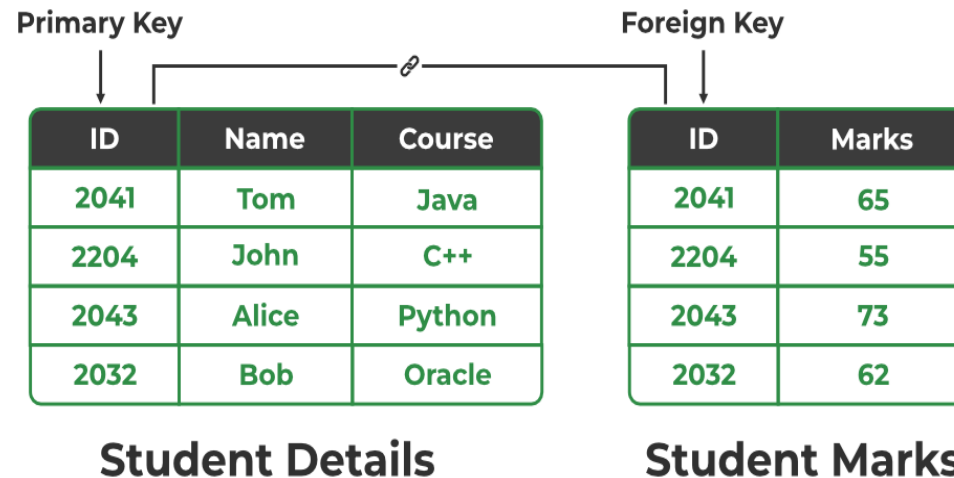
Columns



SQL Concepts Cont.

Primary Keys and Foreign Keys:

- **Primary Key:** Unique identifier for each row in a table.
- **Foreign Key:** A field in one table that uniquely identifies a row of another table.



[Source](#)





SQL Concepts Cont.

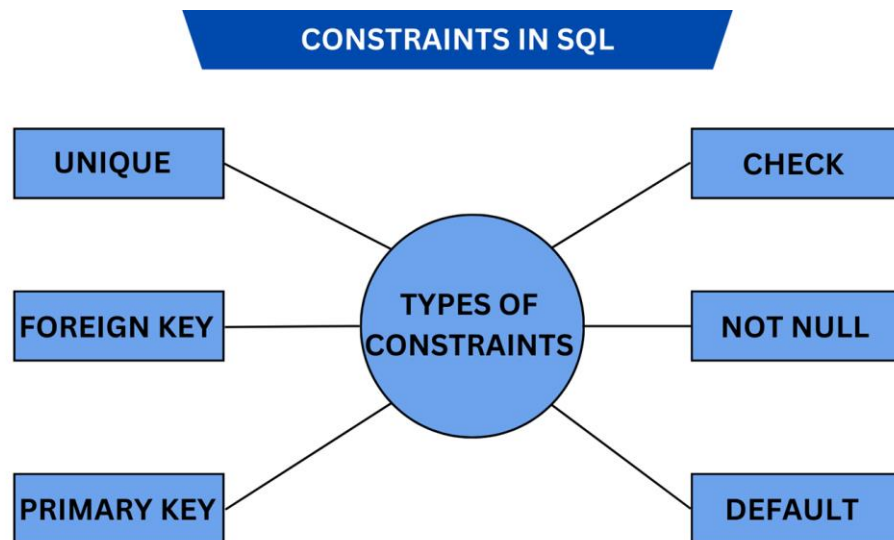
Primary key	Foreign key
It must contain unique values.	It can contain duplicate values.
It cannot contain null values.	It can contain null values.
A database can have only one primary key.	A database can have more than one foreign key.
It is used to identify the records in a table uniquely.	It is used to make a relation between two tables.



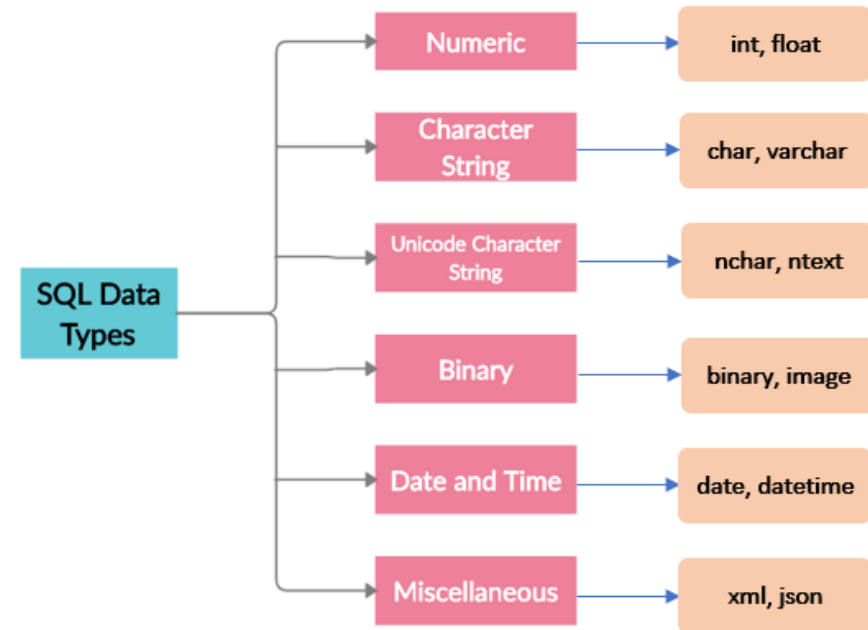
SQL Concepts Cont.

Data Types and Constraints:

- **Common data types:** INTEGER, VARCHAR, DATE, etc.
- **Constraints:** Rules applied to data (e.g., NOT NULL, UNIQUE).



[Source](#)



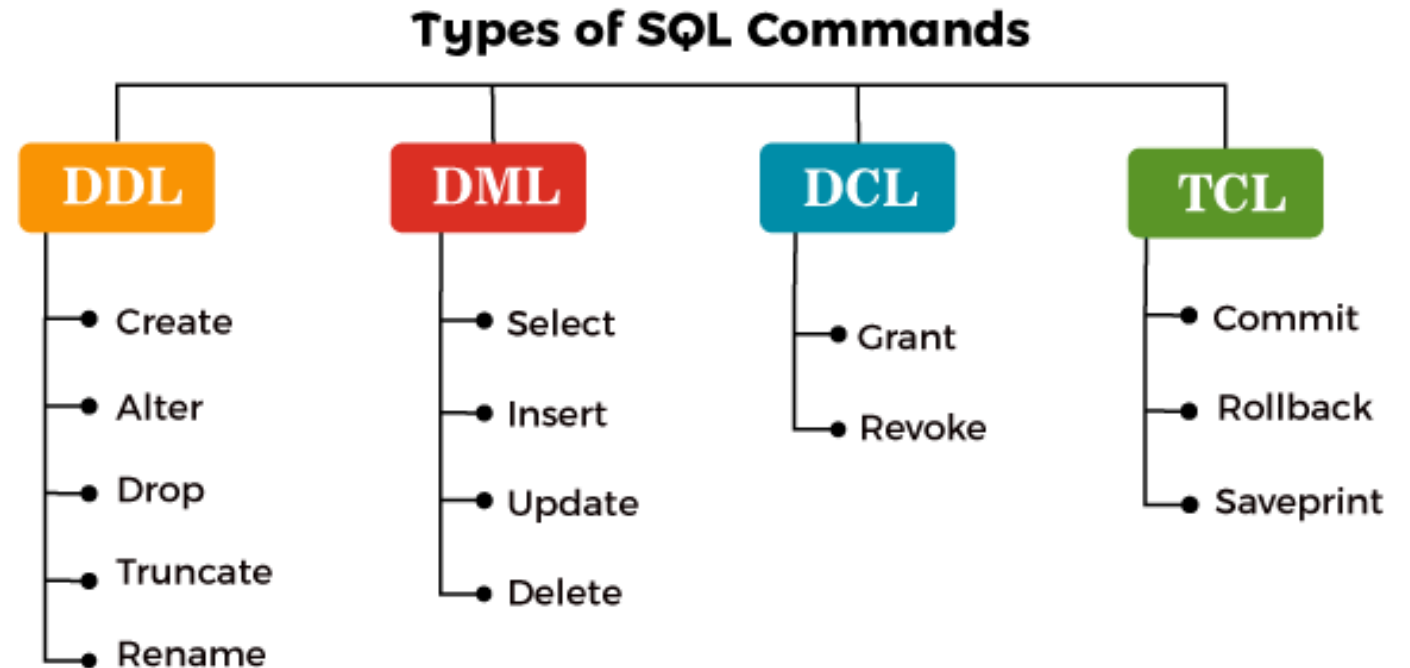
[Source](#)



SQL Concepts Cont.

Basic SQL Operations:

- **SELECT:** Retrieve data from a table.
- **INSERT:** Add new data to a table.
- **UPDATE:** Modify existing data.
- **DELETE:** Remove data from a table.



[Source](#)





SQL Queries

Basic Query Structure:

SELECT [columns] FROM [table] WHERE [conditions]



```
1 SELECT EmployeeID FROM Employees WHERE FirstName = 'John';
2
3 SELECT FirstName, LastName FROM Employees WHERE Salary > 70000;
4
5 SELECT * FROM Employees WHERE HireDate = '2010-01-20';
```

Employees

EmployeeID	FirstName	LastName	BirthDate	HireDate	Position	Salary
1	John	Doe	1985-06-15	2010-01-20	Software Engineer	75000
2	Jane	Smith	1990-07-25	2015-03-15	Data Analyst	68000
3	Michael	Brown	1979-12-30	2005-11-05	Project Manager	90000
4	Emily	Davis	1988-02-18	2012-09-10	HR Manager	85000
5	David	Wilson	1992-04-22	2018-06-01	Marketing Specialist	65000





SQL Queries (Example 1)



```
1 SELECT EmployeeID FROM Employees WHERE FirstName = 'John';  
2  
3 SELECT FirstName, LastName FROM Employees WHERE Salary > 70000;  
4  
5 SELECT * FROM Employees WHERE HireDate = '2010-01-20';
```

Employees

EmployeeID	FirstName	LastName	BirthDate	HireDate	Position	Salary
1	John	Doe	1985-06-15	2010-01-20	Software Engineer	75000
2	Jane	Smith	1990-07-25	2015-03-15	Data Analyst	68000
3	Michael	Brown	1979-12-30	2005-11-05	Project Manager	90000
4	Emily	Davis	1988-02-18	2012-09-10	HR Manager	85000
5	David	Wilson	1992-04-22	2018-06-01	Marketing Specialist	65000

EmployeeID

1





SQL Queries (Example 2)



```
1 SELECT EmployeeID FROM Employees WHERE FirstName = 'John';
2
3 SELECT FirstName, LastName FROM Employees WHERE Salary > 70000;
4
5 SELECT * FROM Employees WHERE HireDate = '2010-01-20';
```

Employees

EmployeeID	FirstName	LastName	BirthDate	HireDate	Position	Salary
1	John	Doe	1985-06-15	2010-01-20	Software Engineer	75000
2	Jane	Smith	1990-07-25	2015-03-15	Data Analyst	68000
3	Michael	Brown	1979-12-30	2005-11-05	Project Manager	90000
4	Emily	Davis	1988-02-18	2012-09-10	HR Manager	85000
5	David	Wilson	1992-04-22	2018-06-01	Marketing Specialist	65000

FirstName	LastName
John	Doe
Michael	Brown
Emily	Davis





SQL Queries (Example 3)

```
1 SELECT EmployeeID FROM Employees WHERE FirstName = 'John';
2
3 SELECT FirstName, LastName FROM Employees WHERE Salary > 70000;
4
5 SELECT * FROM Employees WHERE HireDate = '2010-01-20';
```

Employees

EmployeeID	FirstName	LastName	BirthDate	HireDate	Position	Salary
1	John	Doe	1985-06-15	2010-01-20	Software Engineer	75000
2	Jane	Smith	1990-07-25	2015-03-15	Data Analyst	68000
3	Michael	Brown	1979-12-30	2005-11-05	Project Manager	90000
4	Emily	Davis	1988-02-18	2012-09-10	HR Manager	85000
5	David	Wilson	1992-04-22	2018-06-01	Marketing Specialist	65000

EmployeeID	FirstName	LastName	BirthDate	HireDate	Position	Salary
1	John	Doe	1985-06-15	2010-01-20	Software Engineer	75000



SQL Joins

Joins:

- **INNER JOIN:** Select records with matching values in both tables.
- **LEFT JOIN:** Select all records from the left table and matched records from the right table.
- **RIGHT JOIN:** Select all records from the right table and matched records from the left table.
- **FULL JOIN:** Select all records when there is a match in either left or right table.

Example Database:

Department

DeptID	DeptName
1	HR
2	IT
3	Finance

Employee

EmpID	EmpName	DeptID	Salary
101	John Doe	1	50000
102	Jane Smith	2	60000
103	Michael Johnson	1	55000
104	Emily Davis	3	65000

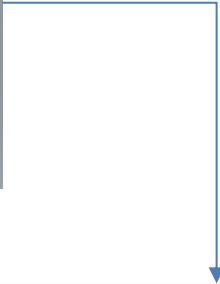




SQL Joins (INNER JOIN)



```
1 SELECT Employee.EmpName, Department.DeptName
2 FROM Employee
3 INNER JOIN Department ON Employee.DeptID = Department.DeptID;
```



EmpName	DeptName
John Doe	HR
Jane Smith	IT
Michael Johnson	HR
Emily Davis	Finance

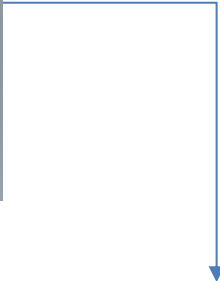




SQL Joins (LEFT JOIN)



```
1 SELECT Employee.EmpName, Department.DeptName
2 FROM Employee
3 LEFT JOIN Department ON Employee.DeptID = Department.DeptID;
```



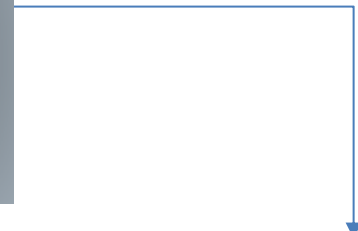
EmpName	DeptName
John Doe	HR
Jane Smith	IT
Michael Johnson	HR
Emily Davis	Finance



SQL Joins (RIGHT JOIN)



```
1 SELECT Employee.EmpName, Department.DeptName
2 FROM Employee
3 RIGHT JOIN Department ON Employee.DeptID = Department.DeptID;
```



EMPNAME	DEPTNAME
John Doe	HR
Jane Smith	IT
Michael Johnson	HR
Emily Davis	Finance

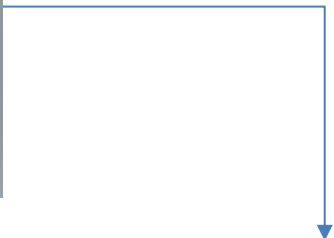




SQL Joins (FULL OUTER JOIN)



```
1 SELECT Employee.EmpName, Department.DeptName
2 FROM Employee
3 FULL OUTER JOIN Department ON Employee.DeptID = Department.DeptID;
```



EMPNAME	DEPTNAME
John Doe	HR
Jane Smith	IT
Michael Johnson	HR
Emily Davis	Finance





SQL Aggregation

Aggregation Functions:

COUNT: Returns the number of rows.

SUM: Returns the total sum of a numeric column.

AVG: Returns the average value of a numeric column.

MIN: Returns the smallest value.

MAX: Returns the largest value.

Example Database:

Customers

CustomerID	CustomerName	City
1	John Doe	New York
2	Jane Smith	Los Angeles
3	Michael Johnson	Chicago

Orders

OrderID	CustomerID	OrderDate	Amount
101	1	2023-01-15	150
102	1	2023-02-20	200
103	2	2023-02-25	350
104	3	2023-03-10	120
105	1	2023-03-15	180





SQL Aggregation (COUNT)



```
1 SELECT COUNT(*) AS TotalOrders
2 FROM Orders;
```

TotalOrders
5



SQL Aggregation (SUM)



```
1 SELECT SUM(Amount) AS TotalSales
2 FROM Orders;
```

TotalSales
1000



SQL Aggregation (AVG)



```
1 SELECT AVG(Amount) AS AvgOrderAmount
2 FROM Orders;
```

AvgOrderAmount
200



SQL Aggregation (MAX)



```
1 SELECT MAX(Amount) AS MaxOrderAmount
2 FROM Orders;
```

MaxOrderAmount

350





SQL Aggregation (MIN)



```
1 SELECT MIN(Amount) AS MinOrderAmount
2 FROM Orders;
```

MinOrderAmount
120





SQL Aggregation (MIN)



```
1 SELECT MIN(Amount) AS MinOrderAmount
2 FROM Orders;
```

MinOrderAmount
120





SQL Grouping Data

Grouping Data:

- **GROUP BY:** Group rows that have the same values in specified columns.
- **HAVING:** Filter groups based on a specified condition.

Example Database:

Customers

CustomerID	CustomerName	City
1	John Doe	New York
2	Jane Smith	Los Angeles
3	Michael Johnson	Chicago

Orders

OrderID	CustomerID	OrderDate	Amount
101	1	2023-01-15	150
102	1	2023-02-20	200
103	2	2023-02-25	350
104	3	2023-03-10	120
105	1	2023-03-15	180

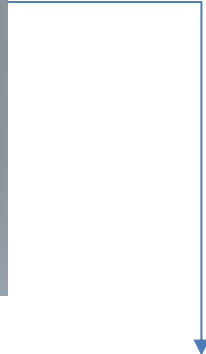




SQL Grouping Data (GROUP BY)



```
1 SELECT Customers.CustomerID, Customers.CustomerName,  
2     COUNT(Orders.OrderID) AS NumOrders,  
3     SUM(Orders.Amount) AS TotalSales,  
4     AVG(Orders.Amount) AS AvgOrderAmount  
5 FROM Customers  
6 LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
7 GROUP BY Customers.CustomerID, Customers.CustomerName;
```



CustomerID	CustomerName	NumOrders	TotalSales	AvgOrderAmount
1	John Doe	3	530	176.66666666666666
2	Jane Smith	1	350	350
3	Michael Johnson	1	120	120





SQL Grouping Data (HAVING)



```
1 SELECT Customers.CustomerID, Customers.CustomerName,  
2     SUM(Orders.Amount) AS TotalSales  
3 FROM Customers  
4 LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
5 GROUP BY Customers.CustomerID, Customers.CustomerName  
6 HAVING SUM(Orders.Amount) > 500;
```

CustomerID	CustomerName	TotalSales
1	John Doe	530



SQL Sorting

Sorting Data:

- **ORDER BY:** Sort the result set by one or more columns.

```
1 SELECT OrderID, CustomerID, OrderDate, Amount
2 FROM Orders
3 ORDER BY OrderDate DESC;
```

OrderID	CustomerID	OrderDate	Amount
105	1	2023-03-15	180
104	3	2023-03-10	120
103	2	2023-02-25	350
102	1	2023-02-20	200
101	1	2023-01-15	150

Example Database:

Customers

CustomerID	CustomerName	City
1	John Doe	New York
2	Jane Smith	Los Angeles
3	Michael Johnson	Chicago

Orders

OrderID	CustomerID	OrderDate	Amount
101	1	2023-01-15	150
102	1	2023-02-20	200
103	2	2023-02-25	350
104	3	2023-03-10	120
105	1	2023-03-15	180





Tutorials and Exercises

Exercises:

3- Advanced Machine Learning/1-Introduction to Database
/LAB/SQL_ Exercises.pdf

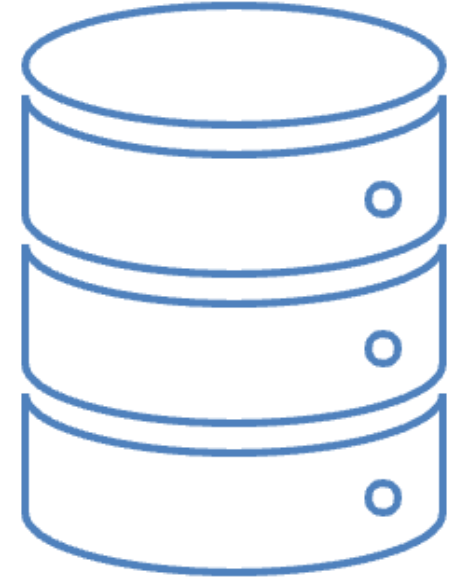


NoSQL Database



Database Components

NoSQL databases provide flexible schemas and horizontal scalability. They are designed to handle large volumes of unstructured or semi-structured data. Four main types of NoSQL databases: **Document**, **Key-Value**, **Column-Family**, and **Graph**.

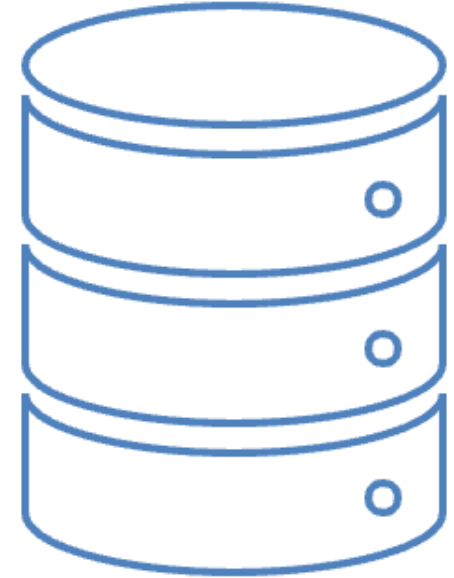




NoSQL databases rule

NoSQL databases has important rule in data science such as:

- Handle large-scale data processing and real-time analytics.
- Efficiently store and query hierarchical data structures.
- Support for complex queries and aggregation.





JSON and MongoDB

JSON (JavaScript Object Notation):

- Lightweight data interchange format.
- Easy to read and write for humans and machines.
- Commonly used for transmitting data in web applications (e.g., APIs).

MongoDB:

- Document-oriented NoSQL database.
- Stores data in BSON format (Binary JSON).
- Dynamic schemas and rich query capabilities.

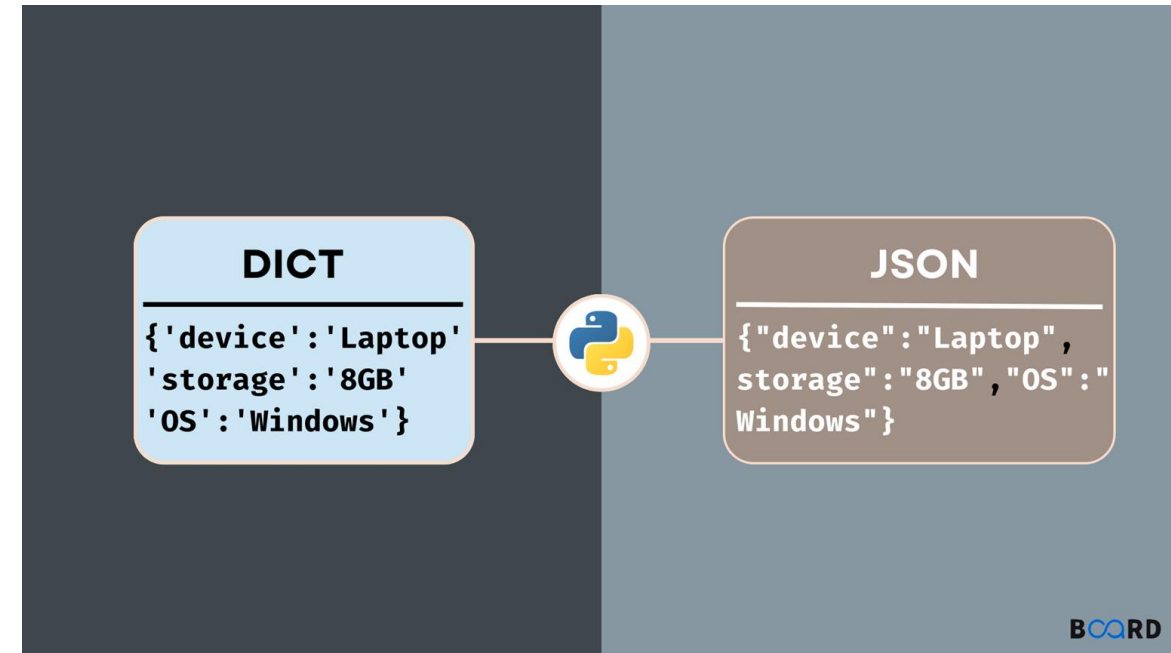




JSON vs Python Dictionary

JSON is a format for storing and sharing data. JSON is text formatted using JavaScript object notation. JSON and Python Dictionary are similar but there are small differences like:

- **Syntax:**
 - **JSON:** Text-based, double quotes
 - **Dictionary:** Native Python, flexible quotes
- **Data Types:**
 - **JSON:** Limited (strings, numbers, booleans, etc.)
 - **Dictionary:** Supports complex types
- **Usage:**
 - **JSON:** Data interchange
 - **Dictionary:** In-program data manipulation



[Source](#)

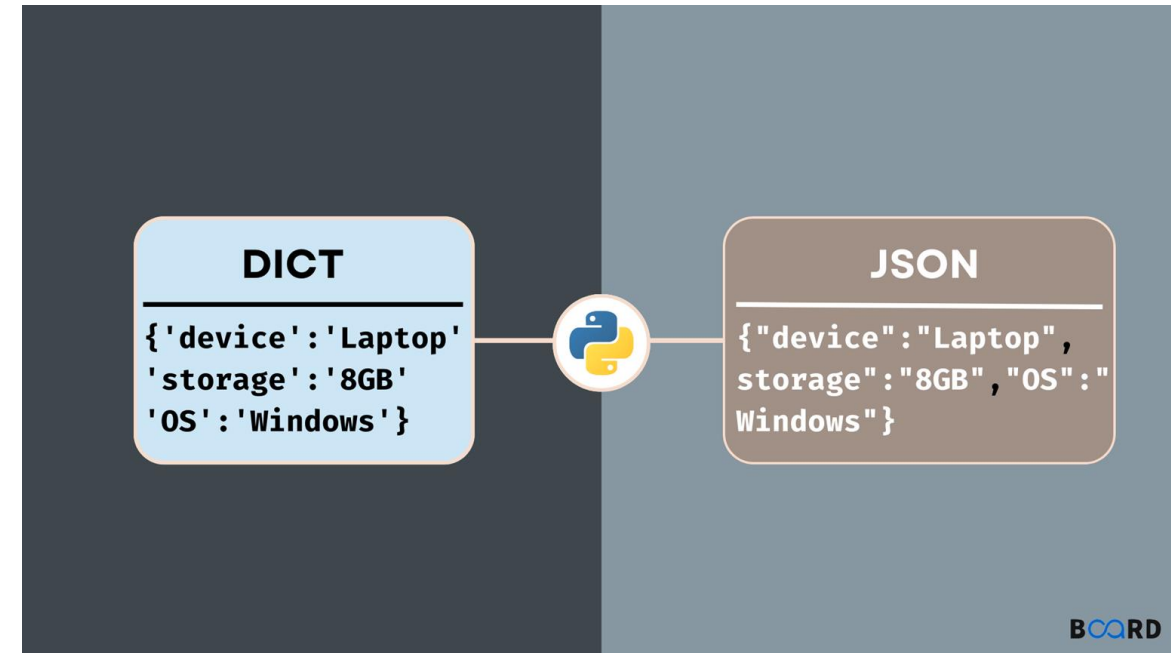




JSON and Python Objects

You can convert Python objects of the following types, into JSON strings:

- dict
- list
- tuple
- string
- int
- float
- True
- False
- None



[Source](#)





Python equivalents in JSON

Python	JSON Equivalent
True	true
False	false
float	Number
int	Number
None	null
dict	Object
list	Array
tuple	Array

[Source](#)





Tutorials and Exercises

Tutorial (Notebook):

3- Advanced Machine Learning/1-Introduction to Database
/LAB/JSON.ipynb

Exercises:

3- Advanced Machine Learning/1-Introduction to Database
/LAB/JSON_exercise.ipynb





Starting with MongoDB

To start with MongoDB, do the following steps:

1. Create an account at the official website of MongoDB.
2. Build a free cluster.
3. Chose the cloud provider and the region.
4. Select a cluster tier (the free tier, M0, is sufficient for most small projects).
5. Create a database user.
6. Set up network access.
7. Connect to your cluster and get your connection string 'add your database username and password to the connection string as specified in the documentation'.
8. Add it to your environment variable '.env'
9. Install PyMongo using `pip install pymongo`.



mongoDB®





Starting with MongoDB Cont.

Connect to MongoDB using the connection string, initiate the client to connect to MongoDB. Then, create a database with the name you desire and create a collection 'which is a table in SQL'.

```
1  # Connect to MongoDB
2  CONNECTION_STRING = os.environ.get('CONNECTION_STRING')
3  client = MongoClient(CONNECTION_STRING)
4
5  # Create a database
6  db = client['database_name']
7
8  # Create a collection
9  employees_collection = db['collection_name']
```



Starting with MongoDB (Insert Documents)

You can insert single single document or multiple documents using python dictionary or JSON as the document, use the `insert_one` method or a list of dictionaries using `insert_many` method.

```
1  # Insert single documentn
2  employees_collection.insert_one({'name': 'John', 'age': 25, 'country': 'USA'})
3  employees_collection.insert_one({'name': 'John', 'age': 28, 'country': 'UK'})
4  employees_collection.insert_one({'name': 'Jane', 'age': 22, 'country': 'UK'})
5  employees_collection.insert_one({'name': 'Ahmed', 'age': 30, 'country': 'Egypt'})
6  # Insert multiple documents
7  employees_collection.insert_many([
8      {'name': 'Ali', 'age': 35, 'country': 'Saudi Arabia'},
9      {'name': 'Mohammed', 'age': 40, 'country': 'Kuwait'},
10     {'name': 'Sara', 'age': 20, 'country': 'UAE'}
11 ])
```



Starting with MongoDB (Query Documents)

You can query single single document or multiple documents using python dictionary or JSON as conditions to filter your search, use the `find_one` method to get the first one that matches your query or a list of dictionaries using `find` method.

```
1  # Find documents
2  first_jhon = employees_collection.find_one({'name': 'John'})
3  print(first_jhon)
4  # Find all the documents in the collection with the name 'John'
5  results = employees_collection.find({'name': 'John'})
6  for employee in results:
7      print(employee)
8      print(f'Age: {employee['age']}')
9      print(f'Country: {employee['country']}')
10     print('-----')
```



Starting with MongoDB (Update Documents)

You can update a single document or multiple documents using a Python dictionary or JSON as conditions to filter the documents to update. Use the `update_one` method to update the first document that matches your condition or use the `update_many` method to update multiple documents.

```
1  # Update single document
2  employees_collection.update_one({'name': 'John'}, {'$set': {'age': 30}})
3
4  # Update multiple documents
5  employees_collection.update_many({'name': 'John'}, {'$set': {'country': 'China'}})
```



Starting with MongoDB (Delete Documents)

You can delete a single document or multiple documents using a Python dictionary or JSON as conditions to filter the documents to delete. Use the `delete_one` method to delete the first one that matches your condition or use the `delete_many` method to delete multiple documents.



```
1  # Delete single document
2  employees_collection.delete_one({'name': 'Jane'})
3
4  # Delete multiple documents
5  employees_collection.delete_many({'name': 'John'})
```





Tutorials and Exercises

Tutorial (Notebook):

3- Advanced Machine Learning/1-Introduction to Database
/LAB/No_SQL.ipynb

Exercises:

3- Advanced Machine Learning/1-Introduction to Database
/LAB/NoSQL_ Exercises.pdf

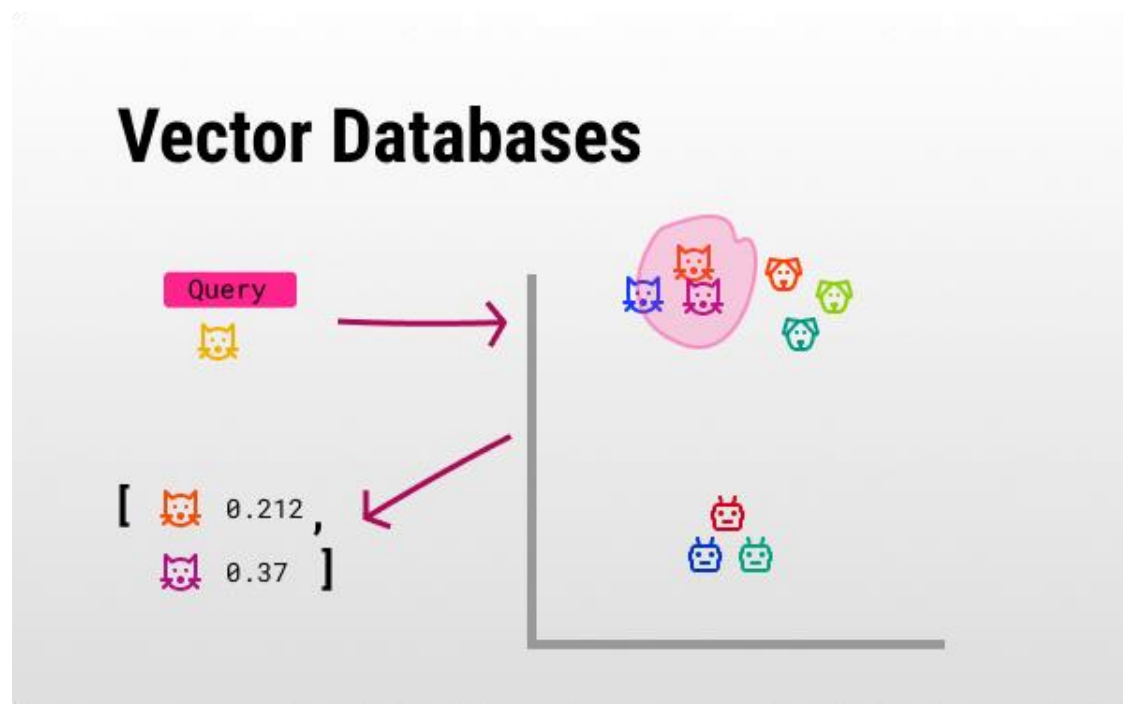


Vector Database

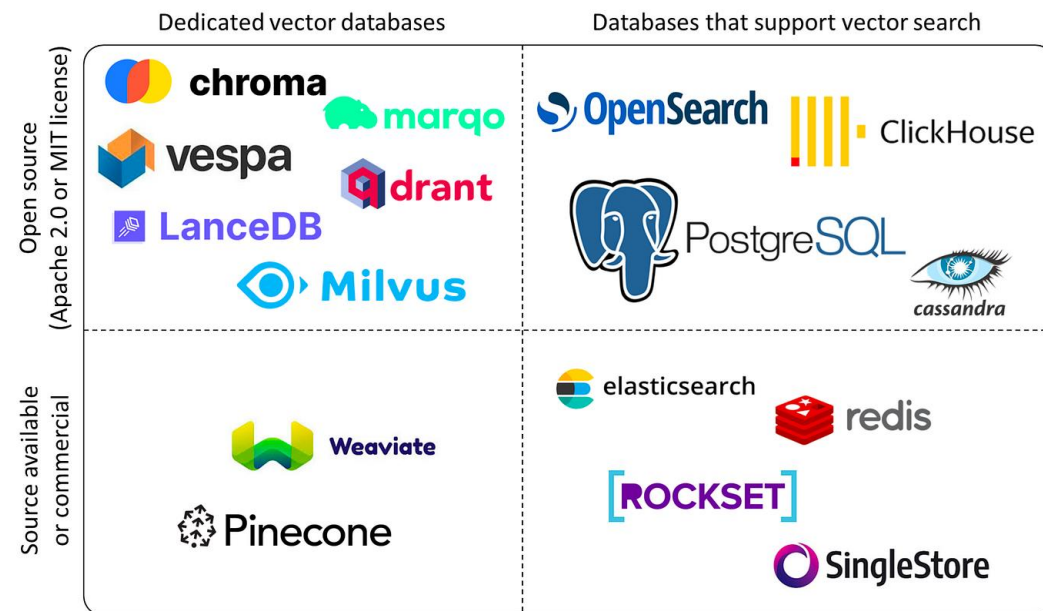


Introduction to Vector Databases

Vector databases are specialized databases designed to handle high-dimensional vector data. Essential for applications in AI, machine learning, and data science that involve similarity searches, clustering, and nearest neighbor search.



[Source](#)

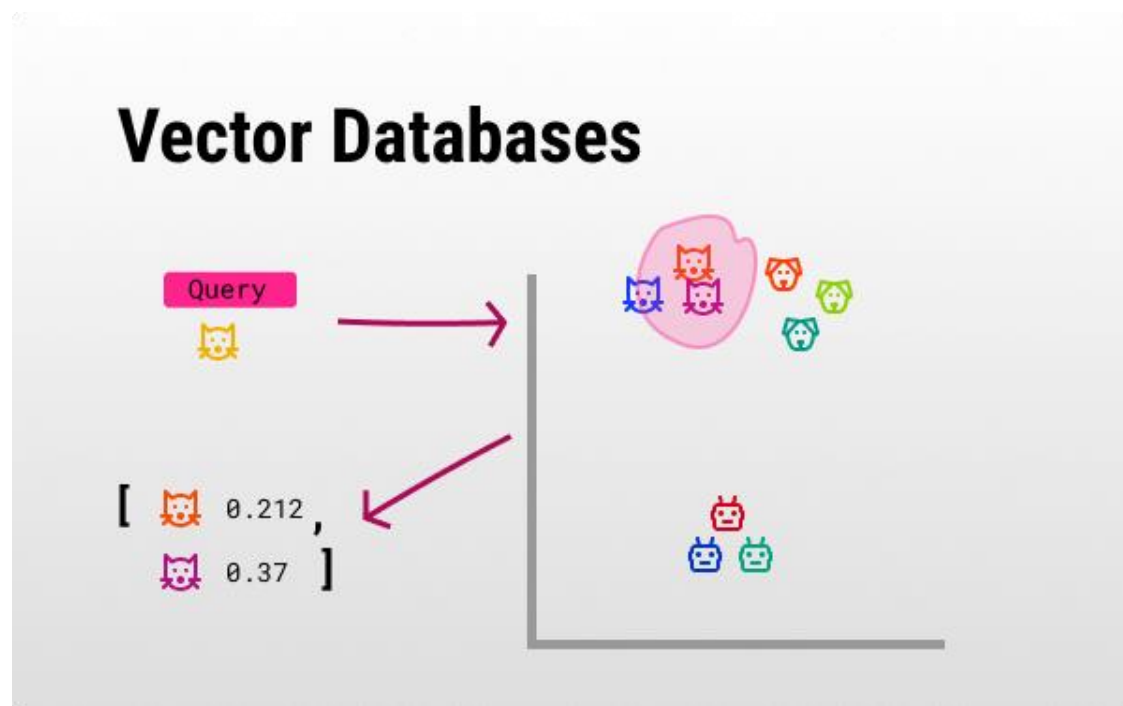


[Source](#)

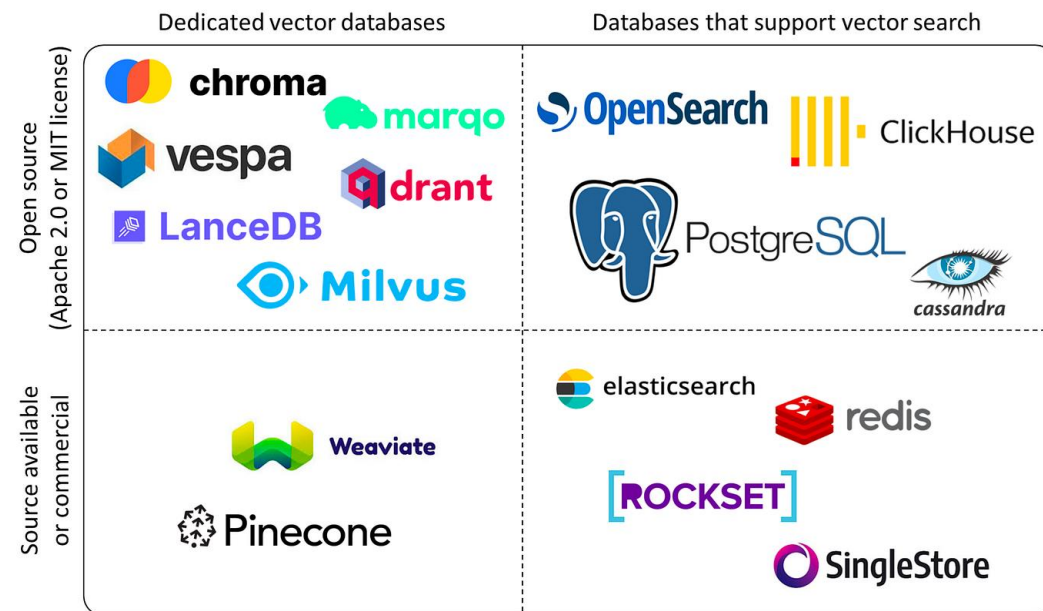


What does Vector Database

Vector databases store embeddings, which are high-dimensional vector representations of various inputs such as words, sentences, images, and audio. These embeddings capture the semantic information of the input data. We will explore this topic in more detail later.



[Source](#)

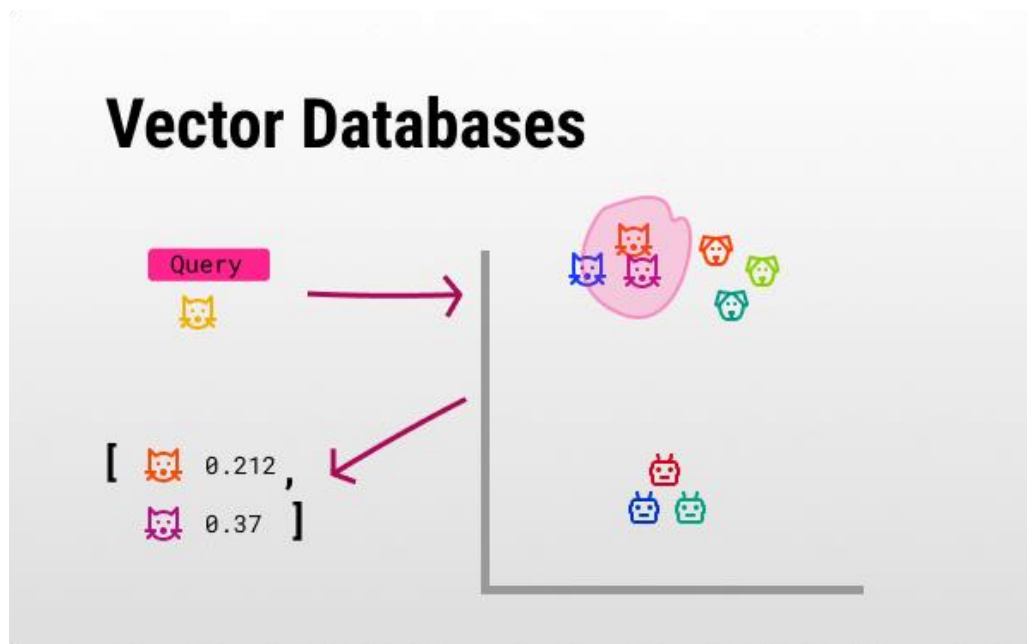


[Source](#)

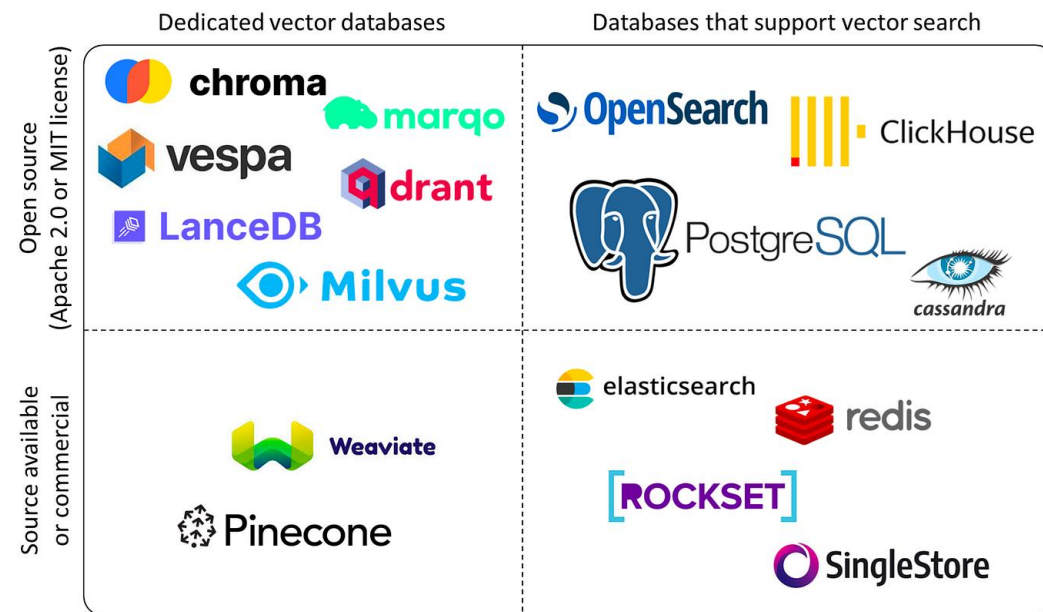


Why not RDMBS/NoSQL databases?

While these databases are excellent at managing structured data with fixed schemas, they often face challenges with unstructured or high-dimensional data like images, audio, and text. Traditional databases are not optimized for efficiently searching for similar items within large datasets, particularly when dealing with high-dimensional vector data, resulting in poorer performance at scale.



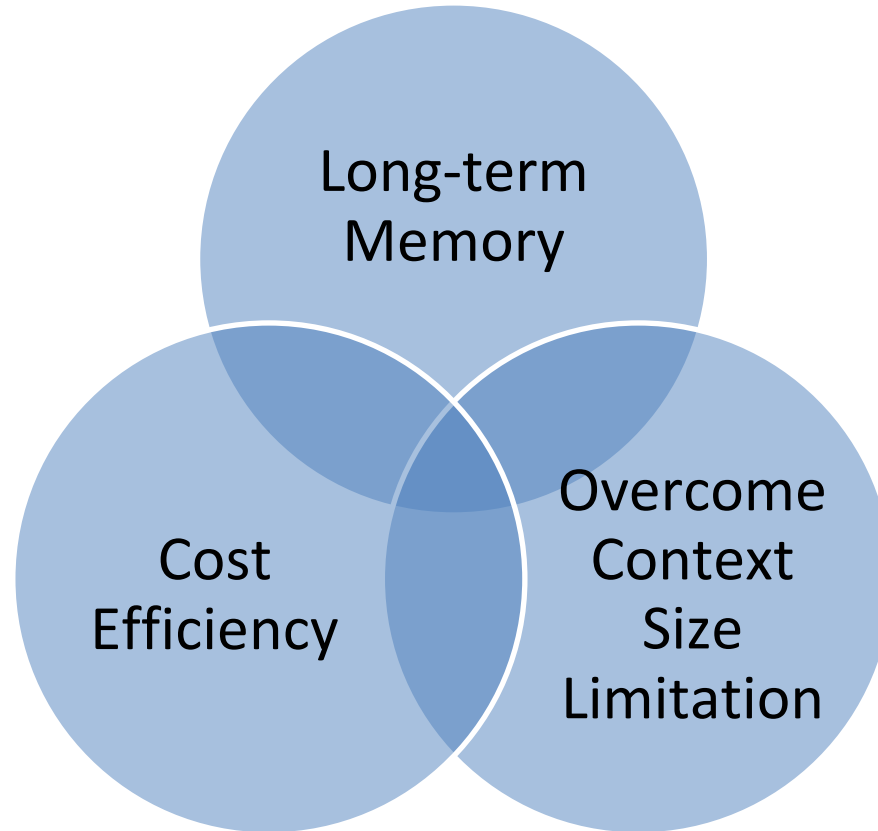
[Source](#)



[Source](#)



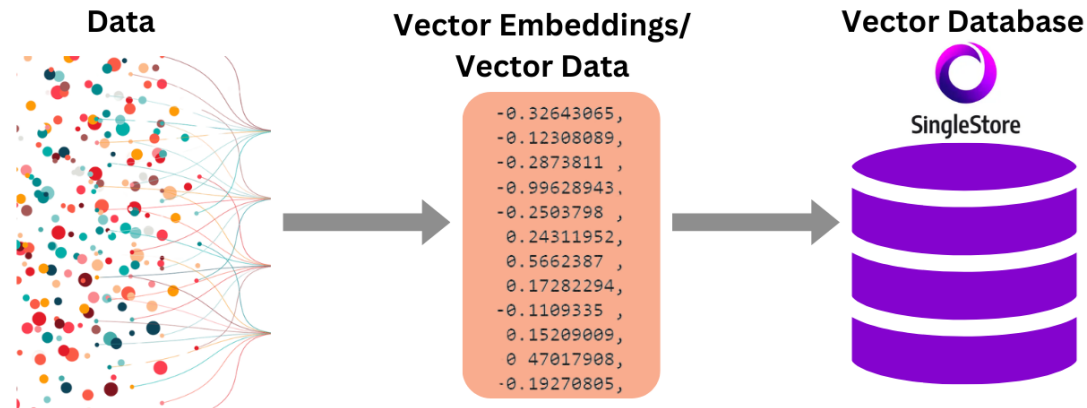
Why Should We Use Vector Databases



Long-term Memory

Storing Information for Later Use:

- Encoding data like text, images, or audio into embeddings helps retain their meanings.
- Similar data is stored closer together, facilitating better information retrieval.
- Enables your product to persistently retain and manage large amounts of knowledge over time.



[Source](#)

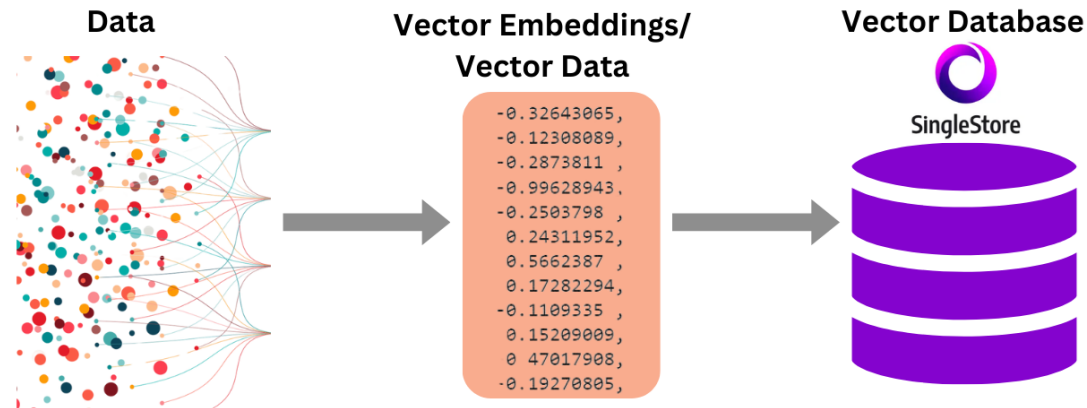




Overcome Context Size Limitation

Handling Large Data Efficiently:

- Vector databases store shorter sections of large data within the prompt context.
- Fetch only the semantically relevant sections from the vector database.
- Overcome token limitations by using relevant shorter sections.



[Source](#)

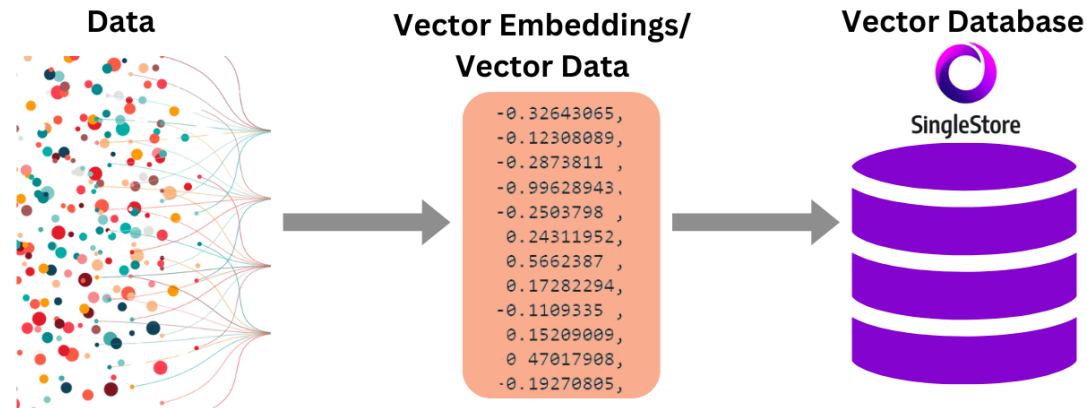




Cost Efficiency

Cost-effective Solution:

- Generating embeddings is cheaper than invoking a full LLM API call.
- Querying embeddings against the database is much more affordable.
- Embeddings retain semantic information, offering a balance between cost and performance.



[Source](#)



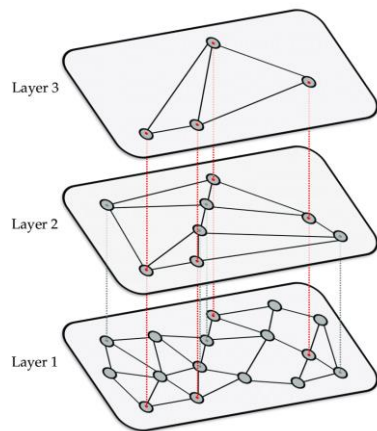


Key Features of Vector Databases

Vector databases offers various features such as:

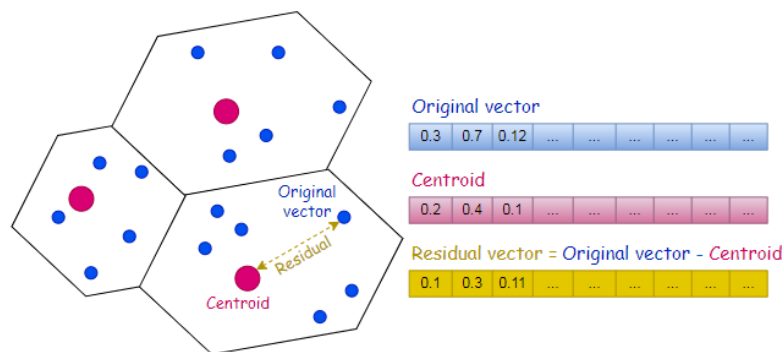
- **Indexing:** Advanced indexing techniques (e.g., HNSW, IVF, PQ) for fast similarity searches.
- **Distance Metrics:** Support for various distance metrics (e.g., Euclidean, cosine similarity).
- **Scalability:** Distributed architectures for horizontal scaling.
- **Integration:** Seamless integration with machine learning frameworks and tools.

hierarchical navigable small world (HNSW)



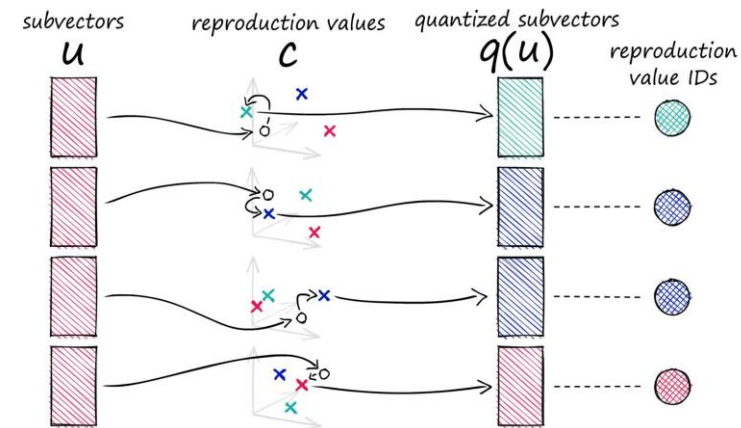
[Source](#)

inverted file index (IVF)



[Source](#)

Product Quantization (PQ)



[Source](#)





Tutorials and Exercises

Tutorial (Notebook):

3- Advanced Machine Learning/1-Introduction to Database
/LAB/vectorDB.ipynb



Collecting Data from Various Sources

Introduction to Data Collection Methods

Data Collection is a foundational step in the field of data science and decision-making.

It is gathering information from various sources to create datasets that accurately represent the phenomenon or subject of interest.

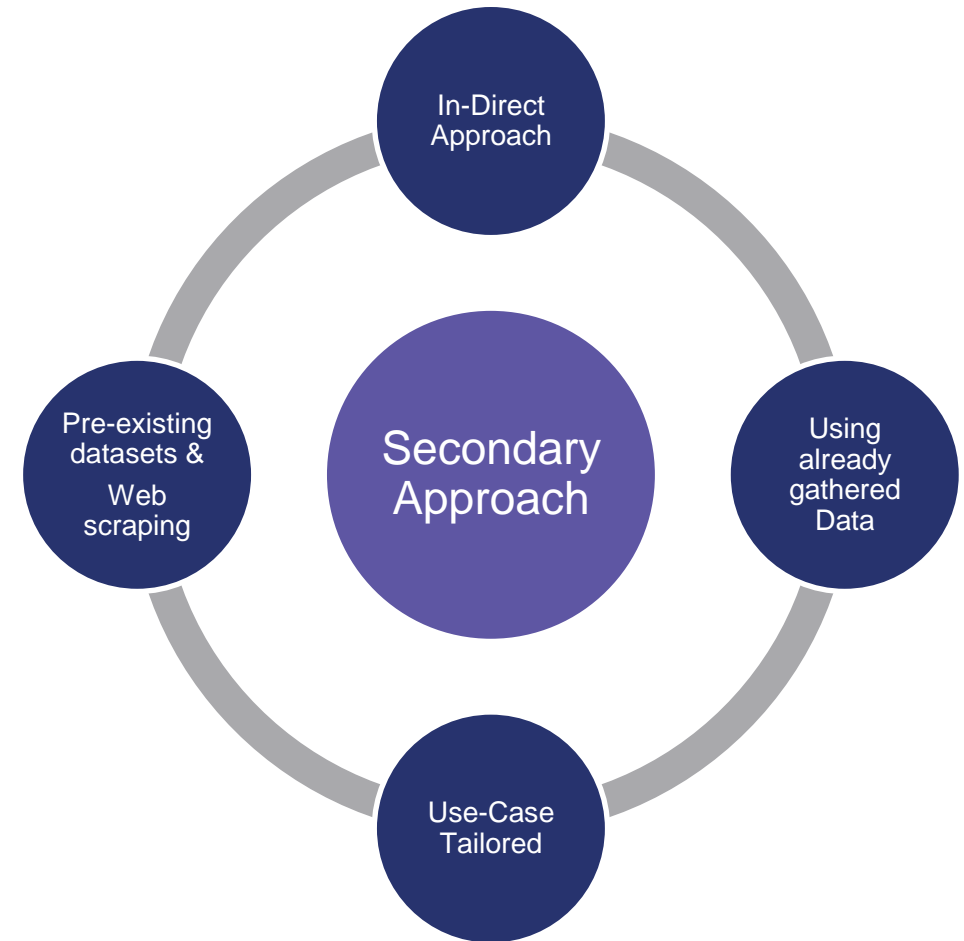
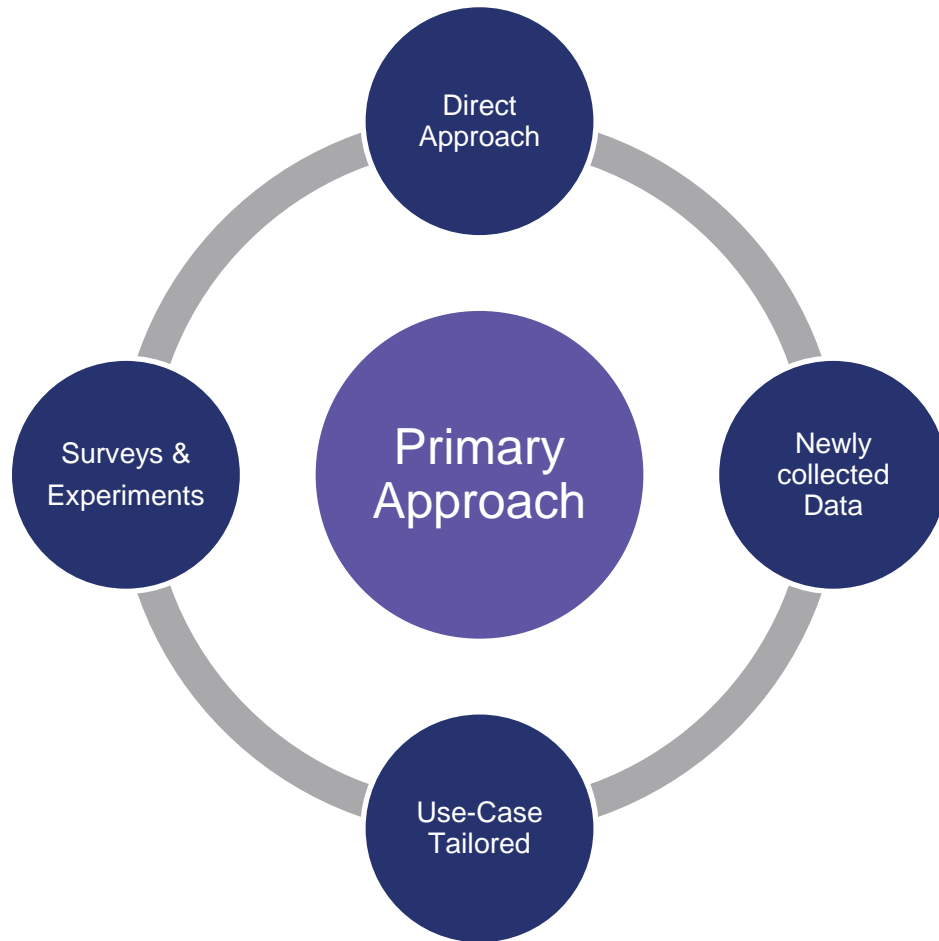
This process can be performed using *surveys, experiments, web scraping, and accessing public datasets or APIs.*

Data Collection Considerations:

The goal of data collection in data science is to amass data that is **relevant, accurate, and of high quality.**



Primary versus Secondary Approaches





Direct Data Collection Approach

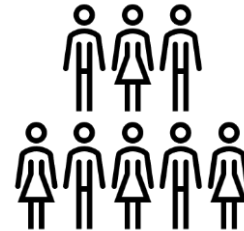
Primary Approach

The Direct Data Collection Approach refers to a method of gathering information straightforwardly from the source for the first time.

Surveys and experiments are two fundamental methods for data collection in various fields, including *social sciences*, *marketing*, *health research*, and *many areas of data science*.



Surveys



Experiments





Survey

Direct Data Collection Approach

Surveys gather self-reported data through formats like online questionnaires and interviews. They're useful for collecting opinions and behaviors.

Their advantages include:

Scalability: Efficiently reach many respondents.

Versatility: Collect diverse data, from demographics to opinions.

Comparability: Standardized questions facilitate cross-group analysis.

Limitations include biases from question phrasing, respondent interpretation, and accuracy of responses





Experiments

Direct Data Collection Approach

Experiments manipulate variables to study effects and infer causality, often under controlled settings like labs, though they can also be in the field or online.

Key features include:

Control: Ability to manage conditions and isolate variables.

Randomization: Random assignment to groups to limit bias and support causal conclusions.

Repeatability: Can be replicated to confirm findings.

Valuable for exploring cause-and-effect

Limitations include high costs, time demands, and practical or ethical limitations.





In Direct Data Collection Approach

Secondary Approach

Indirect(Secondary) Approach: Involves using data that has already been collected by someone else for a different purpose and leveraging existing resources to gather information that can be applied to the current research.

Web scraping and API Calling are two fundamental secondary methods for data collection:



API



Web Scraping





Web Scraping for Data Collection

In-Direct Approach

Web scraping is the process of extracting data from websites, automating the collection of information available online. It serves as a powerful tool in data collection, enabling analysts and scientists to gather vast amounts of data quickly, which is essential for analysis, research, and decision-making processes.

Key tools and technologies for web scraping include:



Beautiful Soup: A Python library for parsing HTML and XML documents. It's widely used for simple projects and tasks that require quick data extraction from websites.



Selenium: Originally a tool for testing web applications, Selenium can automate web browser interaction, making it suitable for scraping dynamic content that requires interaction with the webpage.



Scrapy: An open-source and collaborative framework for extracting the data you need from websites. It's designed for web scraping. Scrapy is highly efficient, scalable, and versatile, making it suitable for large-scale web scraping projects.





Beautiful Soup for Data Collection

In-Direct Approach

Typical Steps to handle a website in Beautiful Soup

- Fetching the web page content using requests.
- Parsing the content with Beautiful Soup to create a parse tree.
- Using Beautiful Soup's searching and navigation methods to find relevant data.
- Extracting and processing the data you need from the elements found.
- Iteratively refining your approach based on the specific requirements of your web scraping project and the structure of the web pages you're working with.



Selenium for Data Collection

In-Direct Approach

What is Selenium?

- An open-source automation tool primarily used for automating testing web applications.
- Allows for browser automation, enabling tasks to be performed as if a real user is navigating the site so it can also render websites Dynamically.

Why Use Selenium for Web Scraping?

- **Dynamic Content:** Selenium can interact with webpages that load content dynamically, making it ideal for scraping modern sites.
- **Real Browser Interaction:** Performs operations in a real browser environment, allowing for actions like clicking buttons, filling forms, and scrolling.





Tutorials and Exercises

Tutorial (Notebook):

3- Advanced Machine Learning/1-Introduction to Database
/LAB/web_scrape.ipynb

Exercises:

3- Advanced Machine Learning/1-Introduction to Database
/LAB/web_scrape_exercise.ipynb





API Access for Data Collection

In-Direct Approach

APIs (Application Programming Interfaces) are software are tools that allow different software applications to communicate with each other. They acts as intermediaries allowing different software applications to communicate, simplifying the process of data collection by providing structured ways to request and receive data.

Advantages of Using APIs:

Efficiency: Streamlines data access and functionality.

Real-Time Data: Offers access to live data, crucial for up-to-date application needs.

Scalability: Eases handling of growing data or demand with minimal infrastructure adjustments.

Cost-Effectiveness: More affordable than developing custom data collection systems.

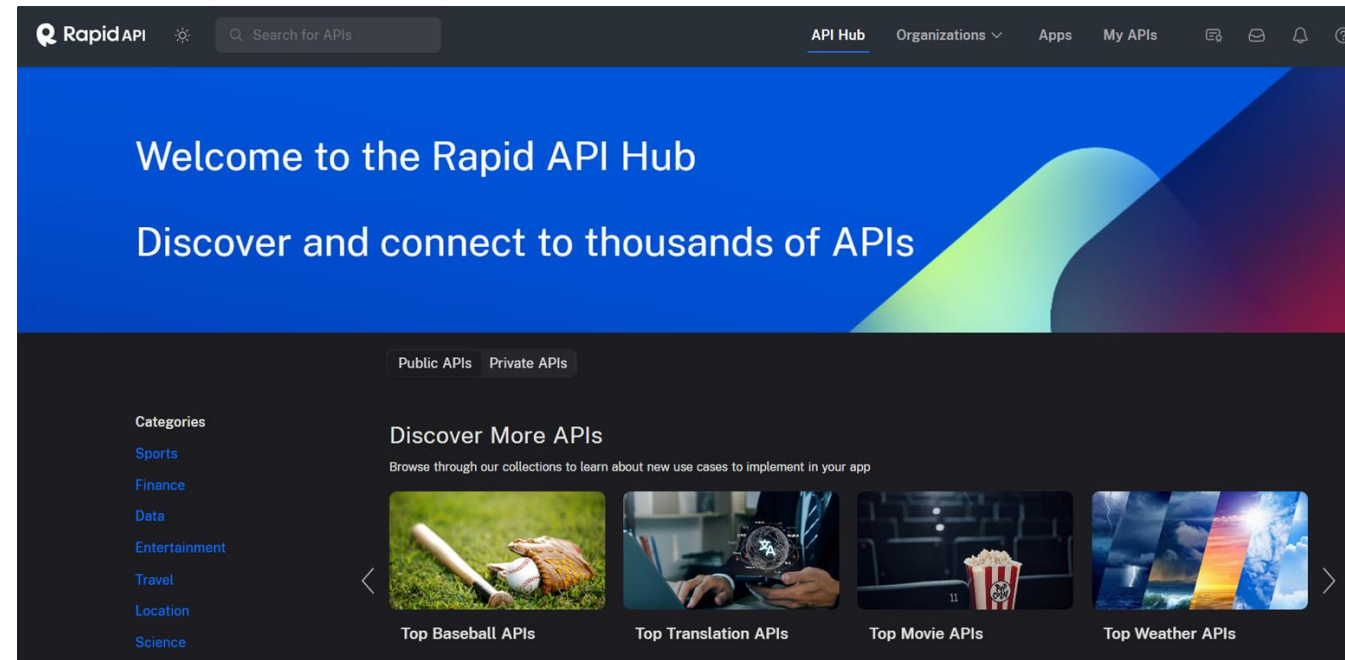





RapidAPI [\[link\]](#) is a comprehensive platform that aggregates thousands of APIs across various domains

It presents a unified platform for developers to discover, connect, and manage APIs through a single, standardized interface.

It offers access to diverse data sources across various categories, including finance, sports, entertainment, weather, and more.





RapidAPI

Considerations

In-Direct Approach

API Limits: Be aware of rate limits and quotas to avoid service interruptions.

Costs: Understand the pricing model of the API and usage charges.

Security: Keep your API key confidential to prevent unauthorized usage.

Performance: Test response times and reliability.

Documentation: Read the API documentation thoroughly.

Updates: Stay informed about any changes or updates to the API.

Support: Check the support options and community forums for help for Q&A.





Getting Data from

Accessing high-quality data from various platforms is crucial for data science projects. In this section, we explore three key platforms: **Kaggle**, **GitHub**, and **Hugging Face** 😊.



[Source](#)





Kaggle is a popular platform for data science competitions, providing a variety of datasets for machine learning and data analysis. Kaggle provides many features like:

- **Datasets:** Access thousands of datasets across numerous domains.
- **Competitions:** Participate in or view ongoing data science challenges.
- **Kernels (Notebooks):** Explore code and analysis shared by others.





GitHub is a platform for version control and collaboration, hosting a vast number of open-source projects that often include datasets and code. GitHub provides many features like:

- **Repositories:** Explore and fork repositories that contain datasets and code.
- **Issues and Discussions:** Engage with the community for support and data-related discussions.





Tutorials and Exercises

Now let's try to save the dataset in MongoDB and how to upload the model's weights.

Tutorial (Notebook):

3- Advanced Machine Learning/1-Introduction to Database/LAB/Storing&Retrieving_ML_Models.ipynb

Exercises:

3- Advanced Machine Learning/1-Introduction to Database/LAB/Storing&Retrieving_ML_Models_Exercise.ipynb

