Overview tables

Sunday, June 11, 2023 1:20 PM

	File Systems	Databases	Excel Sheets
Description	A method for organizing and storing files	Structured storage for managing large datasets	Spreadsheet software for data organization
Advantages	- Simple and easy to use	- Efficient data retrieval and management	- Familiar interface for data entry and analysis
	- No additional software or setup required	- Data integrity and security controls	- Calculation capabilities and formulas
	- Suitable for small-scale and simple data storage	- Concurrent access and multi-user support	- Charting and graphing functionalities
	- Can handle various file types and formats	- ACID compliance for transactional operations	- Data sharing and collaboration capabilities
Limitations	- Lack of data consistency and standardization	- Requires setup and administration (e.g., schemas)	- Limited scalability for large datasets
	- Difficult to search, query, and retrieve data	- Higher complexity for implementation and setup	- Risk of data corruption and formula errors
	- Limited data organization and relationship	- Costly in terms of licensing and infrastructure	- Lack of robust data validation and auditing
	- Limited concurrency and multi-user capabilities	- Requires SQL knowledge for querying	- Lack of data versioning and revision control

Database Type	Short Description	Advantages	Limitations
SQL Relational	Stores data in structured tables with relationships	- Data integrity and consistency	- Complex setup and administration
Database		- Flexible querying capabilities	- Scalability challenges for large datasets
		- ACID compliance for transactional operations	- Requires SQL knowledge for querying
		- Established and mature technology	- Not ideal for unstructured or hierarchical data
Analytical (OLAP)	Optimized for complex data analysis and reporting	- High performance for analytical queries	- Not suitable for real-time or transactional data
Database		- Aggregation and multidimensional data modeling	- Requires specialized knowledge for optimization
		- Efficient handling of large datasets	- Limited transactional capabilities
Key-Value	Stores data as key-value pairs	- Simplicity and high scalability	- Limited query and data manipulation capabilities
Database		- Fast retrieval and high write throughput	- Lack of complex data relationships
		- Flexible schema and easy horizontal scaling	- Limited data indexing and querying capabilities
Column Family	Organizes data into column families	- High scalability and efficient write performance	- Limited support for complex data relationships
Database		- Flexible schema and fast data retrieval	- Limited query capabilities compared to SQL

		- Efficient for sparse and wide data	- Higher storage overhead compared to key-value
Graph-based	Stores data as nodes and edges for graph analysis	- Powerful for complex relationships and graph data	- Less suitable for simple, tabular data
Database		- Efficient traversal and querying of graph structures	- Limited scalability for large datasets
		- Suitable for social networks and recommendation systems	- Higher complexity for data modeling and querying
Document-based	Stores data in flexible, self-describing documents	- Flexible schema and easy data manipulation	- Lack of complex relationship support
Database		- Suitable for semi-structured and unstructured data	- Limited query capabilities compared to SQL
		- Easy horizontal scaling and high availability	- Higher storage overhead for denormalized data

Query	Description	Example
SELECT	Retrieves data from a database table	SELECT * FROM Customers
UPDATE	Modifies existing data in a database table	UPDATE Customers SET city = 'New York' WHERE id = 1
DESC	Describes the structure of a database table	DESC Customers
ALTER	Modifies the structure of a database table	ALTER TABLE Customers ADD email VARCHAR(255)
USE DB	Selects a specific database for use	USE mydatabase
MODIFY	Alters the structure of a column in a database table	ALTER TABLE Customers MODIFY city VARCHAR(100)
WHERE	Filters data based on specified conditions	SELECT * FROM Customers WHERE country = 'USA'
CREATE	Creates a new database, table, or other database objects	CREATE TABLE Employees (id INT, name VARCHAR(100))
SHOW	Displays information about databases, tables, or other objects	SHOW TABLES
DROP	Removes a database, table, or other database objects	DROP TABLE Customers
DELETE	Deletes data from a database table	DELETE FROM Customers WHERE id = 1
INSERT INTO	Inserts new data into a database table	INSERT INTO Customers (name, email) VALUES ('John', 'john@example.com')
INNER JOIN	Retrieves data from multiple tables based on a matching condition	SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID
LEFT JOIN	Retrieves data from two tables based on a matching condition, including unmatched rows from the left table	SELECT Customers.CustomerName, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
RIGHT JOIN	Retrieves data from two tables based on a matching condition, including unmatched rows from the right table	SELECT Customers.CustomerName, Orders.OrderID FROM Customers RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
VIEW	Creates a virtual table based on the	CREATE VIEW SalesView AS SELECT * FROM

	result of a query	Sales
PROCEDURES	Executes a sequence of SQL statements as a named routine	CREATE PROCEDURE GetCustomerByID (IN customerId INT) BEGIN SELECT * FROM Customers WHERE id = customerId; END;

Description	Format	Example of Opening a File in Python	Pros	Cons

Binary	binary	# open binary file
		f = open("data.bin", "rb")
		# read bytes
		bytes = f.read(4)
		do stuff with bytes
		# close binary file f.close()
			- Compact storage
			- Fast read and write
				- Not human-readable
				- Encoding must be known (extra file?)
				- Read and write methods must be implemented

CSV	Text	import csv with open('data.csv') as f: reader = csv.reader(f) for row in reader: print(row)	- Simple and widely supported - Libraries exist - Easy to import/export data	- Requires knowledge of column data types - Requires knowledge of delimiter and quote characters - Missing a default way to include metadata
XML	Text	Import xml.etree.ElementTree as ET tree = ET.parse('data.xml') root = tree.getroot()	- Human-readable- Libraries exist - Default way of adding metadata	- Files can become very large - Elements must be known - Nesting can be complex
JSON	Text	import json with open('data.json') as f: dataseries = json.load(f)	- Human-readable - Libraries exist - Default way of adding metadata	- Names must be known
YAML	Text	import yaml with open('data.yaml') as f: dataseries = yaml.load(f)	- Human-readable - Libraries exist - Default way of adding metadata	- Keys must be known

Data model	Performance	Scalability	Flexibility	Complexity	Functionality
Relational	variable	variable	low	moderate	relational algebra
Key–Value	high	high	high	none	variable (none)
Document	high	variable (high)	high	low	variable (low)
Column	high	high	moderate	low	minimal
Graph	variable	variable	high	high	graph theory