09/07/24 Course # 6 Databases & SQL for Data Science with Python

Chaper (I): Getting Started with SQL

Module # I. (A) Course Introduction

- SQL helps to :-
  a. Query data
  b. Update data
  c. Manipulate data

- Overview :
  Module 1 - basic SQL & databases
  Module 2 - relational databases, creating table & modifying them
  Module 3 - intermediate SQL → functions, string pattern, grouping & sorting
  Module 4 - SQL & Jupyter Notebooks
  Module 5 - final assignment
  Module 6 - Advance SQL for data engineer

Chapter (II): Basic SQL

~~Module # II~~ Module # 1. (II) (A): Introduction

Database :
- a repository of data
- provides functionality for adding, modifying & querying data
- different kinds of database store data in different forms.

Page No.
Date

Relational DB - data stored in tabular form.
- relations can be formed b/t tables.

DBMS - Data Base Mangement System
↳ A software to manage databases & control data - by access, organization and storage.
examples of RDBMS — My SQL, Oracle Database, IBM Db2.

→ 5 basic operation of SQL commands -

1. Create a table
2. Insert
3. select
4. update
5. delete

~~Module #1 # II. (B) SELECT statement~~

Module # 1. II. (B) SELECT statement

- Retrieve data from a relational database table
- Define use of a predicate
- Identify the syntax of select statements using WHERE clause
- List comparison operators supported by RDBMS.

• SELECT statement → A Data Manipulation Language (DML) statement to read & modify data.

• WHERE clause :-
  - Restricts the result set
  - Always require predicate
  - it can make use of comparison operators

Module # 1. II. (C) COUNT, DISTINCT, LIMIT

• COUNT:
  - retreives number of rows from column, which satisfy the given condition.

syntax → select COUNT (col-name) from table-name ...
... where condition

• DISTINCT
  - retrevies distinct / unique values from the specified column.

syntax :-
select DISTINCT col-name from table-name ...
... where condition

• LIMIT

- restrict number of rows retrived from database.

syntax :

```
select * from table-name LIMIT no.
```

- it can be used with OFFSET.

→ OFFSET is used offset the select, rather than starting from row 0, the number of rows will be skipped & selection will start from next row.

eg:

```
select * from table-name LIMIT 25 OFFSET 15
```

→ selection will start from row no. 16

~~No~~

~~Module # 1. II. (D): INSERT Statement~~

Module # 1. II. (D): INSERT Statement

• INSERT statement

- It's a Data Manipulation Language (DML) statement.
- INSERT is used to populate the statement

syntax -

```
INSERT INTO table-name
(col-names, ...)
VALUES (value, ..., ...);
```

- multiple can be inserted from a single INSERT statement.

eg: INSERT INTO table_name
(col_name1, col_name2, ...)
VALUES
(R1a, R1b, R1c, R1d....),
(R2a, R2b, R2c, R2d,...);

Module # I · II · 8(E) : UPDATE and DELETE

goal: Understand importance of where clause in both UPDATE & DELETE statement.

- UPDATE
  - alter existing table data
  - It's a Data Manipulation Language (DML) statements.

syntax :-

UPDATE table_name SET [col_name = (value)] WHERE condition

- If WHERE clause is not used then all rows of the specified column will be affected/altered.

• DELETE statement

- It's a Data Manipulation Language statement.

syntax:

```
DELETE from table-name
    WHERE condition;
```

→ if WHERE clause is not used, all rows will be deleted.

Module #2: Relational Databases

Chapte (I): Introductions to Relational Databases & Tables

Module # 2 · I · (A) Relational Database Concepts

goal · adv. of relational db, entity name, attribute mapping, diff. b/t an entity & attribute, identify some commonly used data types, describe function of primary keys.

• Relational Model

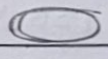
- Allows for data independence
  ↓
  this means, it provides:-
  a. logical data independence
  b. Physical data independence
  c. Physical storage independence

- Entity Relationship (ER) data model is an alternative to Relational data model.

- Entity Relationship Diagram (ERD) represents entities (tables) and their relationships.

- Entity Relationship Model is utilized as a tool to design relational databases.

  - It represents the idea of a database as a collection of entities. Entities are objects that exist independently of other entities in the database.

- Basic Components of ERD are :-
  (i) Entities ——— symbol = 
  (ii) Attributes ——— symbol = 

- Entities has data elements called attributes
- Attributes are connected to exactly one entity.

- Entity- Attribute Mapping

- Entity becomes table
- Attribute gets translated into columns
  ↳ each attribute stores data values of different formats eg char, numbers, date, etc.

• Primary keys & foreign keys

- Each table is assigned a primary key.
- primary key uniquely identifies a tuple or row.
  - preventing duplication of data.
  - providing way to define relationship between tables.

Module # 2 · I · (B) : Types of SQL statements

goal : Differentiate b/t Data Definition Languages statements & Data Manipulation Language statements

• SQL statement types :

A. Data Definition Lang" (DDL)

— define, change or drop data

eg

- CREATE : create table
- ALTER : alter table
- TRUNCATE : delete table data
- DROP : delete table

B. Data Manipulation Lang". (DML)

— Read or modify data

- CRUD - create, read, update & delete rows

eg

INSERT - insert data/rows in table

SELECT - read row/rows from table

UPDATE - update rows

DELETE - deletes rows

- Difference between TRUNCATE & DELETE

TRUNCATE	DELETE
- removes all rows from table, but keeps table structure intact	removes specific rows from table based on a condition
- faster than delete, as it does not generate transaction log & does not log individual row deletion.	Slower than TRUNCATE, as generates transaction log for each row deletion.
- Cannot be rolled back	can be rolled back
- Resets the identity column to its initial seed values	does not reset identity column value

- Identity column value — its the value assigned to the identity column for specific row. Its automatically generated by the database & ensure each row has an unique identifier.
eg: primary keys, or column with employee ID.

~~Module # 2.I~~

Module # 2.I.(C) CREATE TABLE Statement

goal: use entity name and attributes to create table

syntax: CREATE TABLE table_name
(
column_name1 datatype optional_parameters,
⋮
column_nameN datatype
);

Module # 2.I.(D) ALTER, DROP & TRUNCATE tables

- ALTER TABLE
  - add ... or remove columns
  - modify data type of columns
  - add or remove keys
  - add or remove constrains

→ change / alter structure of existing table.

syntax:

```
ALTER TABLE table_name
    ADD COLUMN column_name datatype;

ALTER TABLE table_name
    MODIFY column_name datatype;

ALTER TABLE table_name
    DROP COLUMN column_name;
```

- DROP TABLE : deletes existing table.

syntax: DROP TABLE table-name;

→ deletes table as well as data

- TRUNCATE

syntax: TRUNCATE TABLE table-name
IMMEDIATE;

→ deletes all rows (data) from table but preserves the table structure.

Module # 2. II. (E) Hands on-Lab.

done

Chapter 3: IBM DB2

Module # 2. III. (A) Creating Database Instance on IBM cloud.

goal: Cloud database basic, List some cloud database, describe database instance, create an instance of db2, on cloud.

- A cloud database is a database which is hosted on the cloud.

• Advantages of Cloud :-

1. Ease of use & access :-
   can be accessed through API, Web Interface & cloud or remote app

2. Scalability & Economics
   - expand / shrink storage & compute resource
   - pay per use

3. Disaster recovery
   - cloud backups and geographical distribution

• Examples of Cloud databases :-
   - IBM Db2
   - Database for PostgreSQL
   - Oracle Database Cloud service
   - Microsoft Azure SQL Database
   - Amazon Relational Database Service

• Database Service Instances
   - DbaaS - Database as a Service
   - provides users with DB databases on cloud, without setting up installions.

09/12

Module 3: Intermediate SQL

Chapter I: Refining Results

Module # 3.I.(A) Using String Pattern and Ranges.

goal: String Patterns, Ranges, Sets of values

- String Patterns

syntax :

```
select col_name from table_name
    where col_name like 'R%';
```

var-char characters

example of where clause with like predicate

used to search for a pattern in a column

- Ranges → usually used for numerical values.

syntax

```
select col_name1, col_name2 from table
    where col_name2 >= value1 and col_name2 <= value2;
```

OR

```
select col_name1 from table
    where col_name1 between value1 and value2;
```

→ the value1 & value2 are inclusive in the range.

• Set of values

syntax: select * from table_name
where col_name in ('value A', 'value B' .. 'value n');

→ we can specify a set of values in where clause.

→ Helps to keep the query brief / concise.

• Conclusion: These help to simplify a select statement.

Module # 3.I.(B) Sorting Result sets

goal: sort result set, explain how to indicate which column to use.

• ORDER BY clause

→ sort the result set according to specified column.

syntax: select col_name from table
order by req_col_name;
→ sorts in ascending order

select col_name from table_name
order by req_col_name desc;
sorts in descending order

→ specifying column sequence number.

select col-name1, col-name2 ... col-namen from table
order by 2 ;
↳ results are sorted acc to the 2nd column mention in select statement

Module # 3.I.(c) Grouping Result Sets

goal: eliminate duplicates from result set & restrict result set.

- DISTINCT
- helps to return only unique values.

- GROUP BY

eg: select country, count(country) from author
group by country ;

→ we can also assign name to a column of the result set :-

select country, count(country) as count from author
group by country;

- ~~Having~~

- HAVING clause

→ having is used with group by to further restrict the result set.

eg: select country, count(country) as count from author
group by country having count(country) > 4;

09/13 **Chapter (II) : functions, Multiple tables & Sub-queries**

Module # 3 · II · (A) Built-in Database function

- using built-in functions significantly reduces the amount of data retrieved. i.e reduces network traffic and bandwidth usage.
- Could speed up data processing

- AGGREGATE OR COLUMN FUNCTION :

→ takes collection of values as INPUT & OUTPUTS a single value

→ eg: SUM(), MIN(), MAX(), AVG(), etc

• SCALAR and STRING function

SCALAR : perform operations on every input value

example : ROUND(), LENGTH(), UCASE(), LCASE()

ROUND()	LENGTH()	UCASE()	LCASE()
round up numerical data	return length of strings in column	returns column data in uppercase	returns column data in lowercase

UCASE() & LCASE() can also be used with WHERE clause

eg :

```
SELECT * FROM table_name
WHERE LCASE(colname) = 'cat';
```

Module # 3.II.(B) Date and Time builtin function

• SQL contains these data types for date & time :-

1. date : YYYYMMDD (8 digits)
2. time : HHMMSS (6 digits)
3. timestamp : YYYYXXDD HHMMSSZZZZZZ (20 digits)
   - XX → month
   - ZZZZZZ → microseconds

• Date / Time functions -

YEAR(), MONTH(), DAY(), DAYOFMONTH(), DAYOFWEEK(), DAYOFYEAR(), WEEK(), HOUR(), MINUTE(), SECOND()

• YEAR() → return YYYY
• MONTH() → return MM
• DAY() → return DD

- Date or Time Arithmetic

DATE_ADD (col.name, INTERVAL value DAY|MONTH|YEAR)
↳ returns new dates with added interval

DATE_SUB(col.name, INTERAL value DAY|MONTH|YEAR)
↳ returns new date with subtracted interval

Module # 3. II. (c) Sub Queries & Nested Selects

→ Sub queries are used :
(1) to evaluate aggregate function.
(2) in list of columns (AKA column-expression)
eg
select EMP_ID, SALARY,
(select AVG(SALARY) from employee as 'AVG_SALARY')
from employee;

(3) in FROM clause:
(AKA Derived Tables or Table-Expression)
- these derived tables could be helpful when working with multiple tables and joins

eg select * from
(select EMP_ID, F_NAME, L_NAME, DEP_ID
from employee) AS Newtable_name;

eg

```
select emp_id, fname, l_name, salary from employee
    where salary > (select AVG(salary) from
                    employee);
```

Module # 3. II. (D) Working with Multiple Tables

- ways to access multiple tables in the same query:
  1. Sub-queries
  2. Implicit joins
  3. Join operators (inner, outer, etc)

- Implicit join → joining two table without using join method.

eg `select * from table1, table2;`

→ This results in a full join. (aka Cartesian join)

- limiting results of full join:

```
select * from employee, departments
    where employee.dept_id = department.dept_id;
```

Module 6 : Advanced SQL

(skipping 4 & 5 for now)

Chapter I : Views, Stored Procedures and Transactions.

Module # 6 . I . (A) : Views

goal: define view, when to use it & it's syntax

- View : is alternative way of representing data.
- it can include some or all columns from one or more base tables.
- only the definition of a view is stored, not the data.
- views are dynamic

- View can :-
- show a selection of data (like keep personal data or sensitive data hidden while displaying other data)
- combine two or more tables in meaningful ways
- simplify access to data (giving access to view & not of the whole database)
- show only portions of data in the table

- syntax :

```
CREATE VIEW view_name        -> can be 128 characters in length
( col_alias1, col_alias2, ... )
AS SELECT col-name1, colname2 ...
FROM table_name
WHERE condition/predicate ;
```

- views cannot use ORDER BY clause & cannot name a host variable

Module # 6.I.(B): Stored Procedures

goal: what is stored procedure, its advantages &, its usecase & definition.

- Stored Procedure:
  - act of sql stored & executed on server database.
  - can be written in many languages.
  - accept information in the forms of parameters.
  - peform CRUD operations
  - return results to client

- Benifits of Stored Procedure
  - reduction in network traffic as only one call is needed to perform multiple operations
  - improvement in performance
    - as processing takes place on server end. (where the data is stored)
  - reused of code
    - multiple applications can use same code.
  - increase in security
    - as data & table information is kept hidden from client
    - server side logic can be used to validate data before completing the request.

- **SYNTAX :-**

```
DELIMITER $$ ──→ marks begining of procedure
CREATE PROCEDURE procedure_name ( IN col-name datatype, ... )
BEGIN
" ——— " ——— " ——— } queries
" ——— " ——— " ——— "
END
$$
DELIMITER; ──→ marks end of procedure
```

- **use cases**

calls to procedure can be made from :-
- external applications
- Dynamic SQL queries

eg : CALL procedure_name ('parameter1, parameter2 ... n)

Page No.
Date

Module # 6.I.(C): ACID transactions

goal: explain ACID transaction, syntax, explain commits & rollbacks

- Transaction :-
  - an indivisible unit of work
  - consist of one or more sql statements
  - Either all happens or none.
    - as all sql statements are completed successfully, this leaves data base in stable state.
    - if none are completed, it leaves the database as its previous state

- ACID Transaction

A - Atomic : all changes must be performed successfully or not at all.

C - Consistent : Data must be in consistent state before & after transaction.

I - Isolated : No other process can change data transaction is running.

D - Durable : Changes made by transaction must persist.

- ACID Commands
  - BEGIN → start the ACID transaction, it's implicit in nature.
  - COMMIT → all statements completed successfully, save the new database state.
  - ROLLBACK → one or more statements failed, undo all changes

• Calling ACID commands

- Most languages use EXEC SQL command to execute SQL statements from code.
- ACID commands can be called from various languages. they require the database specific API or connectors
- we can also include error checking routine in our codes.

eg:

```
void main()
{
    EXEC SQL
    ____________
    ____________;

    EXEC SQL
    ____________
    ____________;

    FINISHED :
    EXEC SQL COMMIT WORK;
    return;

    SQLERR:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK;
    return;
}
```

— error checking routine

Page No.
Date

Chapter II: Join Statements

Module # 6.II.(A): Joins Overview

- Join Operator Combines rows of two or more tables.

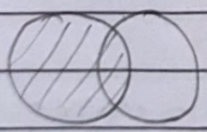
```
SELECT COL.NAME, .... FROM TABLE1 JOIN & TABLE2
  ON   CONDITION
```

- Types of Joins
  - Inner
  - Outer

Module # 6.II.(B): Inner Join

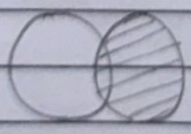
Module # 6.II.(C): Outer Joins

- Left Outer Joins



```
SELECT T1.COL1,T1.COL2, T1.COL3, T2.COL1, T2.COL2
FROM   TABLE1 T1 LEFT JOIN TABLE2 T2
ON     T1.COLN = T2.COLN;
```

- Right Outer Join



```
SELECT T1.COL1, T1.COL2, T2.COL1, T2.COL2
FROM   TABLE T1 RIGTH JOIN TABLE2 T2
ON T1.COLN = T2.COLN
```

- full Join

Module 4: Accessing Databases using Python

~~Etc~~

Chapter I: ~~###~~ Introduction

goal: python to connect to databases, create tables, load & query data, create instance in cloud & connect to a database.

describe sql, API.

Module # 4.I.(A): Introduction

- Benifits of Python
  - Python ecosystem
  - ease of use
  - portable
  - supports relational databases
  - python database api (DB-API)
  - it's detailed documentation

Page No.
Date

- Notebooks:
  - Matlab notebook
  - Jupyter notebook
  - Apache spark
  - Apache Zeppelin

- APIs used by popular SQL based DBMS systems

	API name
MySQL →	MySQLdb API
Postgre SQL →	psycopg2
IBM DB2 →	ibm_db
SQL server →	dbUb API
Database access for Microsoft Windows OS →	ODBC
Oracle →	OCI
Java →	JDBC

Module # 4. I. (B) Coding with DB-API

- Python code communicates with DBMS through DB API calls.
- DB-API is Python's standard API for accessing relational databases.

- Benifits of DB-API:
  - easy to implement & understand
  - achieves consistency
  - portable across databases
  - broad reach of database connectivity from python.

Page No.
Date

• Main Concepts of Python DB-API

A. Connection Objects
- connect to a database & manage your transaction.

B. Cursor Objects
- used to run queries.
- used to scan throught the results of a database.
- retrieves results

• The DB-API includes a connect constructor for creating connection to database. ⟶ It returns a connection object which is then used by various connection methods.

• Connection methods :-
- .cursor() - returns new cursor object using connection
- .commit() - used to commit any pending transaction to the database
- .rollback() - causes database to rollback to the start of any pending transaction.
- .close() - used to close database connection.

• Database Cursor
- A database cursor is a control structure that enables traversal over the records in database.

Page No.
Date

- Writing code using DB-API.

```
from dbmodule import connect
# create connection object
connection = connect ('databasename', 'username', 'pswd')

# create cursor object
cursor = connection.cursor()

# run queries
cursor.execute ('select * from mytable')
results = cursor.fetchall()

# free resources
cursor.close()
```

#

~~Module 4~~

Module # 4. I.(C) : Accessing Database with SQL Magic

goal: what is SQL Magic, line magics & cell magics, use SQL magic in Jupyter notebooks.

- **Magic Commands**
  - special commands
  - not valid python code but affect the behavior of the notebook.
  - designed to solve standard data analysis problem.

- **Types of Magic Statements:**

1. **Line Magics**
   - commands prefixed with a single % character,
   - operate on single line of input.

2. **Cell Magics**
   - commands prefixed with two %% characters
   - operate on multiple lines of input.
   - can transform entire cell, or execute cell in different programming language

- **Some examples :-**

%pwd - prints current working directory
%ls - list all files in current directory
%history - show all command history
%matplotlib inline - makes plots appear within notebook
%timeit - time the execution of a single statement

%. timeit - time execution of a single statement (.)

%%. timeit - time execution of the entire cell.

%%. writefile myfile.txt - write all statements of the cell to txt file mentioned.

• Cell Magic Statements - to write other languages

- %%. HTML - write HTML code in cells
- %%. javascript - write Java code in cells
- %%. bash cell - write bash commands

• Necessary dependencies / libraries for SQL Magic

- ipython-sql

```
!pip install --user ipython-sql
```

- enable SQL magic in Jupyter notebook

```
%load_ext sql
```

- Make sure to establish connection b/t SQL server & SQL Magic model of the python notebook.

eg

```
import sqlite3
conn = sqlite3.connect('HR.db')
```

```
%load_ext sql  ⟶ loading SQL magic
%sql sqlite:///HR.db  ⟶ establishing conn^n b/t sqlMagic & SQLite server
```

Module # 4.I.(D): Analyzing Data with Python.

- sqlite 3 -> process python library that implements a that implements, self-contained, serverless, zero configuration transactional SQL, data base engine.

Module 5 : Course Assignment

Chapter (I) : Working with real world datasets & built in SQL functions.

Module # 5.(I).(A) : Assignment Preparation. with real world data

goals: discuss format of databases, loading of data in database

- CSV files
  - one of most widely used file types
  - the first column usually consist of Attribute labels, (column -

- Tips
  - If the column names has spaces or special character.
    → use backticks (`) to enclose the column names.
    ↳ button below esc key.

    eg:

    ```
    SELECT `Name of Dog` from dogs.
    ```

  - spilling long queries in Jupyter by using "\"

    ```
    %. sql SELECT * \
           FROM DOGS \
           WHERE `Name of Dog` = 'Huggy';
    ```

Page No.
Date

- Using quotes in Jupyter Notebook

data = pandas.read_sql(querystatement, connection_variable)

querystatement = 'SELECT "Name of Dog" FROM dogs'

- using backslash as the escape character in cases where the query contains single quotes.

querystatement = 'SELECT * FROM dogs WHERE
"Name of Dog" = \'Huggy\''

Module # 5.I.(B): Getting Table and Column details

- Getting a list of tables in the database:

- A database might have more than one table. So it's important to be aware about the other tables.
- ~~In IBM DB2~~
- Here's how to get a list of tables from every SQL server

- DB2 — syscat.tables
- sqlite3 — sqlite_master
- MySQL — SHOW TABLES

Page No.
Date

- Query

SQLite3: SELECT name FROM sqlite_master WHERE type = "table";

~~select name from~~

MySQL: SHOW TABLES

- Getting table attributes

SQLite 3 → PRAGMA table_info ([table_name])

MySQL → DESCRIBE table_name