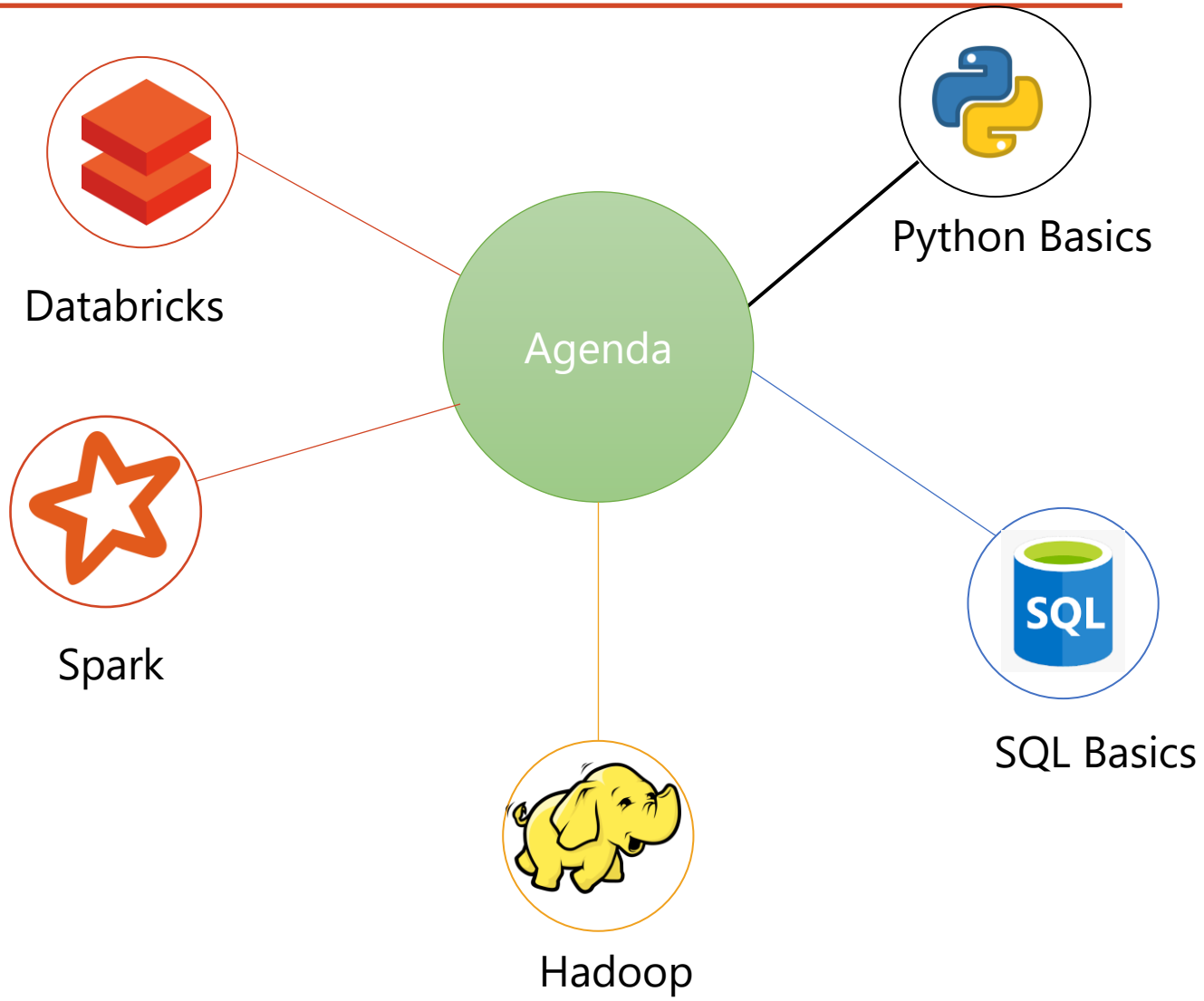
DataBricks Training

By: Gaurav Gangwar

Agenda of the training

Topics to be covered:

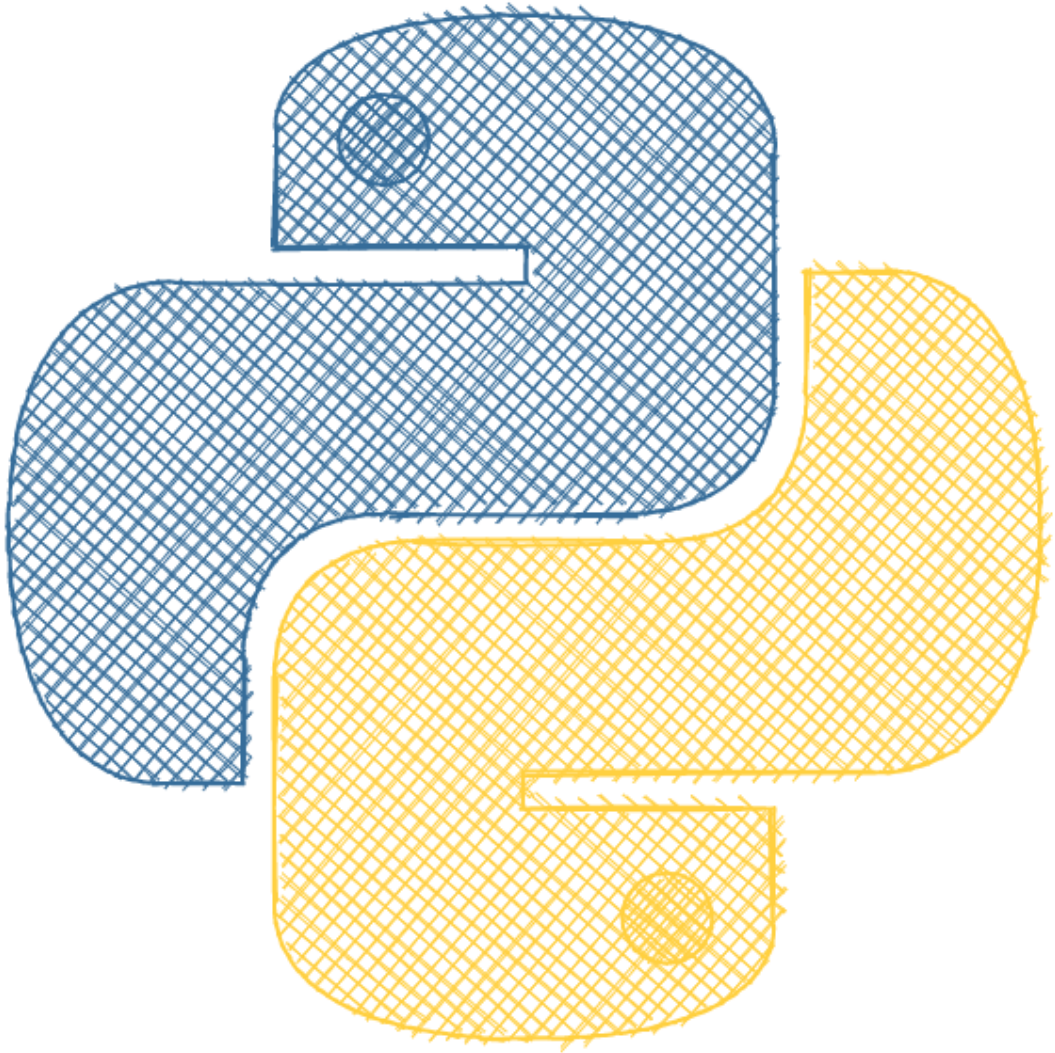
- ❑Python Basics
- ❑SQL Basics
- ❑Hadoop
- ❑Apache Spark
- ❑Databricks



Pre Requisite

- Visual Studio Code
- Python IDE
- Databricks Community Account
- Snowflake Account
- Free Azure Account*
- Anaconda IDE*

Basics of Python



Python Topics

Introduction	Lists
Syntax	Tuples
Comments	Sets
Variables	Dictionaries
Data Types	If...Else
Numbers	While Loops
Casting	For Loops
Strings	Functions
Booleans	Lambda
Operators	Arrays

Python Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- Web development (server-side),
- Software development,
- Mathematics,
- System scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Python Syntax

Python QuickStart

- Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

- Command to check python version:

```
C:\Users\gaura>python --version
Python 3.9.7
```

- The way to run a python file is like this on the command line:

- Execute Python file Syntax:

```
C:\Users\Your Name>python helloworld.py
```

- Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Python Comments & Variables

Python Comments

- Comments can be used to explain Python code and to make the code more readable
- Comments can be used to prevent execution when testing code.

Creating a Comment

Comments starts with a `#`, and Python will ignore them:

Example

```
#This is a comment
print("Hello, World!")
```

Python Comments & Variables

Python Variable

- We do not need to declare variables before using them or declare their type.

- A variable is created the moment we first assign a value to it.

Example

```
x = 5
y = "John"
print(x)
print(y)
```

Python Case Sensitivity

Case Sensitive

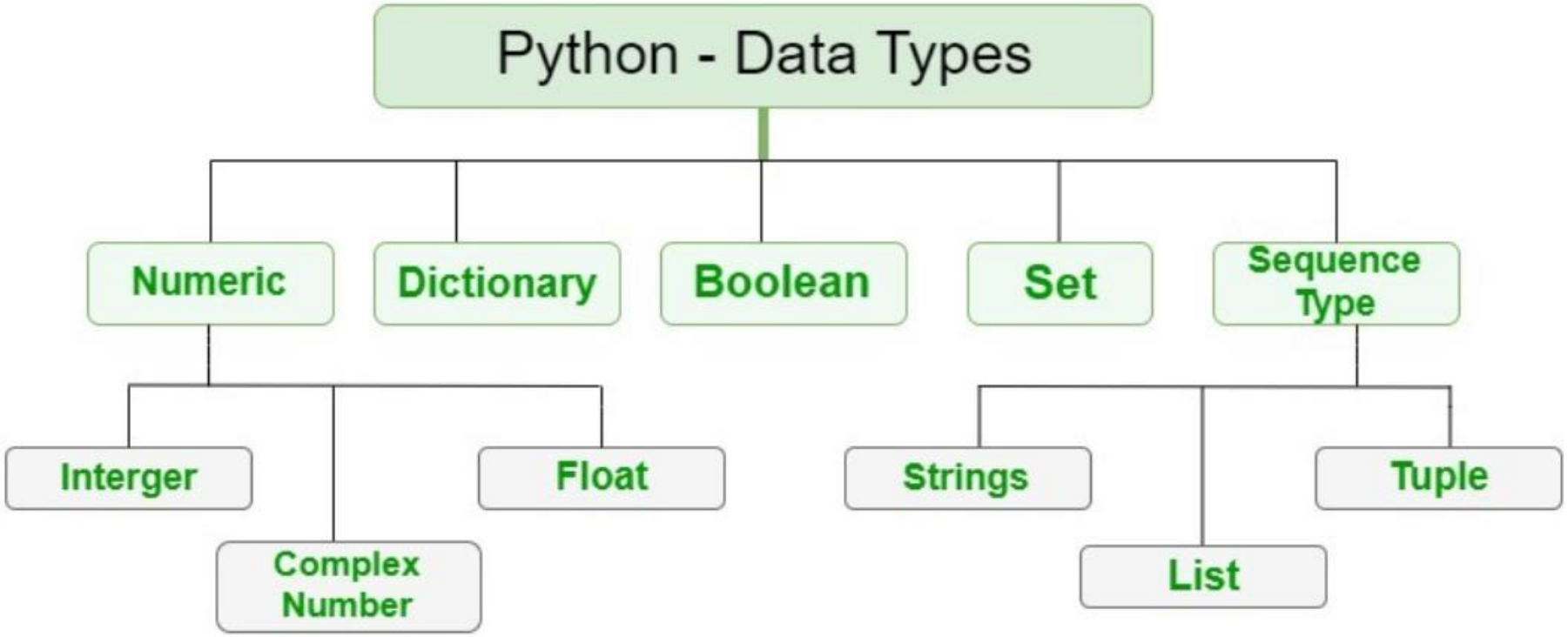
Variable names are case-sensitive.

eg:

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

Python Data Type



Python Data Type

- **Variables can store data of different types, and different types can do different things.**
- **Python has the following data types built-in by default, in these categories:**

Group	Data Type
Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Text Type

- Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

- **eg:**
```
a = "Hello"
print(a)
```

- Multiline Strings & String Operations:

eg:

- You can use three double quotes:

- ```
  a = """Lorem ipsum dolor sit amet,
  consectetur adipiscing elit,
  sed do eiusmod tempor incididunt
  ut labore et dolore magna aliqua."""
  print(a)
  ```
  ```

Numeric Type

There are three numeric types in Python:

•`int()` - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

•`float()` - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

•`complex()` - Complex numbers are written with a "j" as the imaginary part

Variables of numeric types are created when you assign a value to them:
Eg:

```
x = 1  # int
y = 2.8 # float
z = 1j # complex
```

Sequence Type

- **List:** Lists in Python can be created by just placing the sequence inside the square brackets[].

eg:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

- **Tuple:** Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created

eg:

```
tuple = (‘big', ‘data’)

print("\nTuple with the use of String: ")

print(tuple)
```

- **Range:** The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

eg: range(start, stop, step)

```
x = range(3, 6)
for n in x:
  print(n)
```

Mapping Type

- **Dictionary:** Dictionaries are used to store data values in key:value pairs.
- **Changeable:** Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
- **Ordered or Unordered?**
- As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.
- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
- Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.
- Dictionaries are written with curly brackets, and have keys and values:

eg:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Set Type

Set: Sets are used to store multiple items in a single variable

- Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
- A set is a collection which is unordered, unchangeable*, and unindexed.
- **Note:** Set items are unchangeable, but you can remove items and add new items.
- **Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

eg:

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Python Operator

Python divide operators mentioned below:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Conditional Statement

- Python supports the usual logical conditions from mathematics:
- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- These conditions can be used in several ways, most commonly in "if statements" and loops.
- An "if statement" is written by using the if keyword.

Python Conditional Statement

Some Examples:

If:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Elif:

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python Conditional Statement

- **If Placement in top 10 Companies.**

Python Conditional Statement

- **Elif Placement in top 500 Companies**

Python Conditional Statement

- **Else Placement in top 1000 Companies .**

Python Loops

Python has two primitive loop commands:

- **For Loop:** A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

Eg:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

- **While Loop:** With the while loop we can execute a set of statements as long as a condition is true.

Eg:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Python Function

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Eg:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function(“Amber", ”Jd")


Function with return Statement:

def add(a, b):

    return a + b


# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))
```

Python Lambda

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

Syntax: `lambda *arguments* : *expression*`

Eg:

```
x = lambda a : a + 10
print(x(5))
```

- The power of lambda is better shown when you use them as an anonymous function inside another function.

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

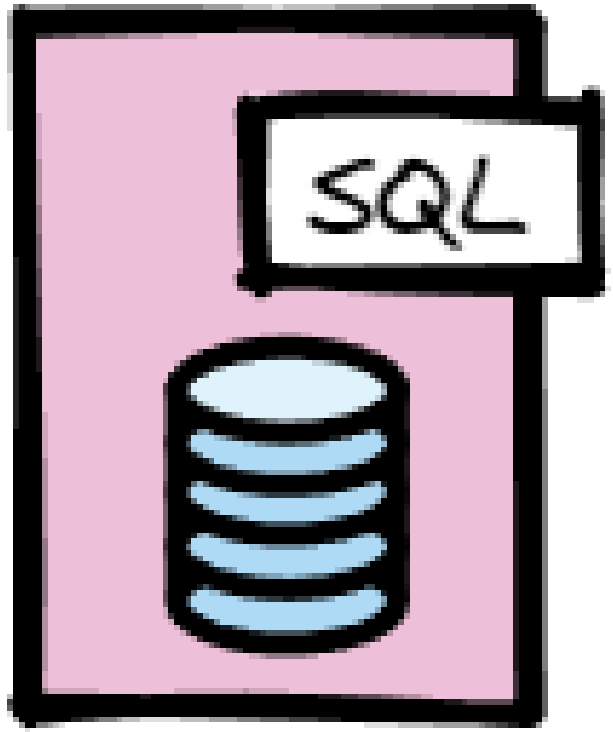
print(mydoubler(11))
```

More Python Deep Dive

- More Deep Dive Into Python

Time for 10 Min Break

Basics of SQL

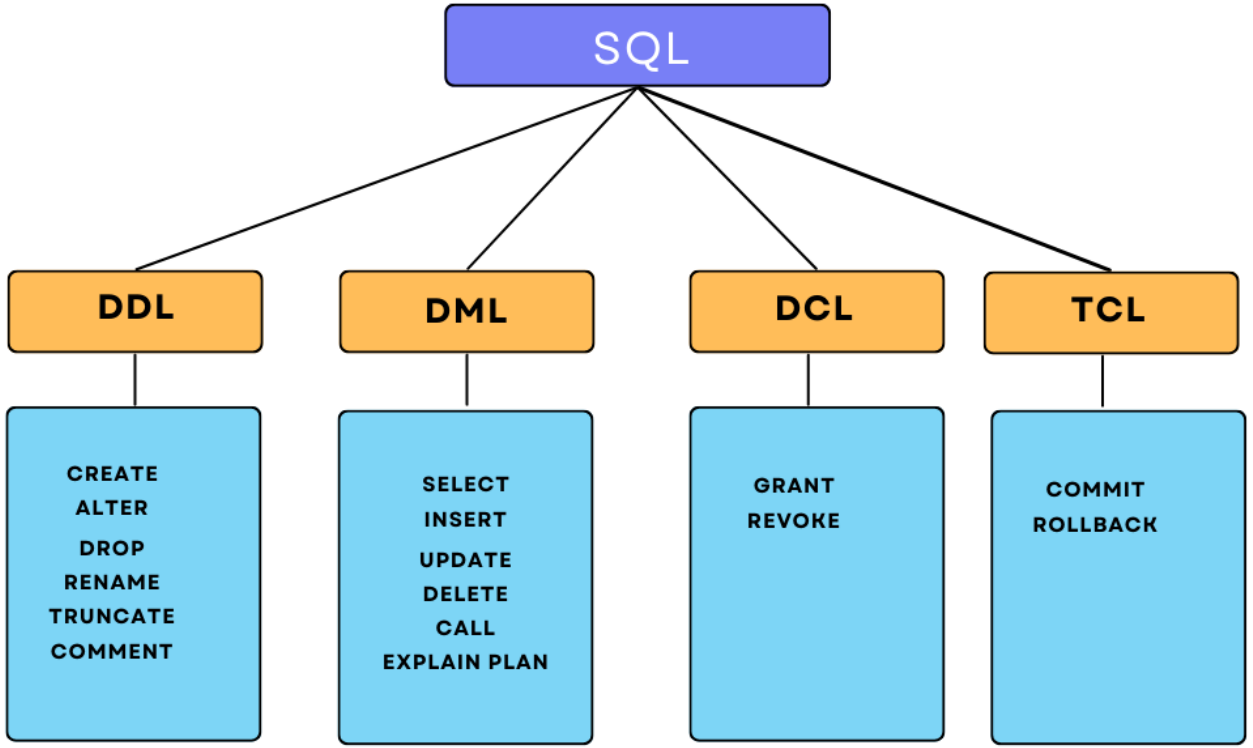


SQL

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

Category Flow Chart



SQL Statements

•Most of the actions you need to perform on a database are done with SQL statements.
•The following SQL statement selects all the records in the "Customers" table:

Eg:

•SELECT * FROM Customers;

Some of The Most Important SQL Commands:

•SELECT - extracts data from a database
•UPDATE - updates data in a database
•DELETE - deletes data from a database
•INSERT INTO - inserts new data into a database
•CREATE DATABASE - creates a new database
•ALTER DATABASE - modifies a database
•CREATE TABLE - creates a new table
•ALTER TABLE - modifies a table
•DROP TABLE - deletes a table
•CREATE INDEX - creates an index (search key)
•DROP INDEX - deletes an index

SELECT, Update & Delete

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

Eg:

```
SELECT * FROM table_name;
```

- The UPDATE statement is used to modify the existing records in a table.

Eg:

```
UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;
```

- The DELETE statement is used to delete existing records in a table.

Eg:

```
DELETE FROM Customers WHERE CustomerName='xyz';
```

The SQL AND, OR and NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND & OR operators are used to filter records based on more than one condition:
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- ```
  SELECT * FROM Customers
  WHERE City='Berlin' OR City='München';
  ```
  ```

SQL Function

COUNT()- The COUNT() function returns the number of rows that matches a specified criterion.

Syntax:

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

AVG()-The `AVG()` function returns the average value of a numeric column.

Syntax:

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SUM()-The `SUM()` function returns the total sum of a numeric column.

Syntax:

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

SQL Function

MIN()-The MIN() function returns the smallest value of the selected column.

Syntax:

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

MAX()-The MAX() function returns the largest value of the selected column.

Syntax:

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

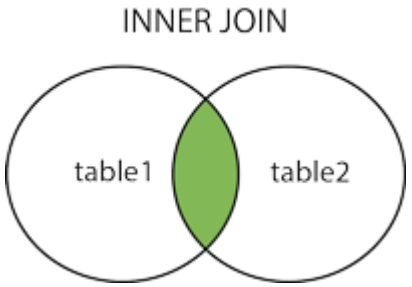
Check Example

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

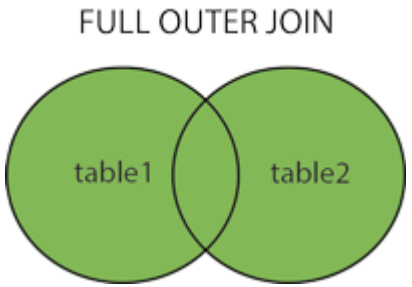
- `(INNER) JOIN`: Returns records that have matching values in both tables.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```



- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

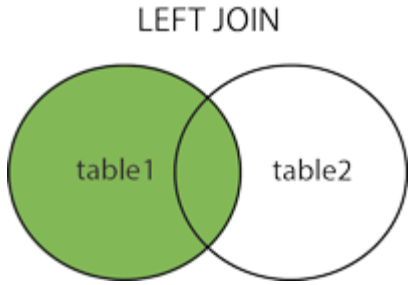


Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

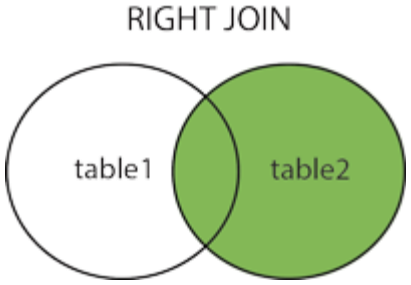
•`LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```



•`RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table.

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```



Check Example

Union Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order.
- ```
  SELECT City FROM Customers
  UNION
  SELECT City FROM Suppliers
  ORDER BY City;
  ```

Check Example
  ```

SQL CASE Expression

- The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

- If there is no ELSE part and no conditions are true, it returns NULL.

Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

SQL EXISTS Operator

- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.

Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

Group BY & Having Clause

Group By:

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

Having:

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.
- But it can we access with group by.

Syntax

- SELECT column_name(s)
- FROM table_name
- WHERE condition
- GROUP BY column_name(s)
- HAVING condition
- ORDER BY column_name(s);

More SQL Deep Dive

- More Deep Dive Into SQL

Thank You For the day

See You Tomorrow