



Methods in Open Science

Structured Query Language

Image from NASA:
https://commons.wikimedia.org/wiki/File:Earth_Western_Hemisphere_transparent_background.png#filelinks



AmericaView™
Empowering Earth Observation Education
AmericaView.org - Est. 2003

This module will introduce Structured Query Language (SQL), which is used to work with and manage databases along with making queries to databases. I will primarily focus on how SQL is used to query, subset, and summarize data in the context of data science.

SQL Overview



What can you do with SQL?



- ❖ Structured Query Language
- ❖ Access, manipulate, and query databases
- ❖ ANSI/ISO standard
- ❖ Different versions of SQL are available



<https://www.w3schools.com/sql/default.asp>

3

SQL stands for Structured Query Language. It is the primary language for working with relational databases and querying data.

Generally, SQL allows you to access, manipulate, and query data stored in a relational database.

SQL syntax can vary between different relational database systems. SQL is an ANSI/ISO standard, and, in order to be compliant, a specific version of SQL must adhere to some common rules and include the same major commands. In this course, we will focus on SQL as implemented with SQLite, which is open-source, free, and ANSI/ISO compliant.

w3schools.com, which is referenced on this slide, is a great resources for learning and working with SQL. It was used extensively in the production of this module.



Relational Database



- ❖ Stores data as **tables**
- ❖ Tables can be related to one another
- ❖ **Indexing** used to speed up query and data access
- ❖ Examples: MS SQL Server, IBM, DB2, MySQL, Microsoft Access, SQLite, PostgreSQL

4

A relational database is a data structure in which multiple datasets, represented as tables, can be stored collectively and related to one another.

In order to speed up data query and access, data can be indexed.

There are a variety of relational database management systems.



- ❖ Database contains tables that are named
- ❖ Rows in tables are called Records
- ❖ Data are stored in Columns as Attributes
- ❖ SQL keywords are not case sensitive
- ❖ Standard is to use UPPER CASE
- ❖ Include ; at end of statement

As with all computer languages, SQL has some standards and best practices.

When discussing databases, files stored in the database are commonly called tables. Rows are called records and columns are called columns or attributes.

For example, a table of student information could have one row or record for each student with each column holding information or a specific attribute about that student, such as academic year, hometown, major, or GPA.

When typing SQL, keywords are not case sensitive. For example, SELECT, Select, and select are all acceptable. However, it is considered best practice to use upper case for all keywords (e.g., SELECT).

It is also recommended to end each SQL statement with a semicolon.

Query Basics

6

In this section we will discuss querying data from tables stored in a database.



SELECT



- ❖ **SELECT** records from a table
- ❖ **FROM** used to specify table
- ❖ ***** indicates to select all columns/attributes
- ❖ Can also select subset of columns/attributes

```
SELECT * FROM county_summary;
```

```
SELECT NAME, GEOID, elev_median FROM county_summary;
```

7

A query is initiated with the **SELECT** keyword, and **FROM** is used to specify what table you want to query.

In the first example, I am selecting all columns from the “county_summary” table. The asterisks indicate to select all columns.

If you want to select just a subset of the columns, you can list the column names, separated by commas.

Note that both queries end with a semicolon.



WHERE



❖ Used to subset data based on a condition

```
SELECT NAME, elev_median  
FROM county_summary  
WHERE elev_median > 1200;
```

8

Using SELECT and FROM will allow you to select all records in a table and all or a subset of columns.

WHERE is used to define a subset of the records. Data that meet the condition defined by the WHERE clause will be selected, and all other records will not be selected.

Here, I am selecting all counties that have a median elevation larger than 1200 meters from the “county_summary” table. The query will only return the “NAME” and “elev_median” attributes for the records that satisfy the WHERE clause.

With the addition of WHERE, you can now query or subset your data/records based on criteria.



LIMIT



- ❖ Some DBs use **SELECT TOP**
- ❖ **LIMIT** number of records returned

```
SELECT NAME, elev_median  
FROM county_summary  
WHERE elev_median > 1200  
LIMIT 10;
```

LIMIT is used to limit the number of records returned. In this example, only the first 10 records that meet the condition are returned.



❖ Tables and columns can be assigned alias names using **AS**

```
SELECT NAME AS County, elev_median AS Elevation
FROM county_summary
WHERE elev_median > 1200;
```

10

You may want to define a different name for a column or table for use in the query and output.

The **AS** keyword can be used to define an alias name. In this case, the “NAME” field is assigned the alias “County”, and “elev_median” is assigned the alias “Elevation”. These will be the new column headings in the query result.



Operators



Comparison Operator	Description
=	Equal
>	Greater Than
<	Less Than
>=	Greater Than Or Equal To
<=	Less Than or Equal To
<>	Not Equal To

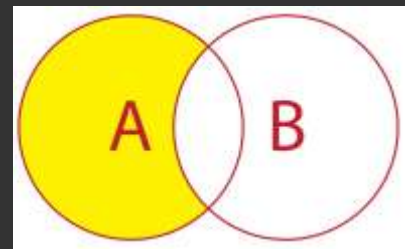
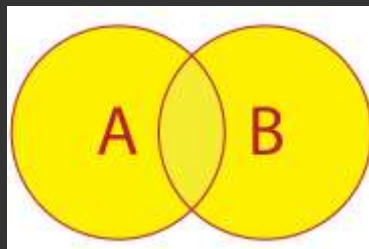
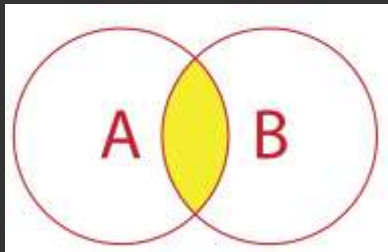
These are the comparison operators available for use in a WHERE clause.



AND, OR, NOT



```
SELECT NAME AS County, elev_median AS Elevation
FROM county_summary
WHERE elev_median > 1200 OR elev_median < 300;
```



Logical Operator	Description
AND	A AND B
OR	A OR B
NOT	A NOT B

12

In order to query based on multiple conditions or criteria, you can use logical operators.

AND means that both criteria must be satisfied.

OR means at least one of the two criteria must be satisfied (including both criteria).

NOT indicates that one criteria but not the other must be satisfied.

In the example, only records that have an elevation greater than 1200 meters or less than 300 meters will be selected. If OR was replaced with AND, no results would be returned because the median elevation cannot be both greater than 1200 and less than 300.



BETWEEN



❖ Return records **BETWEEN** two values in a column

```
SELECT NAME AS County, elev_median AS Elevation
FROM county_summary
WHERE elev_median BETWEEN 1000 AND 2000;
```

13

BETWEEN can be used in a WHERE clause to denote values between two end members.



IN



- ❖ Return records that match an element **IN** a list
- ❖ Can use **NOT IN** to reverse the selection

```
SELECT * FROM geology  
WHERE MAJOR1 IN ("Limestone", "Dolostone");
```

```
SELECT * FROM geology  
WHERE MAJOR1 NOT IN ("Limestone", "Dolostone");
```

14

IN allows you to select based on whether a value for an attribute matches one of the records in a list. In the example, a record will be returned if its rock type attribute ("MAJOR1" column) is either "limestone" or "dolostone".

NOT IN is used to invert IN; records with a value for a specific attribute or column not in the list will be returned.



LIKE



- ❖ Search for a pattern
- ❖ % = 0, 1, or multiple characters
- ❖ _ = a single character
- ❖ \ is escape character
- ❖ NOT LIKE is also available

```
SELECT * FROM geology  
WHERE MAJOR1 LIKE '%stone';
```

15

LIKE allows you to find records based on a pattern.

In the example, any unit with a rock type ending with “stone” will be returned.

NOT LIKE can be used to invert LIKE. So, rock types not ending in “stone” will be returned.

% before “stone” indicates that this is the end of the string. In contrast, ‘stone%’ would indicate the beginning of the string and ‘%stone%’ would indicate at the beginning, middle, or end. _ is used to denote a single character.

\ can be used as an escape character to avoid characters that have special meaning in SQL being misinterpreted, such as quotes.



MIN, MAX, COUNT, AVG, SUM



- ❖ Return summary or statistical measure for records in rows that meet query criteria

```
SELECT NAME as County, AVG(elev_median) AS Elevation  
From county_summary;
```

```
SELECT NAME as County, MAX(elev_median) AS Elevation  
From county_summary
```

```
SELECT COUNT(NAME) AS County  
From county_summary  
WHERE temp_median > 20;
```

16

Numeric data can be summarized using different summary methods.

In the first example, the average of all median county elevations will be returned.

In the second example, the maximum of all median county elevations will be returned.

In the last example, the number of counties with a median mean annual temperature greater than 20 degrees Celsius will be returned.

You will see more examples of this when we investigate grouping.



ORDER BY



- ❖ Sort results **ascending** or **descending** based on strings or numbers

- ❖ **ASC** = Ascending

- ❖ **DESC** = Descending

- ❖ Can use multiple columns/attributes

```
SELECT NAME AS Counties, elev_median AS  
Elevation, temp_median AS Temp  
From county_summary  
ORDER BY Elevation DESC, Temp ASC;
```

17

ORDER BY can be used to sort the results in ascending (ASC) or descending (DESC) order relative to a column.

When used on character or text data, this will sort alphabetically. When used on numeric data, it will sort increasing/decreasing.

It is also possible to order using multiple fields. In the example, the results are ordered based on decreasing elevation and increasing temperature.



CASE



- ❖ Return different results to new column based on a condition
- ❖ Like **If...Else** statement

```
SELECT NAME AS Counties, elev_median AS Elevation,  
       CASE  
         WHEN county_summary.elev_median < 1000 THEN 'Low Elevation'  
         WHEN county_summary.elev_median >= 1000 AND  
           county_summary.elev_median < 2000 THEN 'Medium Elevation'  
         ELSE 'High Elevation' END  
       AS ElevationClass  
FROM county_summary  
ORDER BY Elevation DESC;
```

18

CASE is used to return different results based on a case and are similar to If...Else statements used in most programming languages, such as Python or R.

Here, CASE is being used to create a new column in which to return the query results. The string stored in the query will vary based on a condition.

CASE initiates the clause. WHEN is used to define a specific case while THEN is used to denote what to return if that specific case is true. ELSE denotes the default case and does not require a condition. END is used to end the CASE clause, and AS is used to name the output column.

In the example, CASE is used to return different text depending on the elevation of the county. The result is written to a column called “ElevationClass”.



NULL VALUES



- ❖ **NULL** = missing data (not the same as zero)
- ❖ **IS NULL** = use in **WHERE** statement to find **NULL** records
- ❖ **IS NOT NULL** = use in **WHERE** statement to find not **NULL** records

```
SELECT NAME AS County
From county_summary
WHERE temp_median IS NOT NULL;
```

19

NULL indicates missing data. IS NULL can be used in a WHERE clause to find records with missing data in a specific column while IS NOT NULL is used to find records without missing data in a column.



Summary of Key Points: Query Basics



- ❖ A variety of methods are available to query data tables using SQL.
- ❖ You can select subsets of columns with the SELECT clause and rename the columns in the output.
- ❖ You can use a WHERE clause to subset records or rows based on a criteria.
- ❖ It is possible to reorder the records in the output.
- ❖ Summary statistics can be calculated as part of the output.

Grouping



GROUP BY



- ❖ Group data based on a **nominal** or **ordinal** variables
- ❖ Use summary measures to obtain group statistics: **COUNT**, **MIN**, **MAX**, **AVG**, **SUM**, etc.

```
SELECT UNIT_AGE AS Age,  
COUNT(UNIT_NAME) AS Unit_Count  
FROM carbonates  
GROUP BY Age  
ORDER BY Unit_Count DESC;
```

22

You may want to group the results based on a nominal or ordinal variable or attribute. This can be accomplished using GROUP BY. In the example, the result is being grouped based on the geologic age, which is an ordinal variable.

So, each age will be returned to a column called “Age”. The number of units in the age will be returned to a column called “Unit_Count”, and the results will be sorted in descending order based on the number of units in the age group.

It is common to use summary keywords with GROUP BY. Here, I am using COUNT.



HAVING



- ❖ Replaces **WHERE** when data are grouped
- ❖ Cannot use **WHERE** with **GROUP BY**

```
SELECT UNIT_AGE AS Age,  
COUNT(UNIT_NAME) AS Unit_Count  
FROM carbonates  
GROUP BY Age  
HAVING Unit_COUNT > 1500  
ORDER BY Unit_Count DESC;
```

23

WHERE cannot be used once data are grouped. So, HAVING serves a similar purpose for grouped data; it allows you to subset the groups based on a query.

So, in this example, only geologic ages with more than 1,500 carbonate records will be returned.



Summary of Key Points: Grouping



- ❖ It is possible to calculate summary statistics by group using the GROUP BY clause.
- ❖ WHERE cannot be used on grouped data. Instead, HAVING is used.

Joins



Joins



- ❖ Allow you to **relate** tables in a **database**
- ❖ Relationship established using a common field/attribute or unique identifier
- ❖ **Primary Key** = unique identifier in table
- ❖ **Foreign Key** = unique identifier in table being associated

26

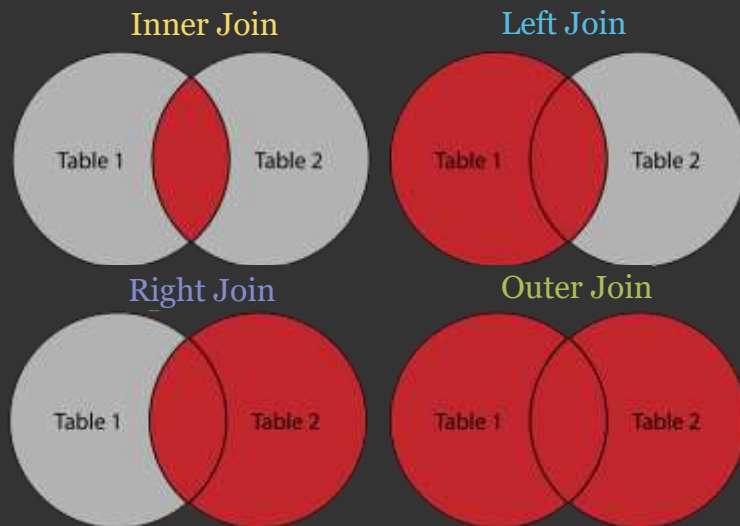
Joins allow you to relate or associate two tables in a database based on a shared or common field.

In the case of a table join, a unique ID in one table, the primary key, is used to associate the data to another table based on the shared ID, which is called the foreign key in the second table.

So, table joins require that a relationship be established based on shared IDs or a common column.



Join Types



27

Different types of joins are possible.

An inner join will return only records that occur in both tables or only records where a match is found between the primary and foreign keys.

A left join will return all records from the first table even if no match is found in the second table.

A right join will return all records in the second table even if there is no match in the first.

Lastly, a full join will return all records from both tables, regardless of whether there are matches.

I find that I use inner and left joins most often.



Join Example



```
SELECT cnty.NAME AS CountyName, st.NAME AS StateName
FROM county_summary AS cnty
INNER JOIN state_fips AS st
ON cnty.STATEFP = st.FIPS;
```

28

In this example, state names are being associated with each county based on the state FIPS (Federal Information Processing Standard) code, which is unique for each county. The FIPS code occurs in both datasets. So, it is used to relate the data.

So, a table will be returned that includes the county name and state name. To simplify the syntax, I am using alias names for both tables. An inner join is used and is specified using the INNER JOIN keywords. The ON keyword is used to define the primary/foreign key relationship.



Join Example



```
SELECT st.NAME AS StateName, AVG(cnty.elev_median)
AS Elevation
FROM county_summary AS cnty
INNER JOIN state_fips AS st
ON cnty.STATEFP = st.FIPS
GROUP BY StateName
HAVING Elevation > 1000;
```

29

In this example, I am grouping using the state name to return an average county-level median elevation by state. So, a join field is used to define the subsequent grouping. Also, I am only returning states that have an average median county elevation greater than 1,000 meters. Note the use of HAVING as opposed to WHERE since the data have been grouped.



Summary of Key Points: Joins



- ❖ Different types of joins exist to relate tables.
- ❖ Joins make use of shared or common attributes.

Databases Manipulation



❖ **CREATE DATABASE** <https://www.postgresql.org/docs/12/sql-createdatabase.html>

❖ **DROP DATABASE** <https://www.postgresql.org/docs/12/sql-dropdatabase.html>



<https://www.postgresql.org/docs/>

We will not focus on database, table, and record manipulation in this module. Instead, we focus on data queries, subsetting, and summarization. However, I did want to mention some of the techniques that are available to manipulate databases and their content.

CREATE DATABASE and DROP DATABASE can be used to create and delete databases, respectively.

When a database is created, some parameters can be defined. For example, you can define the database owner, a template to use, and a character set encoding.

If you use SQL to manipulate tables, it is important that you use caution. You don't want to accidentally delete records, tables, or entire databases, for example.



❖ **CREATE TABLE** <https://www.postgresql.org/docs/12/sql-createtable.html>

❖ **DROP TABLE** <https://www.postgresql.org/docs/12/sql-droptable.html>

❖ **ALTER TABLE** <https://www.postgresql.org/docs/12/sql-altertable.html>



<https://www.postgresql.org/docs/>

This slide shows examples for manipulating tables.

CREATE TABLE allows you to create a new table to store in an existing database while DROP TABLE can be used to delete a table from a DATABASE.

ALTER TABLE allows you to change the definition of the table once it is created. For example, you can add or delete columns, change data types for columns, set or change default column values, or change the name of the table or the names assigned to specific columns in the table.



Manipulate Records



❖ **INSERT** <https://www.postgresql.org/docs/12/sql-insert.html>

❖ **UPDATE** <https://www.postgresql.org/docs/12/sql-update.html>

❖ **DELETE** <https://www.postgresql.org/docs/12/sql-delete.html>



<https://www.postgresql.org/docs/>

34

It is also possible to manipulate individual records in a table using INSERT, UPDATE, and DELETE.

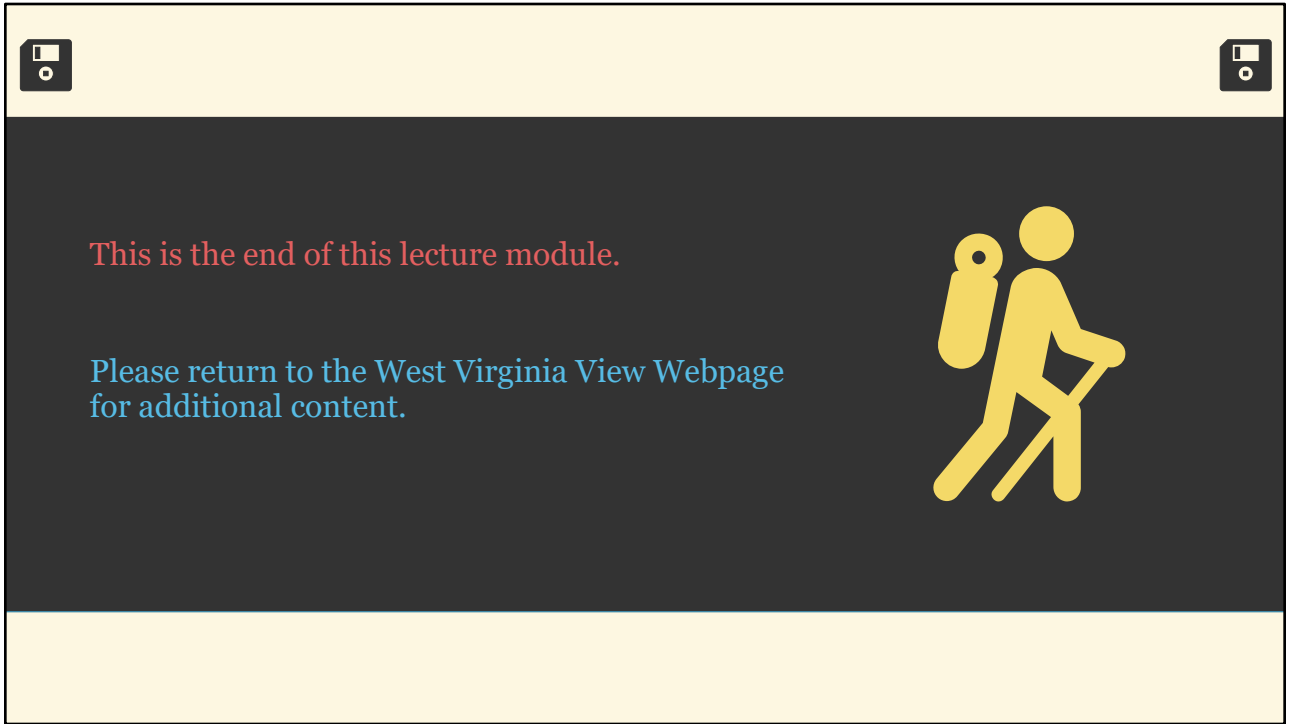
INSERT allows you to add records while DELETE will remove them. UPDATE can be used to edit an existing record, such as change the values in certain columns for that record.



Summary of Key Points



- ❖ Some uses of SQL:
 - ❖ Manipulate databases, including creating and deleting databases
 - ❖ Creating, deleting, or altering tables in a database
 - ❖ Inserting, updating, and deleting records within a specific table in a database
 - ❖ And more!



Thanks! Hope you found this useful.