# **Module 15-** Databases

Contents

[**Module 14-** Databases 1](#_Toc216013889)

[General Terms 2](#_Toc216013890)

[Database Roles 2](#_Toc216013891)

[Database Models 2](#_Toc216013892)

[Relational Database 3](#_Toc216013893)

[SQL 3](#_Toc216013894)

[SQLite 4](#_Toc216013895)

[**Importing sqlite** 4](#_Toc216013896)

[**SQLite and Python Types** 4](#_Toc216013897)

[SQL Statements 4](#_Toc216013898)

[**Create Database** 5](#_Toc216013899)

[**Drop Database** 5](#_Toc216013900)

[**SQLite Commands** 5](#_Toc216013901)

[Alter Table 8](#_Toc216013902)

[Multiple Table Query 9](#_Toc216013903)

[**Normalization** 9](#_Toc216013904)

[**Relational Keys** 9](#_Toc216013905)

[**Joining Multiple Tables** 10](#_Toc216013906)

[**SQLite Joins** 10](#_Toc216013907)

[Reference and Additional Materials 11](#_Toc216013908)

## General Terms

A [*database*](https://comp.mga.edu/learning/python/content/16#:~:text=4.-,Database,-(Table%2C%20Row%20and) is an electronic system that allows data to be easily manipulated, accessed, and updated. Databases are a collection of data in a structured format.

*Data* is the collection of information that describes the real-world system. It can vary in scope, format, and accessibility.

A [*Database Management System*](https://comp.mga.edu/learning/python/content/16#:~:text=1.-,Database%20Management%20Systems%20(DBMS),-Utilizing%20traditional%20file) (DBMS) is a systematic method of creating, updating, storing, and retrieving data in a database. It provides an easy to use GUI.

Data along with the DBMS is called *Database Systems*.

A [*Query Language*](https://www.tutorialspoint.com/what-is-a-query-language-in-dbms) is a specialized programming language designed specifically for creating, searchin, and retrieving data in database systems.

*Database applications* are software that helps business users interact with database systems.

*General purpose programming language* lets users create numerous programs for numerous general tasks. Examples include: Python, C++, Java, and C#

*Special Purpose Programming Language* is designed for specific functions. Examples include: SQL, HTML, MATLAB, LISP, and LaTeX.

## Database Roles

Using a database comes with many roles from user to administrator.

A *database administrator* secures data from unauthorized access and enforces procedures for providing access.

A *database designer* determines the format of each data element and the overall database structure considering different priorities like storage, response time, and data governance.

A *database programmer* creates software and/or applications that combine query language and general-purpose programming language.

A *database user* is the consumer of data in a database. They can also submit queries to database systems.

## Database Models

Data models are used to represent the structure of the data in the database. Data models capture the nature and relationships among data. We will mainly focus on the Relational Model for now.

E.F Codd introduced the [relational model](https://www.ibm.com/history/relational-database) in the 1970s. Initially the data was stored and processed as files. Components of the relational model include data structure, data manipulation, and data integrity. [Data structure](https://www.pingcap.com/article/sql-data-structures-explained/) deals with the way data is organized and accessed. It will include tables, rows, and columns. Data manipulation involves SQL operation for retrieving and modifying data such as statements like SELECT, INSERT, and UPDATE. Data integrity implements business rules that maintain integrity of manipulated data.

Here are a few other types of data models:

* [Hierarchical database model](https://mariadb.com/kb/en/understanding-the-hierarchical-database-model/)
* [Network Model](https://www.datamation.com/big-data/what-is-a-network-data-model-examples-pros-and-cons/)
* [Object-oriented database model](https://www2.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter8/node3.html)
* Entity-Relationship Model
* Document Model
* Entity-attribute-value model
* Star schema
* Object-relational model

## Relational Database

The relational database originated from the mathematical model of relational algebra. It deals with manipulation through their relationship. It has tables, rows, and columns as basic elements. Each table stores information about one thing, like orders or customers. Each column, or field, has a name and holds a specific type of data, like a number, date, or text. Each row, or record, is one entry in the table and stores related data across the columns.

In a relational database, a relation is basically a table that follows a few rules. Relations are important because they make it possible to connect data across multiple tables using shared values, like IDs. This structure helps retrieve and manage data easily with SQL.

Data should meet the following requirements to qualify as a relation:

* It must have a unique name
* Every attribute must also have a unique name and only store one value per cell.
* Every row must be unique, no two rows can be exactly the same
* Attributes (columns) in tables must have unique names
* The order of the columns must be relevant
* The order of the rows must be relevant

## SQL

[SQL](https://youtu.be/zpnHsWOy0RY?si=OnKlyZvT133J69qN) is **S**tructured **Q**uery **L**anguage that allows retrieval and manipulation of data in tables in the relational database. SQL is a special purpose programming language and a data manipulation language. It was [developed](https://aws.amazon.com/what-is/sql/#:~:text=SQL%20was%20invented%20in%20the,SQL%20relational%20database%20management%20system.) in the 1970s. SQL became the standard language for relational databases and was adopted by the American National Standards Institute (ANSI) in 1986 and the International Organization for Standardization (ISO) in 1987. Since then, it has evolved with extensions and variations used by major database systems like MySQL, Oracle, SQL Server, and PostgreSQL, remaining the dominant language for managing structured data.

## SQLite

[SQlite](https://www.sqlitetutorial.net/what-is-sqlite/) is a lightweight, easy to use database engine that stores data in a single file on your computer. It’s called “lite” because it's small, fast, and doesn't require a separate server to run. It is disk-based and allows everything to happen right on your program. It works well with Python. You can [create](https://comp.mga.edu/learning/python/content/16#:~:text=5.%20Connect%20to%20Database) and use databases with the built-in ‘sqlite3’ module.

### **Importing sqlite**

#import sqlite

import sqlite3

#connect with database

db = sqlite3.connect["cars\_database.db"]

### **SQLite and Python Types**

[SQLite](https://comp.mga.edu/learning/python/content/16#:~:text=3.-,SQLite,-SQLite%20is%20a) supports the following types: [NULL](https://youtu.be/RKUYYrmv6gw?si=yIOQG-JthBm3kmrj), INTEGER, REAL, TEXT, BLOB

This means that you can use Python types that match the SQL types and it will automatically translate it for you. For example, if you insert a Python int, SQLite will store it as an INTEGER. A Python float becomes a REAL, a str becomes TEXT, and bytes are stored as BLOBs. This automatic type conversion makes it easier to work with databases in Python without worrying about manually converting data types.

## SQL Statements

SQL statements are used to interact with a database by performing tasks like retrieving, adding, updating, or deleting data. Each statement follows a specific structure and tells the database exactly what action to take and where to apply it.

### **Create Database**

The CREATE Database statement in SQL is used to create a new SQL database within your SQL server. This is often the first step when setting up a project that needs to store data..

**Syntax**

CREATE DATABASE database\_name;

**Create a database and name it testdb**

CREATE database testdb;

### **Drop Database**

The DROP DATABASE statement in SQL is used to permanently delete an existing database and all of its contents. This action cannot be undone, so use it with caution.

**Syntax**

DROP DATABASE database\_name;

**Delete a database named testdb:**

DROP DATABASE testdb;

### **SQLite Commands**

Sqlite commands are similar to SQL commands. There are 3 different types.

1. Data Definition Language: There are three commands in this group. These commands define and modify the structure of database objects like tables.
   1. CREATE: This command is used to create tables, a view of a table, or other object in the database
   2. ALTER: This command is used to modify an existing database object, like a table.
   3. DROP: This command is used to delete an entire table, view of a table, or object in the database.
2. Data Manipulation Language: These commands are used to manage data inside tables. There are three commands.
   1. [INSERT](https://www.sqlitetutorial.net/sqlite-insert/): This command is used to create a record.
   2. [UPDATE](https://www.sqlitetutorial.net/sqlite-update/): This command is used to modify the records.
   3. [DELETE](https://www.sqlitetutorial.net/sqlite-delete/): This command is used to delete records.
3. Data Query Language:
   1. [SELECT](https://www.w3schools.com/sql/sql_select.asp): This command is used to retrieve certain records from one or more tables. The SELECT statement retrieves a group of records. You can retrieve all records or a subset of the records with a [WHERE clause](https://youtu.be/L9GBpVXq-zc?si=S8s-5APAdlCyPe2-).

**Create a Table**

When creating a table, there are a few things you need to do. You will need to identify data types for attributes. Also, identify columns that can and cannot be null. Identify columns that must be unique. Then identify primary key and foreign keys. You should also determine default values for certain columns, and define any constraints to enforce rules on the data.

**Insert Into statement**

INSERT INTO in SQL is used to insert new records in a table. We can write an [INSERT INTO](https://youtu.be/Dndsh1dcx-U?si=eohRZwiAyM7eLQg0) statement two ways. One way specifies the column names and values explicitly, which is more reliable when the table structure might change. The second way omits the column names and requires you to provide values for all columns in the correct order. This command helps populate a table with data for queries and updates later on.

Here’s an example of that first method, which will be more flexible and safer if your table structure were to change later:

**Syntax**

INSERT INTO table\_name (column1, column2, column3) VALUES (value1, value2, value3);

**Example**

INSERT INTO students (name, age, major) VALUES (‘Ryan’, 21, 'Information Technology');

**Delete Row Statement**

A [delete](https://www.tutorialspoint.com/sqlite/sqlite_python.htm) row statement will delete a row in a table based on selection criteria. If no criteria is specified, all rows in the table will be deleted. Always use a WHERE clause with [DELETE](https://youtu.be/s7ehCYyEmVw?si=3r4YLcVMCJgB5WSB) to avoid removing every row in the table.

**Syntax**

DELETE FROM table\_name WHERE condition;

**Example**

DELETE FROM students WHERE name = ‘Ryan’;

**Example in SQLite:**

c.execute("DELETE FROM Student WHERE email='ted@umich.edu'")

**Select Statement**

The [SELECT](https://youtu.be/af4LckivJT8?si=HhbrPYxscc05gXEO) statement is used to query a database and retrieve data from one or more tables. It allows you to specify which columns and rows you want to retrieve, apply conditions, and even sort or group the data.

**Syntax**

SELECT column1, column2 FROM table\_name WHERE condition;

**Example 1 - Selecting Specific Columns**

SELECT name, age FROM students WHERE age > 18;

This selects the name and age columns from the students table where the age is greater than 18.

**Example 2 - selecting All Columns**

SELECT \* FROM students;

The \* symbol selects all columns from the students table.

**Example in SQLite**

**rows=c.execute("SELECT \* FROM Student").fetchall()**

**print(rows)**

There are many ways to use the SELECT Statement but these are the most common.

**Order By Clause**

You can add an [ORDER BY clause](https://youtu.be/uUO-ynRJKo0?si=FaRvcygw3aAhh3ll) to SELECT statements to get the results sorted in ascending or descending order.

To sort in ascending order:

SELECT \* FROM Student ORDER BY email

To sort in descending order:

SELECT \* FROM Student ORDER BY name DESC

**Examples in SQLite**

**rows=c.execute("SELECT \* FROM Student ORDER BY email").fetchall()**

**print(rows)**

rows=c.execute("SELECT \* FROM Student ORDER BY name DESC").fetchall()

print(rows)

**Update Statement**

The [UPDATE](https://youtu.be/i5aGjhTluxo?si=c8plBl2jbPycjsNa) statement in SQL is used to modify existing records in a table. You can update one or more columns for one or more rows, depending on the condition you specify.

**Syntax**

UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;

**Example 1 - Update a Single Column**

UPDATE students SET age = 25 WHERE name = 'Anthony';

**Example 2 - Update Multiple Columns**

UPDATE students SET age = 21, grade = 'A' WHERE id = 5;

**Example in SQLite**

rows=c.execute("SELECT \* FROM Student").fetchall()

print(rows)

## Alter Table

The [ALTER TABLE](https://www.w3schools.com/sql/sql_alter.asp) statement is used in SQL to make changes to the structure of an existing table — such as adding, deleting, or modifying columns.

**Add New Column**

**Syntax**

ALTER TABLE table\_name ADD column\_name DATATYPE;

**Example**

ALTER TABLE Students ADD email TEXT;

**Modify/Alter Existing Column**

**Alter**

ALTER TABLE table\_name ADD COLUMN column\_name DATATYPE;

**Modify**

ALTER TABLE Students MODIFY Age TEXT;

In this example, we have already assigned the date of birth with the datatype date. Now we changed the datatype from date to year.

**Drop Column**

**Syntax**

ALTER TABLE table\_name DROP COLUMN column\_name;

**Example**

ALTER TABLE Students DROP COLUMN dob;

## Multiple Table Query

The actual power of relational databases comes when we have more than one table and we can exploit the relationships between the tables. The SQL JOIN helps in retrieving data from two or more database tables. The tables are mutually related using primary keys and foreign keys. Using JOINs, we can combine data across tables in a meaningful way without duplicating information. This makes data retrieval more efficient, maintains data integrity, and supports more complex queries and reports.

### **Normalization**

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves breaking up large tables into smaller ones and linking them using relationships like primary and [foreign keys](https://youtu.be/GfuH_8uH16k). This makes the database more efficient, easier to update, and less prone to errors. Well-structured relations contain minimal data redundancy and allow users to insert, delete, and update rows without causing data inconsistencies. Before normalization, all data might be stored in a single table. For example, if Sarah is enrolled in both Math and History, her name would appear twice in the table—once for each course. This repetition leads to unnecessary redundancy. After normalization, the data is split into separate tables, such as a “Students” table that stores each student’s ID and name only once, and an “Enrollments” table that links students to the courses they’re taking using their student ID. This makes the database more efficient, organized, and easier to maintain.

### **Relational Keys**

Relational keys are essential in organizing and connecting tables within a relational database. A primary key is a unique identifier for each record in a table, ensuring that no two rows have the same value in that column. A foreign key is a field in one table that refers to the primary key in another table, creating a link between the two tables. This relationship allows data to be organized efficiently without redundancy.

For example, imagine you have a ‘Students’ table where each student has a unique ‘StudentID’ as the primary key. You also have a ‘Grades’ table that records each student’s test scores. Instead of repeating the student's name in the ‘Grades’ table, you use ‘StudentID’ as a foreign key. This way, each grade entry in the ‘Grades’ table is linked back to the correct student in the ‘Students’ table using that ID. This setup helps maintain consistency, reduces duplication, and allows easy updates or deletions without affecting related data incorrectly.

### **Joining Multiple Tables**

Joining multiple tables allows you to combine data from two or more related tables using a shared column, typically a primary key and a corresponding foreign key. This is useful when data is split across different tables to reduce redundancy and improve organization. By removing the replicated data and replacing it with references to a single copy of each bit of data we build a “web” of information that the relational database can read through quickly.

For instance, consider a bookstore database with one table called Books that stores book details (like BookID, Title, and Price), and another table called Authors that stores author information (AuthorID, Name). A third table, BookAuthors, links books to authors using foreign keys: BookID and AuthorID. To get a list of books along with their authors' names, you’d join all three tables. By using an SQL JOIN, you can match rows from the Books table to the correct rows in the Authors table through the BookAuthors link. This lets you see combined information (like book title and author name) in a single query, even though the data is stored across separate tables.

### **SQLite Joins**

In SQLITE, the [JOIN clause](https://www.sqlitetutorial.net/sqlite-join/) is used to combine records from two or more tables in a database. It unites fields from two tables by using the common values of both tables. There are three main types of JOINs in SQLite: INNER JOIN, OUTER JOIN, and CROSS JOIN.

An INNER JOIN returns only the rows where there is a match in both tables. For example, if you have a Customers table and an Orders table, an inner join on CustomerID will return only customers who have placed orders. The syntax looks like:

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

An OUTER JOIN returns all records from one table and the matched records from the second table. If there’s no match, it still returns the row from the first table with NULL for the second. SQLite supports LEFT OUTER JOIN (commonly just written as LEFT JOIN). The example below will return all customers, even if they haven't placed an order yet.

SELECT Customers.Name, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

rows=c.execute("SELECT Author, title,Year FROM classics INNER JOIN customers ON

A CROSS JOIN in SQL combines every row from one table with every row from another table. It doesn't require any condition to match the rows—so if one table has 3 rows and the other has 4, the result will have 12 rows. It’s like mixing all possible combinations. For example, if you have a Shirts table with sizes and a Colors table with colors, a cross join would show every size with every color. This can be useful for generating test data or listing all combinations of two sets.

Example of CROSS JOIN in SQLite

rows=c.execute("SELECT \* FROM classics CROSS JOIN customers")

## [Reference and Additional Materials](#_Reference_and_Additional)

<https://www.pingcap.com/article/sql-data-structures-explained/>

<https://www.ibm.com/history/relational-database>

[www.tutorialspoint.com](http://www.tutorialspoint.com)

<https://www.datamation.com>

<https://www.sfu.ca/fas/computing.html>