6

SQL Data Science Python

Week 1: Basic SQL

Command	Syntax	Description	Example
SELECT	`SELECT column1, column2, ... FROM table_name;`	SELECT statement is used to fetch data from a database.	`SELECT city FROM placeofinterest;`
WHERE	`SELECT column1, column2, ...FROM table_name WHERE condition;`	WHERE clause is used to extract only those records that fulfill a specified condition.	`SELECT * FROM placeofinterest WHERE city == 'Rome' ;`
COUNT	`SELECT COUNT * FROM table_name ;`	COUNT is a function that takes the name of a column as argument and counts the number of rows when the column is not NULL.	`SELECT COUNT(country) FROM placeofinterest WHERE country='Canada';`
DISTINCT	`SELECT DISTINCT columnname FROM table_name;`	DISTINCT function is used to specify that the statement is a query which returns unique values in specified columns.	`SELECT DISTINCT country FROM placeofinterest WHERE type='historical';`
LIMIT	`SELECT * FROM table_name LIMIT number;`	LIMIT is a clause to specify the maximum number of rows the result set must have.	`SELECT * FROM placeofinterest WHERE airport="pearson" LIMIT 5;`
INSERT	`INSERT INTO table_name (column1,column2,column3...) VALUES(value1,value2,value3...);`	INSERT is used to insert new rows in the table.	`INSERT INTO placeofinterest (name,type,city,country,airport) VALUES('Niagara Waterfalls','Nature','Toronto','Canada','Pearson');`
UPDATE	`UPDATE table_name SET[[column1]=[VALUES]] WHERE [condition];`	UPDATE used to update the rows in the table.	`UPDATE placeofinterest SET name = 'Niagara Falls' WHERE name = "Niagara Waterfalls";`
DELETE	`DELETE FROM table_name WHERE [condition];`	DELETE statement is used to remove rows from the table which are specified in the WHERE condition.	`DELETE FROM placeofinterest WHERE city IN ('Rome','Vienna');`

Author(s)

Malika Singla

Skills Network

Changelog

Date	Version	Changed by	Change Description
2023-05-04	1.1	Benny	Formatting changes
2021-07-27	1.0	Malika	Initial Version

 Week2: Introduction to RDBMS

SQL Cheat Sheet: CREATE TABLE, ALTER, DROP, TRUNCATE

Command	Syntax	Description	Example
CREATE TABLE	`CREATE TABLE table_name (col1 datatype optional keyword, col2 datatype optional keyword,col3 datatype optional keyword,..., coln datatype optional keyword)`	`CREATE TABLE` statement is to create the table. Each column in the table is specified with its name, data type and an optional keyword which could be **PRIMARY KEY**, **NOT NULL**, etc.,	`CREATE TABLE employee ( employee_id char(2) PRIMARY KEY, first_name varchar(30) NOT NULL, mobile int);`
ALTER TABLE - ADD COLUMN	`1. ALTER TABLE table_name ADD column_name_1 datatype....ADD COLUMN column_name_n datatype;` `2. ALTER TABLE table_name ADD COLUMN column_name_1 datatype....ADD COLUMN column_name_n datatype;`	`ALTER TABLE` statement is used to add the columns to a table.	`1. ALTER TABLE employee ADD income bigint;` `2. ALTER TABLE employee ADD COLUMN income bigint;`
ALTER TABLE - ALTER COLUMN	`ALTER TABLE table_name ~~ALTER COLUMN~~ column_name_1 ~~SET DATA TYPE~~ datatype;` MODIFY	`ALTER TABLE ALTER COLUMN` statement is used to modify the data type of columns.	`ALTER TABLE employee [redacted] mobile ~~int~~ [redacted] CHAR(20);` MODIFY
ALTER TABLE - DROP COLUMN	`ALTER TABLE table_name DROP COLUMN column_name_1 ;`	`ALTER TABLE DROP COLUMN` statement is used to remove columns from a table.	`ALTER TABLE employee DROP COLUMN mobile ;`
ALTER TABLE - RENAME COLUMN	`ALTER TABLE table_name RENAME COLUMN current_column_name TO new_column_name;`	`ALTER TABLE RENAME COLUMN` statement is used to rename the columns in a table.	`ALTER TABLE employee RENAME COLUMN first_name TO name ;`
TRUNCATE TABLE	`TRUNCATE TABLE table_name ~~IMMEDIATE~~;`	`TRUNCATE TABLE` statement is used to delete all of the rows in a table. The IMMEDIATE specifies to process the statement immediately and that it cannot be undone.	`TRUNCATE TABLE employee ~~IMMEDIATE~~ ;`
DROP TABLE	`DROP TABLE table_name ;`	Use the `DROP TABLE` statement to delete a table from a database. If you delete a table that contains data, by default the data will be deleted alongside the table.	`DROP TABLE employee ;`

Author(s)

Himanshu Birla

Skills Network

Changelog

Date	Version	Changed by	Change Description
2023-05-04	1.1	Benny Li	Formatting changes made
2021-07-27	1.0	Himanshu Birla	Initial Version

 Week3a: Intermediate SQL: Refining Results.

SQL Cheat Sheet: Intermediate - LIKE, ORDER BY, GROUP BY

Command	Syntax	Description	Example
LIKE	`SELECT column1, column2, ... FROM table_name WHERE columnN LIKE pattern;`	`LIKE` operator is used in a WHERE clause to search for a specified pattern in a column. Two wildcards often used in conjunction with the LIKE operator are percent sign(%) and underscore sign (_), depending upon the SQL engine being used.	`SELECT f_name , l_name FROM employees WHERE address LIKE '%Elgin,IL%';` This command will output all entries with `Elgin,IL` in the Address.
BETWEEN	`SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;`	The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates. The `BETWEEN` operator is inclusive: begin and end values are included.	`SELECT * FROM employees WHERE salary BETWEEN 40000 AND 80000;` This generates all records of employees with salaries between 40000 and 80000.
ORDER BY	`SELECT column1, column2, ... FROM table_name ORDER BY column1, column2, ... ASC|DESC;`	`ORDER BY` keyword is used to sort the result-set in ascending or descending order. The default is ascending. In case of multiple columns in ORDER BY, the sorting will be done in the sequence of the appearance of the arguments.	`SELECT f_name, l_name, dep_id FROM employees ORDER BY dep_id DESC, l_name;` This displays the first name, last name, and department ID of employees, first sorted in descending order of department IDs and then sorted alphabetically as per their last names.
GROUP BY	`SELECT column_name(s) FROM table_name GROUP BY column_name(s)`	`GROUP BY` clause is used in collaboration with the SELECT statement to arrange data with identical values into groups.	`SELECT dep_id, COUNT(*) FROM employees GROUP BY dep_id;` This returns the department IDs and the number of employees in them, grouped by the department IDs.
HAVING	`SELECT column_name(s) FROM table_name GROUP BY column_name(s) HAVING condition`	`HAVING` clause is used in conjunction with `GROUP BY` clause in collaboration with the SELECT statement in order to filter the data as per the given condition and then group as per identical values of a specified parameter.	`SELECT DEP_ID, COUNT(*) AS "NUM_EMPLOYEES", AVG(SALARY) AS "AVG_SALARY" FROM EMPLOYEES GROUP BY DEP_ID HAVING count(*) < 4 ORDER BY AVG_SALARY;`

Author(s)

Lakshmi Holla
Abhishek Gagneja

Skills Network

Changelog

Date	Version	Changed by	Change Description
2023-10-03	1.3	Steve Hord	QA pass with edits
2023-10-01	1.2	Abhishek Gagneja	Updated the document
2023-05-04	1.1	Benny Li	Formatting changes
2021-07-28	1.0	Lakshmi Holla	Initial Version

Week 3b : Intermediate SQL: Functions, Multiple Tables & Sub-queries

SQL Cheat Sheet: FUNCTIONS and Implicit JOIN

Command	Syntax	Description	Example
COUNT	`SELECT COUNT(column_name) FROM table_name WHERE condition;`	`COUNT` function returns the number of rows that match a specified criterion.	`SELECT COUNT(dep_id) FROM employees;`
AVG	`SELECT AVG(column_name) FROM table_name WHERE condition;`	`AVG` function returns the average value of a numeric column.	`SELECT AVG(salary) FROM employees;`
SUM	`SELECT SUM(column_name) FROM table_name WHERE condition;`	`SUM` function returns the total sum of a numeric column.	`SELECT SUM(salary) FROM employees;`
MIN	`SELECT MIN(column_name) FROM table_name WHERE condition;`	`MIN` function returns the smallest value of the SELECTED column.	`SELECT MIN(salary) FROM employees;`
MAX	`SELECT MAX(column_name) FROM table_name WHERE condition;`	`MAX` function returns the largest value of the SELECTED column.	`SELECT MAX(salary) FROM employees;`
ROUND	`SELECT ROUND(2number, decimals, operation) AS RoundValue;`	`ROUND` function rounds a number to a specified number of decimal places.	`SELECT ROUND(salary) FROM employees;`
LENGTH	`SELECT LENGTH(column_name) FROM table;`	`LENGTH` function returns the length of a string (in bytes).	`SELECT LENGTH(f_name) FROM employees;`
UCASE	`SELECT UCASE(column_name) FROM table;`	`UCASE` function displays the column name in each table in uppercase.	`SELECT UCASE(f_name) FROM employees;`
LCASE	`SELECT LCASE(column_name) FROM table;`	`LCASE` function displays the column name in each table in lowercase.	`SELECT LCASE(f_name) FROM employees;`
DISTINCT	`SELECT DISTINCT column_name FROM table;`	`DISTINCT` function is used to display data without duplicates.	`SELECT DISTINCT UCASE(f_name) FROM employees;`
DAY	`SELECT DAY(column_name) FROM table;`	`DAY` function returns the day of the month for a given date.	`SELECT DAY(b_date) FROM employees where emp_id = 'E1002';`
CURRENT_DATE	`SELECT CURRENT_DATE;`	`CURRENT_DATE` is used to display the current date.	`SELECT CURRENT_DATE;`
DATEDIFF()	`SELECT DATEDIFF(date1, date2);`	`DATEDIFF()` is used to calculate the difference between two dates or time stamps. The default value generated is the difference in number of days.	`SELECT DATEDIFF(CURRENT_DATE, date_column) FROM table;`
FROM_DAYS()	`SELECT FROM_DAYS(number_of_days);`	`FROM_DAYS()` is used to convert a given number of days to YYYY-MM-DD format.	`SELECT FROM_DAYS(DATEDIFF(CURRENT_DATE, date_column)) FROM table;`
DATE_ADD()	`SELECT DATE_ADD(date, INTERVAL n type);`	`DATE_ADD()` is used to calculate the date after lapse of mentioned number of units of date type, i.e. if n=3 and type=DAY, the result is a date 3 days after what is mentioned in date column. The type valiable can also be months or years.	`SELECT DATE_ADD(date, INTERVAL 3 DAY);;`
DATE_SUB()	`SELECT DATE_SUB(date, INTERVAL n type);`	`DATE_SUB()` is used to calculate the date prior to the record date by mentioned number of units of date type, i.e. if n=3 and type=DAY, the result is a date 3 days before what is mentioned in date column. The type valiable can also be months or years.	`SELECT DATE_SUB(date, INTERVAL 3 DAY);;`
Subquery	`SELECT column_name [, column_name ] FROM table1 [, table2 ] WHERE column_name OPERATOR (SELECT column_name [, column_name ] FROM table1 [, table2 ] [WHERE])`	`Subquery` is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.	`SELECT emp_id, fmame, lname, salary FROM employees where salary < (SELECT AVG(salary) FROM employees);` `SELECT * FROM ( SELECT emp_id, f_name, l_name, dep_id FROM employees) AS emp4all;`
Implicit Inner Join	`SELECT column_name(s) FROM table1, table2 WHERE table1.column_name = table2.column_name;`	`Implicit Inner Join` combines two or more records but displays only matching values in both tables. Inner join applies only the specified columns.	`SELECT * FROM employees WHERE job_id IN (SELECT job_ident FROM jobs);` `SELECT * FROM employees, jobs where employees.job_id = jobs.job_ident;`
Implicit Cross Join	`SELECT column_name(s) FROM table1, table2;`	`Implicit Cross Join` is defined as a Cartesian product where the number of rows in the first table is multiplied by the number of rows in the second table.	`SELECT * FROM employees, jobs;`

Author(s)

akshmi Holla
Abhishek Gagneja

Skills Network

Subqueries

- To evaluate aggregate functions sub-Select expression (in where clause)

```
Select emp_id, f_name, salary, ...
from employee
where salary <
(select avg(salary) from employees
```

- Substitute column name with sub-query (Column expressions) (in SELECT clause)

```
Select emp_id, salary,
   (select avg(salary) from empl
    as AVG_SALARY
from employees;
```

- Sub-queries in FROM Clause (Derived tables or Table expressions)

```
Select * from
   (select emp_id, f_name, ...
    from employees)
   as empAll;
```

Week 4: Accessing Databases using Python

Menu

Summary: Accessing databases using Python

Congratulations! You have completed this lesson. At this point in the course, you know:

- Magic commands are special commands that provide special functionalities.
- Line magics are commands prefixed with a single % character... single line of input
- Cell magics are commands prefixed with two %% characters and operate on multiple input lines.
- DB APIs are commands prefixed with two %% characters and operate on multiple input lines.
- The two main concepts in the Python DB API are Connection Objects and Query Objects.
- A database cursor is a control structure that enables traversal over the records in a database.
- Pandas' methods are equipped with common mathematical and statistical methods.
- The pandas.read_csv functionis used to read the database CSV file.
- The SQLite3.connect functionis used to connect to a database.
- To use pandas to retrieve data from the database tables, load data using the read_sql method and select the SQL Select Query
- A categorical scatterplot is created using the swarmplot() method by the seaborn package.

Concepts of Python DB API

- Connection objects
  - Database connections
  - Manage transactions

  Methods [.cursor(), .close()]

```
from dbmodule import connect
# Create connection object
Connection = connect('database name', 'username', 'pswd')
```

- Cursor objects (instance to invoke methods that execute SQLite statements, fetch data etc.)
  - Used to run queries
  - Scrolls through result sets and retrieves results

  Methods: .execute(), .fetchall(), .fetchmany(), .close()

```
# Create cursor object
Cursor = connection.cursor()
# Run queries
Cursor.execute('select * from mytable')
Results = cursor.fetchall()
for row in Results:
    print(row)
```

Line Magic statements

```
% matplotlib inline
% timeit # time for execution of a single statement
% lsmagic # lists all available line magics
%% timeit # Cell magic: time for executing the whole cell.
```

Cell magic statements

```
%%HTML : Writes HTML code in cells and render it
%%javascript
%%bash
```

SQL magic

```
!pip install --user ipython-sql
% load_ext sql
% sql sqlite:///HR.db # Establish a connection for the SQL magic module to the SQL server being accessed.
% sql select * from table_name
```

```
sql_magic_query = %sql select * from ...
df = sql_magic_query.DataFrame()
```

Pandas

```
# Use backslash "\" to split the query into multiple lines
% sql select Id, 'Name of dog', \
    from dogs \
    where condition
```

or use %%.sql in the cell

```
# Use backticks for col_names with spaces
```

Pandas

```
df = pandas.read_csv('url' or 'path')
df.to_sql('table_name', connection, if_exists='replace', index=False, method='multi
```

```
# Use pandas and quotes. Pandas
datax = pandas.read_sql(query_statement, connection)
query_statement = 'select "Name of dog" from dogs'
                  or  >>  >>  where "Name of dog" = \'Huggy\'
```

Getting list of tables

Show Tables (MySQL)
sqlite_master (SQLite3): Select name from sqlite_master where type='table'
syscat.tables (DB2)

Getting table attributes

pragma_table_info('table_name') (SQLite3): select name, type, length(type) from pragma_...
describe table_name (MySQL)

SQL Cheat Sheet: Accessing Databases using Python

SQLite

Topic	Syntax	Description	Example
connect()	`sqlite3.connect()`	Create a new database and open a database connection to allow sqlite3 to work with it. Call `sqlite3.connect()` to create a connection to the database INSTRUCTOR.db in the current working directory, implicitly creating it if it does not exist.	1. 1 2. 2 1. import sqlite3 2. con = sqlite3.connect("INSTRUCTOR.db") Copied!
cursor()	`con.cursor()`	To execute SQL statements and fetch results from SQL queries, use a database cursor. Call `con.cursor()` to create the Cursor.	1. 1 1. cursor_obj = con.cursor() Copied!
execute()	`cursor_obj.execute()`	The `execute` method in Python's SQLite library allows to perform SQL commands, including retrieving data from a table using a query like "Select * from table_name." When you execute this command, the result is obtained as a collection of table data stored in an object, typically in the form of a list of lists.	1. 1 1. cursor_obj.execute('''insert into INSTRUCTOR values (1, 'Rav', 'Ahuja', 'TORONT(Copied!
fetchall()	`cursor_obj.fetchall()`	The `fetchall()` method in Python retrieves all the rows from the result set of a query and presents them as a list of tuples.	1. 1 2. 2 3. 3 4. 4 5. 5 1. statement = '''SELECT * FROM INSTRUCTOR''' 2. cursor_obj.execute(statement) 3. output_all = cursor_obj.fetchall() 4. for row_all in output_all: 5. print(row_all) Copied!
fetchmany()	`cursor_obj.fetchmany()`	The `fetchmany()` method retrieves the subsequent group of rows from the result set of a query rather than just a single row. To fetch a few rows from the table, use fetchmany(numberofrows) and mention how many rows you want to fetch.	1. 1 2. 2 3. 3 4. 4 5. 5 1. statement = '''SELECT * FROM INSTRUCTOR''' 2. cursor_obj.execute(statement) 3. output_many = cursor_obj.fetchmany(2) 4. for row_many in output_many: 5. print(row_many) Copied!
read_sql_query()	`read_sql_query()`	`read_sql_query()` is a function provided by the Pandas library in Python, and it is not specific to MySQL. It is a generic function used for executing SQL queries on various database systems, including MySQL, and retrieving the results as a Pandas DataFrame.	1. 1 1. df = pd.read_sql_query("select * from instructor;", conn) Copied!
shape	`dataframe.shape`	It provides a tuple indicating the shape of a DataFrame or Series, represented as (number of rows, number of columns).	1. 1 1. df.shape Copied!
close()	`con.close()`	`con.close()` is a method used to close the connection to a MySQL	1. 1 1. con.close()

		database. When called, it terminates the connection, releasing any associated resources and ensuring the connection is no longer active. This is important for managing database connections efficiently and preventing resource leaks in your MySQL database interactions.	Copied!
CREATE TABLE	`CREATE TABLE table_name ( column1 datatype constraints, column2 datatype constraints, ... );`	The `CREATE TABLE` statement is used to define and create a new table within a database. It specifies the table's name, the structure of its columns (including data types and constraints), and any additional properties such as indexes. This statement essentially sets up the blueprint for organizing and storing data in a structured format within the database.	1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 1. CREATE TABLE INTERNATIONAL_STUDENT_TEST_SCORES (2. country VARCHAR(50), 3. first_name VARCHAR(50), 4. last_name VARCHAR(50), 5. test_score INT 6.); Copied!
barplot()	`seaborn.barplot(x="x-axis_variable", y="y-axis_variable", data=data)`	`seaborn.barplot()` is a function in the Seaborn Python data visualization library used to create a bar plot, also known as a bar chart. It is particularly used to display the relationship between a categorical variable and a numeric variable by showing the average value for each category.	1. 1 2. 2 1. import seaborn 2. seaborn.barplot(x='Test_Score',y='Frequency', data=dataframe) Copied!
read_csv()	`df = pd.read_csv('file_path.csv')`	`read_csv()` is a function in Python's Pandas library used for reading data from a Comma-Separated Values (CSV) file and loading it into a Pandas DataFrame. It's a common method for working with tabular data stored in CSV format	1. 1 2. 2 1. import pandas 2. df = pandas.read_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv') Copied!
to_sql()	`df.to_sql('table_name', index=False)`	`df.to_sql()` is a method in Pandas, a Python data manipulation library used to write the contents of a DataFrame to a SQL database. It allows to take data from a DataFrame and store it structurally within a SQL database table.	1. 1 2. 2 3. 3 1. import pandas 2. df = pandas.read_csv('https://data.cityofchicago.org/resource/jcxq-k9xf.csv') 3. df.to_sql("chicago_socioeconomic_data", con, if_exists='replace', index=False, Copied!
read_sql()	`df = pd.read_sql(sql_query, conn)`	`read_sql()` is a function provided by the Pandas library in Python for executing SQL queries and retrieving the results into a DataFrame from an SQL database. It's a convenient way to integrate SQL database interactions into your data analysis workflows.	1. 1 2. 2 1. selectQuery = "select * from INSTRUCTOR" 2. df = pandas.read_sql(selectQuery, conn) Copied!

Db2

Topic	Syntax	Description	Example
connect()	`conn = ibm_db.connect('DATABASE=dbname; HOST=hostname;PORT=port;UID=username; PWD=password;', '', '')`	`ibm_db.connect()` is a Python function provided by the ibm_db library, which is used for establishing a connection to an IBM Db2 or IBM	1. 1 2. 2 3. 3 4. 4 1. import ibm_db 2. conn = ibm_db.connect('DATABASE=mydb; 3. HOST=example.com;PORT=50000;UID=myuser;

		Db2 Warehouse database. It's commonly used in applications that need to interact with IBM Db2 databases from Python.	`4. PWD=mypassword;', '', '')` Copied!
server_info()	`ibm_db.server_info()`	`ibm_db.server_info(conn)` is a Python function provided by the `ibm_db` library, which is used to retrieve information about the IBM Db2 server to which you are connected.	`1. 1` `2. 2` `3. 3` `4. 4` `1. server = ibm_db.server_info(conn)` `2. print ("DBMS_NAME: ", server.DBMS_NAME)` `3. print ("DBMS_VER: ", server.DBMS_VER)` `4. print ("DB_NAME: ", server.DB_NAME)` Copied!
close()	`con.close()`	`con.close()` method used to close the connection to a db2 database. When called, it terminates the connection, releasing any associated resources and ensuring the connection is no longer active. This is important for managing database connections efficiently and preventing resource leaks in your db2 database interactions.	`1. 1` `1. con.close()` Copied!
exec_immediate()	`sql_statement = "SQL statement goes here"` `stmt = ibm_db.exec_immediate(conn, sql_statement)`	`ibm_db.exec_immediate()` is a Python function provided by the ibm_db library, which is used to execute an SQL statement immediately without the need to prepare or bind it. It's commonly used for executing SQL statements that don't require input parameters or don't need to be prepared in advance.	`1. 1` `2. 2` `3. 3` `1. # Lets first drop the table INSTRUCTOR in case it exists from a previc` `2. dropQuery = "drop table INSTRUCTOR"` `3. dropStmt = ibm_db.exec_immediate(conn, dropQuery)` Copied!

Author(s)

Abhishek Gagneja

D.M Naidu

Skills Network

Changelog

Date	Version	Changed by	Change Description
2023-10-30	1.2	Mary Stenberg	QA Pass with edits
2023-10-16	1.1	Abhishek Gagneja	Updated instruction set
2023-05-08	1.0	D.M.Naidu	Initial Version

SQL Queries

mysql-workbench.
terminal: mysql -u root -p

Database management.

> CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] DB_Name
> USE DB_Name
> DROP {DATABASE|SCHEMA} DB_Name
> SHOW Tables
> SHOW {databases|schemas}

Table management.

> CREATE TABLE TableName
{(colname datatype [constraints] [AUTO_INCREMENT]
[PRIMARY KEY (list of Columns)]
[FOREIGN KEY (list of FK columns) REFERENCES parentTableName
[(list of FK Columns)]}
[ON DELETE option]
[ON UPDATE option]
option: RESTRICT | CASCADE | SET NULL | SET DEFAULT
[CHECK (condition)]
> CREATE TABLE newTable LIKE Old_Table # cloning
> ALTER TABLE TableName alterSpecifications
> DROP TABLE tblName [, ...]
> DELETE FROM tblName # delete the values.
> CREATE [UNIQUE] INDEX IndexName ON TableName ColumnName
[ASC|DESC] [, ...])
> DROP INDEX indexName ON tableName
> DESCRIBE tableName
> CREATE TABLE new_table SELECT * FROM old_table # duplicate table
> INSERT INTO table2 SELECT * FROM table1 # populating a table exactly the same
> CREATE VIEW view_name AS SELECT statem. # view of a table
> DROP VIEW [IFEXISTS] viewName [, viewName]

Data Manipulation & Retrieval

> SELECT [DISTINCT|ALL] {* | [column expression [AS new Name]]}
FROM tableName [alias][,...]
[WHERE condition]
[GROUP BY columnList]
[HAVING condition]
[ORDER BY columnList]

SELECT COUNT(colname)
COUNT(*)
LIMIT N

> WHERE BETWEEN / NOT BETWEEN
IN / NOT IN.
LIKE / NOT LIKE
IS NULL / IS NOT NULL

> INSERT INTO tableName (columnList) VALUES (dataValueList)
> DELETE FROM tableName [WHERE searchCondition]
> UPDATE tableName SET colName1 = dataValue1 [colName2 = ...]
[WHERE search condition]

FROM Table1 T1 JOIN Table2 T2
ON T1.colname = T2.colname
FROM Table1 JOIN Table2 USING (colname)

```
ALTER TABLE tableName
    [alter_option [, alter_option]...]

alter_option: {
    table_options
  | RENAME [TO|AS] new_tableName  # renames table
  | ADD [COLUMN] col_name column_definition  # add column
  | MODIFY [COLUMN] col_name column_definition  # change datatype of a column
  | ALTER [COLUMN] col_name
        {SET DEFAULT {literal | (expr)} | DROP DEFAULT}  # set a default value
  | DROP [COLUMN] col_name  # drop a column.
```