**Module 1 – Getting Started with SQL**

**BASIC SQL**

**Introduction to Databases**

You can now describe:

* SQL
* Data
* Database
* DBMS
* RDBMS
* 5 Basic SQL Commands
  + Create, Insert, Select, Update, Delete

The search condition of the **WHERE** clause uses a predicate to refine the search.

**SELECT Statement**

Now you can

* retrieve data and select columns from a relational database table,
* define the use of a predicate,
* identify the syntax of the SELECT statement using the WHERE clause, and
* list the comparison operators supported by a relational database management system

Syntax:

SELECT column1, column2, ...

FROM table\_name

WHERE condition

;

**COUNT, DISTINCT, LIMIT**

* COUNT, DISTINCT, and LIMIT are expressions that are used with SELECT statements.

**Count:** SELECT COUNT \* FROM table\_name ;  
 **SELECT COUNT(\*) FROM FilmLocations;**

**SELECT COUNT(Locations) FROM FilmLocations WHERE Writer="James Cameron";**

**SELECT Count(\*) FROM FilmLocations WHERE ReleaseYear<1950;**

**Distinct:** SELECT DISTINCT columnname FROM table\_name;

**SELECT DISTINCT Title FROM FilmLocations;**

SELECT COUNT(DISTINCT ReleaseYear) FROM FilmLocations WHERE ProductionCompany="Warner Bros. Pictures";

Limit: SELECT \* FROM table\_name LIMIT number;

SELECT \* FROM FilmLocations LIMIT 25;

SELECT \* FROM FilmLocations LIMIT 15 OFFSET 10; *(starts from row 11.)*

**INSERT Statement**

Now you can

* identify the syntax of the INSERT statement,
* explain the two methods to add rows to a table. One row at a time or multiple rows.

Syntax:

INSERT INTO table\_name (column1, column2, ... )

VALUES (value1, value2, ... )

;

**UPDATE and DELETE Statements**

Now you can

* identify the syntax of the UPDATE statement and
* DELETE statement and explain the importance of the WHERE clause in these statements.

Syntax:

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition

;

DELETE FROM table\_name

WHERE condition

;

**Module 2**

**Introduction to Relational Databases and Tables**

**Relational Database Concepts**

Now you know that

* the key advantage of the relational model is logical and physical data independence and storage independence.
* Entities are independent objects which can have multiple characteristics called attributes. When mapping to a relational database, entities are represented as tables and attributes map to columns.
* Common data types include characters such as car and VAR char, numbers such as integer and decimal, and timestamps including date and time.
* A primary key uniquely identifies a specific row in a table and prevents duplication of data

**Types of SQL statements (DDL vs. DML)**

Now you know that:

* DDL or Data Definition Language statements are used for defining or changing objects in a database such as tables. (Create, Alter, Truncate, Drop)
* DML or Data Manipulation Language statements are used for manipulating or working with data in tables (CRUD Operations & Common DML I.e Insert, Select, Update, Delete)

**CREATE TABLE Statement**

Now you know that:

* CREATE is a DDL statement for creating Entities or tables in a database.
* The CREATE TABLE statement includes definition of attributes of columns in the table, including
  + Names of columns;
  + Datatypes of columns;
  + and other Optional values if required such as the Primary Key constraint

create table TABLENAME (  
COLUMN1 datatype,  
COLUMN2 datatype,  
COLUMN3 datatype,  
…  
);

**ALTER, DROP, and Truncate tables**

* The ALTER TABLE statement changes the structure of an existing table, for example, to add, modify, or drop columns.
* The DROP TABLE statement deletes an existing table.
* The TRUNCATE TABLE statement deletes all rows of data in a table.

ALTER TABLE table\_name

ADD COLUMN column\_name data\_type column\_constraint;

ALTER TABLE table\_name CHANGE column\_name

column\_name VARCHAR(20);

TRUNCATE TABLE table\_name;

DROP TABLE table\_name;

**Module 3 – INTERMEDIATE SQL**

**REFINING YOUR RESULTS**

**Using String Patterns and Ranges**

**Sorting result sets**

**Grouping Result Sets**

‘Where’ clause can be used for entire dataset, ‘having’ clause can be only used for Group By statements.

SELECT column1, column2, ... FROM table\_name WHERE columnN LIKE pattern;

SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2;

SELECT column1, column2, ... FROM table\_name ORDER BY column1, column2, ... ASC|DESC;

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s) ORDER BY column\_name(s);

**Summary & Highlights**

* You can use the WHERE clause to refine your query results.
* You can use the wildcard character (%) as a substitute for unknown characters in a pattern.
* You can use BETWEEN ... AND ... to specify a range of numbers.
* You can sort query results into ascending or descending order, using the ORDER BY clause to specify the column to sort on.
* You can group query results by using the GROUP BY clause.

**Functions, Multiple Tables, and Sub-queries**

**Built-in Database Functions**

**Date and Time Built-in Functions**

**Sub-Queries and Nested Selects**

Select \* from employees, where salary > AVG(salary). ----> throws error

However, running this query will result in an error like the one shown. Indicating an invalid use of the aggregate function.

One of the limitations of built in aggregate functions, like the average function, is that they cannot always be evaluated in the WHERE clause. So to evaluate a function like average in the WHERE clause, we can make use of a sub-select expression like the one shown here

Select EMP\_ID, F\_NAME, L\_NAME, SALARY from employees where SALARY < ( select AVG (SALARY) from employees).

Sub-Queries

**SELECT column\_name [, column\_name ]**

**FROM table1 [, table2 ]**

**WHERE column\_name OPERATOR**

**(SELECT column\_name [, column\_name ]**

**FROM table1 [, table2 ]**

**WHERE condition);**

**Working with Multiple Tables**

Implicit version of CROSS JOIN (also known as Cartesian Join) statement syntax

SELECT column\_name(s) FROM table1, table2;

Implicit version of INNER JOIN statement syntax

SELECT column\_name(s)

FROM table1, table2

WHERE table1.column\_name = table2.column\_name;

**Summary & Highlights**

* Most databases come with built-in functions that you can use in SQL statements to perform operations on data within the database itself.
* When you work with large datasets, you may save time by using built-in functions rather than first retrieving the data into your application and then executing functions on the retrieved data.
* You can use sub-queries to form more powerful queries than otherwise.
* You can use a sub-select expression to evaluate some built-in aggregate functions like the average function.
* Derived tables or table expressions are sub-queries where the outer query uses the results of the sub-query as a data source.

**Note**: You can retrieve information from multiple tables by using a **sub-query, an implicit join, or a JOIN operator** like INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, or FULL OUTER JOIN.

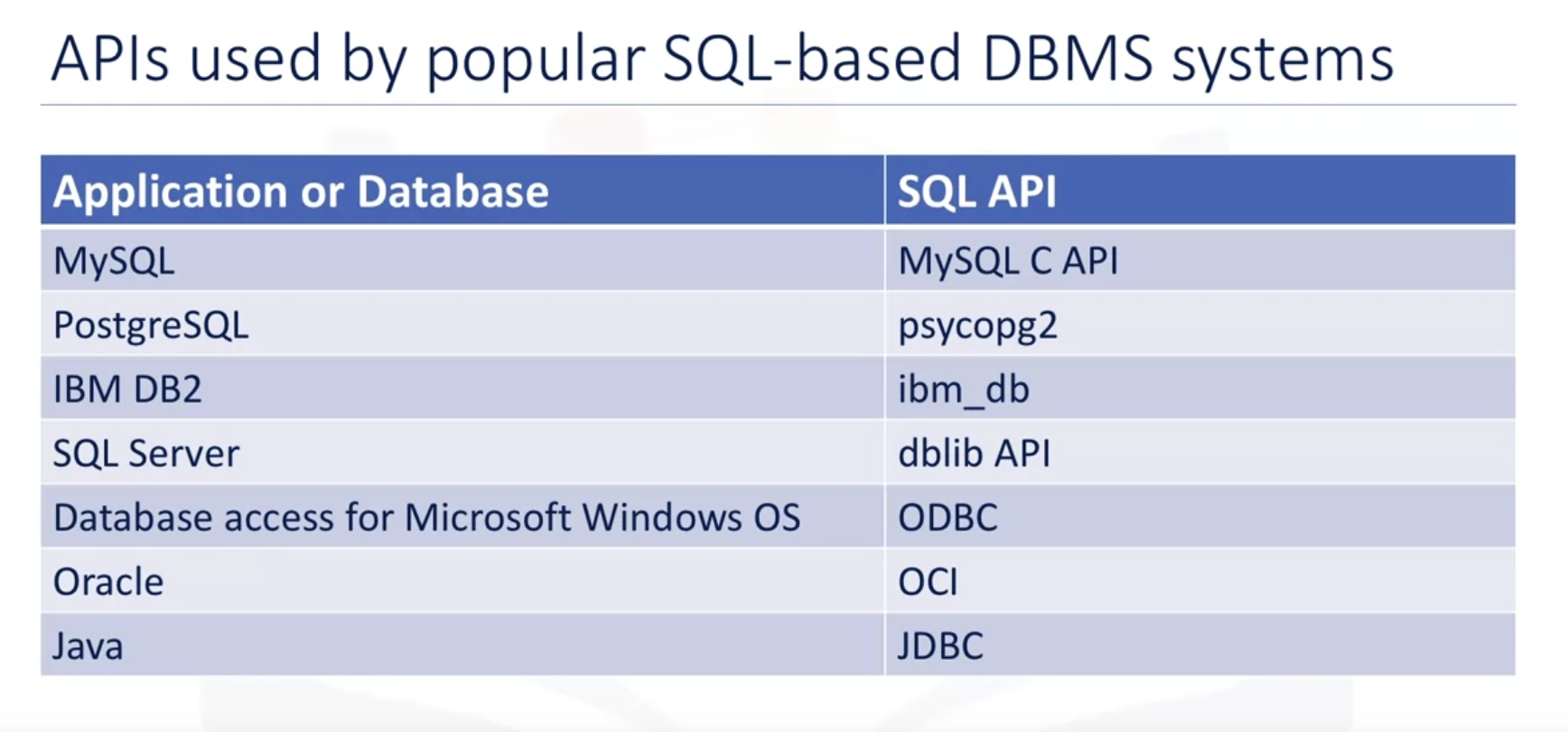
|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Syntax** | **Description** | **Example** |
| COUNT | SELECT COUNT(column\_name) FROM table\_name WHERE condition; | COUNT function returns the number of rows that matches a specified criterion. | SELECT COUNT(dep\_id) FROM employees; |
| AVG | 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 that displays the column name in each table in uppercase. | SELECT UCASE(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 - COLUMN) FROM table; | CURRENT DATE is used to display the current date.This can be subtracted from the previous date to get the difference. | SELECT YEAR(CURRENT DATE - b\_date) As AGE, CURRENT\_DATE, b\_date FROM employees; |
| 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;    SELECT \* FROM employees WHERE job\_id IN (SELECT job\_ident FROM jobs); |
| Implicit Inner Join | SELECT column\_name(s) FROM table1, table2 WHERE table1.column\_name = table2.column\_name; | Implicit Inner Join combines the two or more records but displays only matching values in both tables. Inner join applies only the specified columns. | SELECT \* FROM employees, jobs where employees.job\_id = jobs.job\_ident; |
| Implicit Cross Join | SELECT column\_name(s) FROM table1, table2; | Implicit Cross Join defines as a Cartesian product where the number of rows in the first table multiplied by the number of rows in the second table.. | SELECT \* FROM employees, jobs; |

**Module 4**

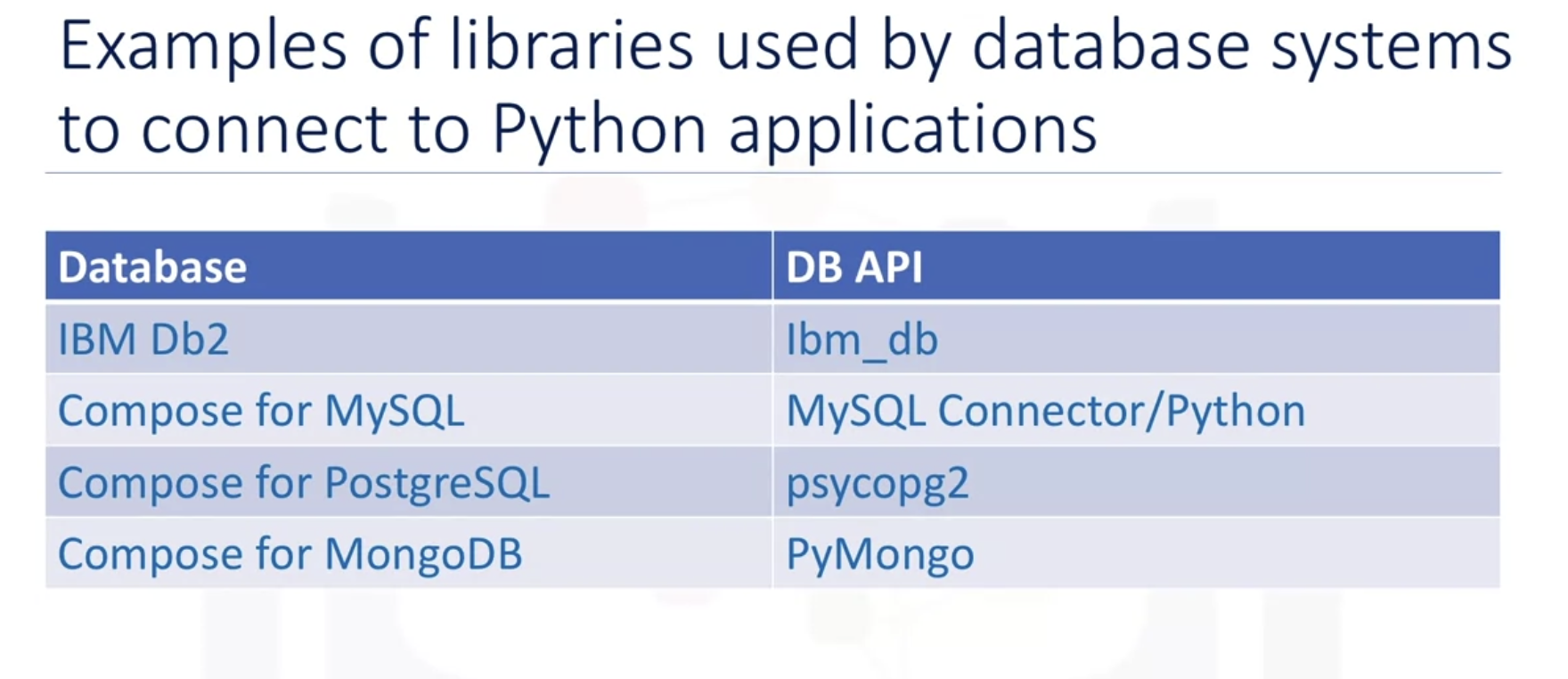
**ACCESSING THE DATABASE USING PYTHON**

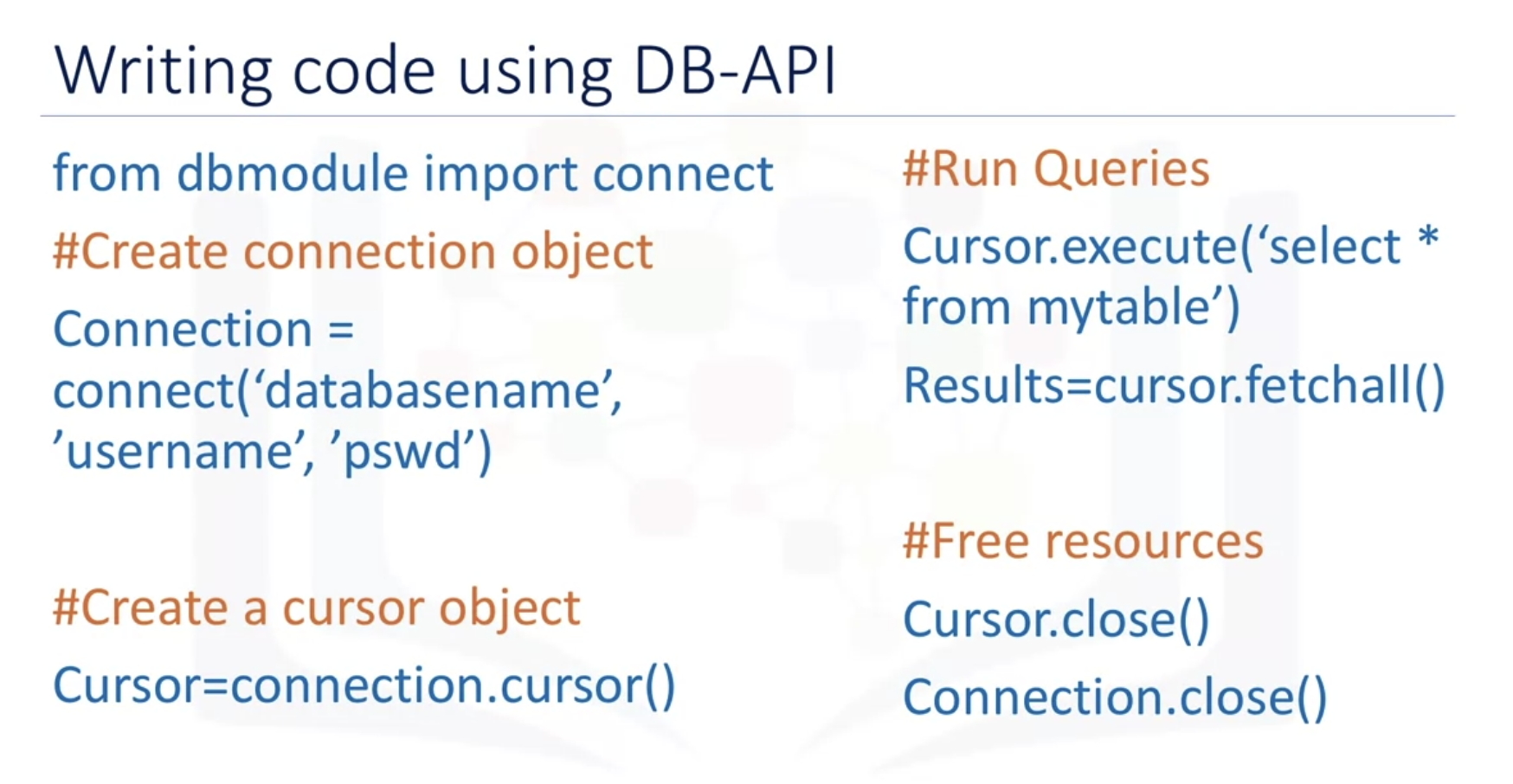
**(Optional) How to Access Databases Using Python**





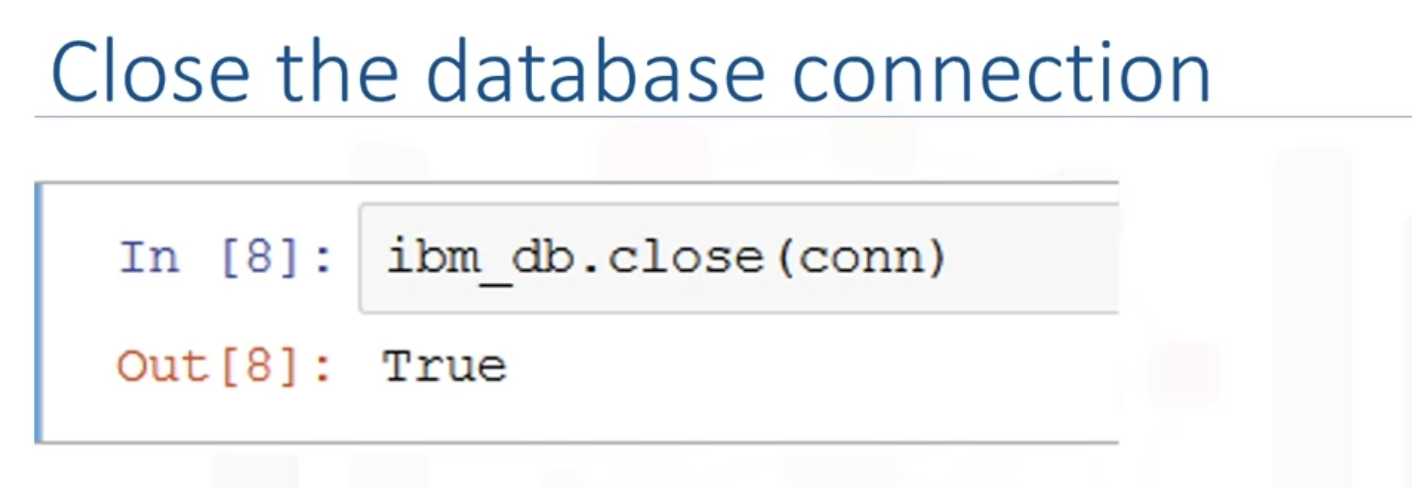
**(Optional) Writing code using DB-API**





**(Optional) Connecting to a database using ibm\_db API**





**Introducing SQL Magic**

Jupyter notebooks have a concept of Magic commands that can simplify working with Python, and are particularly useful for data analysis. Your notebooks can have two types of magic commands:

Cell magics: start with a double %% sign and apply to the entire cell

Line magics: start with a single % (percent) sign and apply to a particular line in a cell

Their usage is of the format:

%magicname arguments

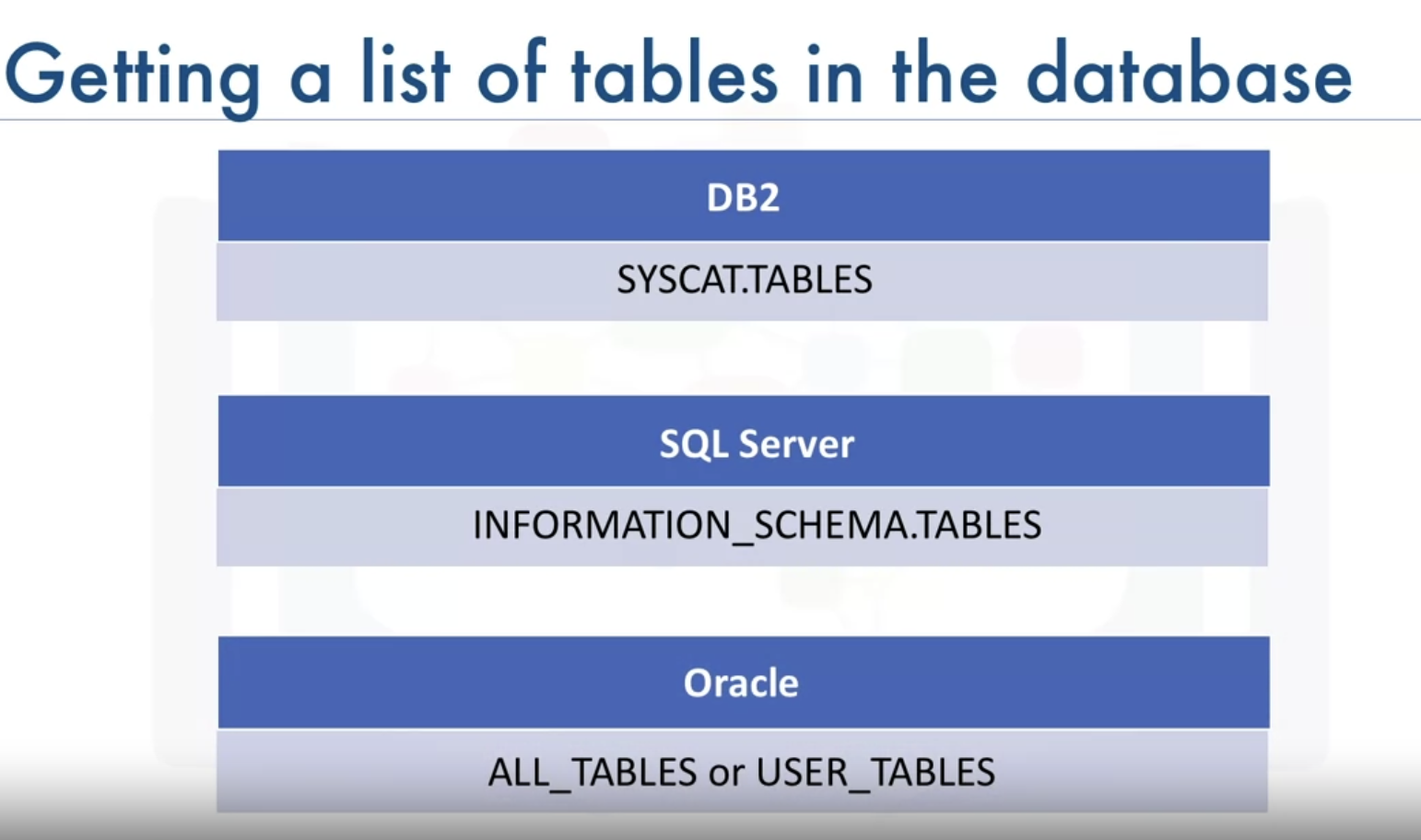
%sql select \* from tablename

**(OPTIONAL) Analyzing data with Python**

**Summary & Highlights**

* You can access a database from a language like Python by using the appropriate API. Examples include ibm\_db API for IBM DB2, psycopg2 for ProstgreSQL, and dblib API for SQL Server.
* DB-API is Python's standard API for accessing relational databases. It allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one.
* The DB\_API connect constructor creates a connection to the database and returns a Connection Object, which is then used by the various connection methods.
* The connection methods are:
  1. The cursor() method, which returns a new cursor object using the connection.
  2. The commit() method, which is used to commit any pending transaction to the database.
  3. The rollback() method, which causes the database to roll-back to the start of any pending transaction.
  4. The close() method, which is used to close a database connection.
* You can use SQL Magic commands to execute queries more easily from Jupyter Notebooks. Magic commands have the general format %sql select \* from tablename. Cell magics start with a double %% (percent) sign and apply to the entire cell. Line magics start with a single % (percent) sign and apply to a particular line in a cell.

**Module 5 – COURSE ASSIGNMENT**



**Module 6**

**ADVANCE SQL FOR DATA ENGINEERING**

**Views**

* Views are an alternate way of accessing data in tables. They can include specified columns from multiple base tables and existing views.
* Once created, views can be queried like a table, and the data in the base table can be modified through the view.
* Views are dynamic; only the definition of the view is stored, not the data.
* You can use the CREATE VIEW statement to create a view based on one or more tables or existing views.

Syntax of a CREATE VIEW statement:

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

Syntax of a REPLACE VIEW statement:

CREATE OR REPLACE VIEW view\_name AS

SELECT column1, column2, ...

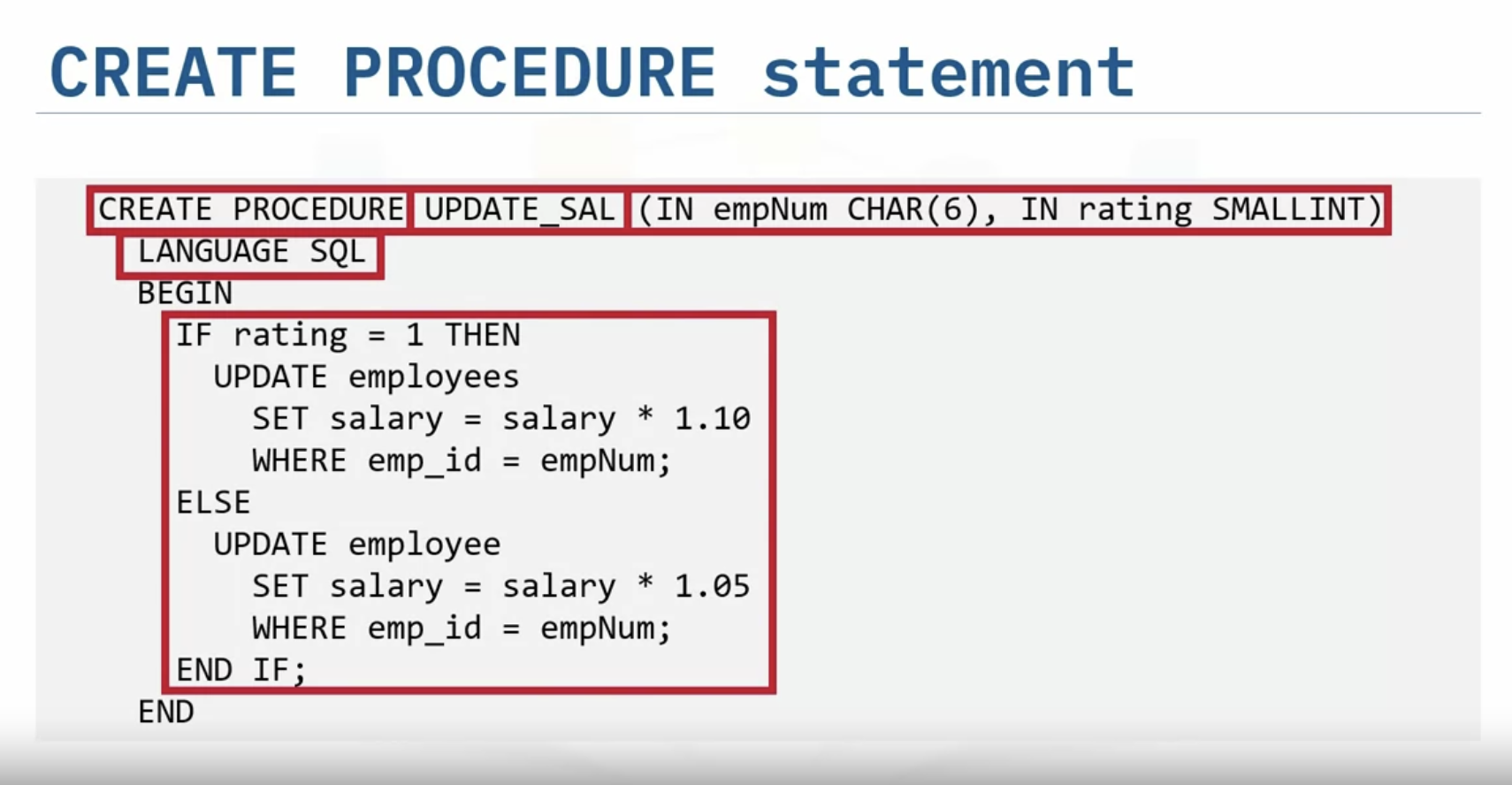
FROM table\_name

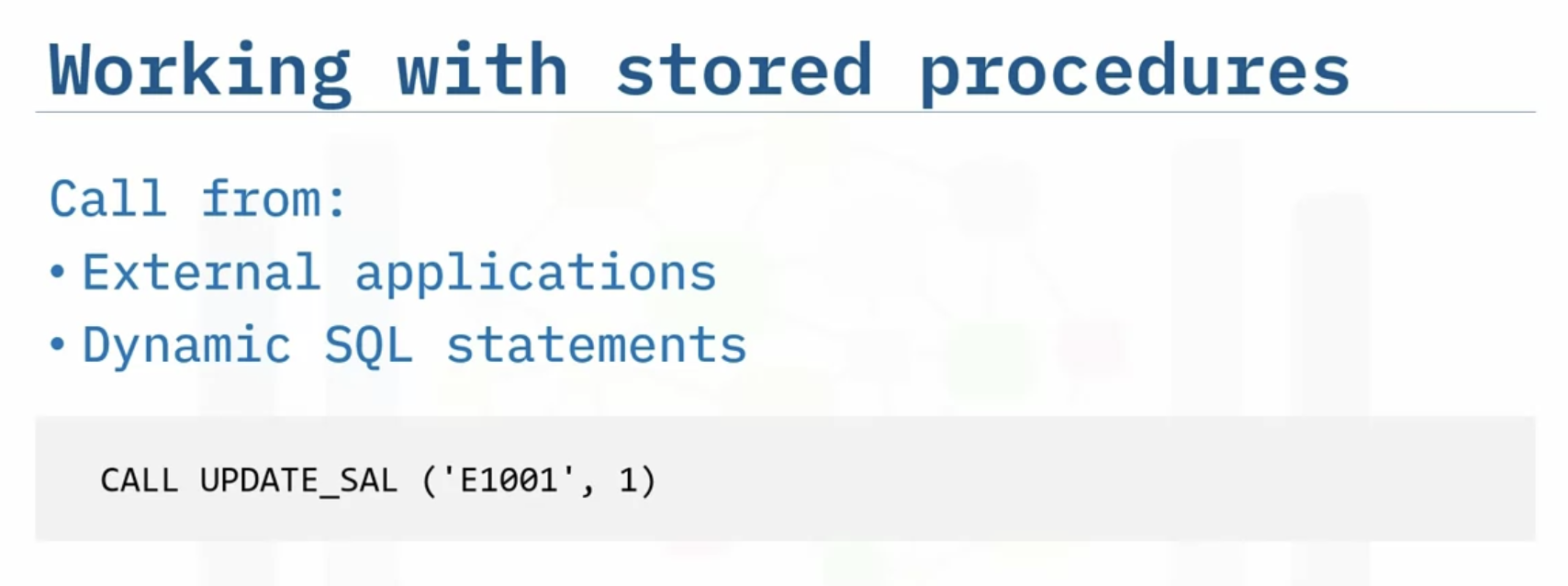
WHERE condition;

syntax of a DROP VIEW statement  
 DROP VIEW view\_name;

**Stored Procedures**

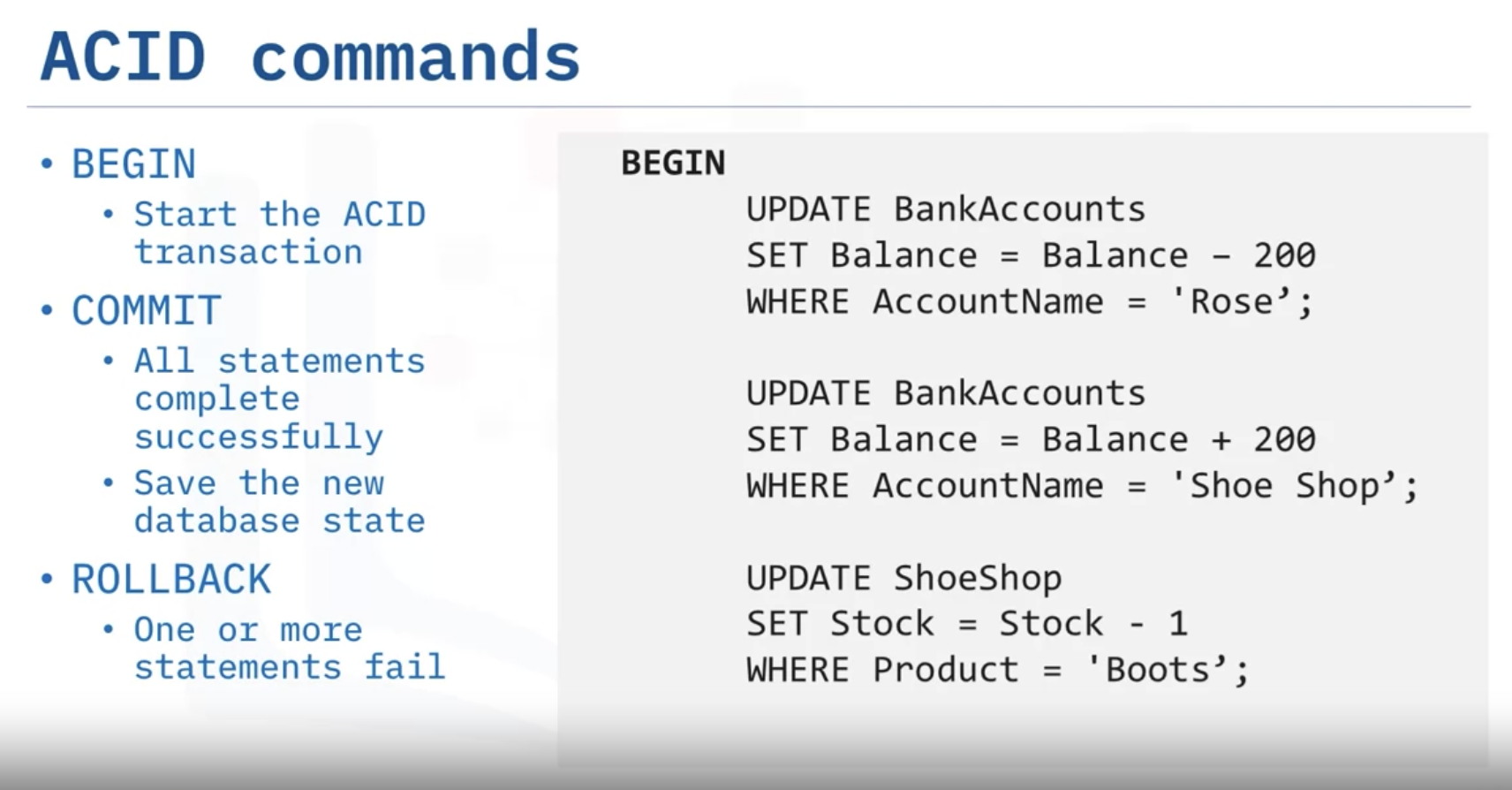
* Stored procedures are a set of SQL statements that execute on the server
* Stored procedures offer many benefits over sending SQL statements to the server like increase in security, improvement in performance, reuse of code etc.
* You can use stored procedures in dynamic SQL statements and external applications

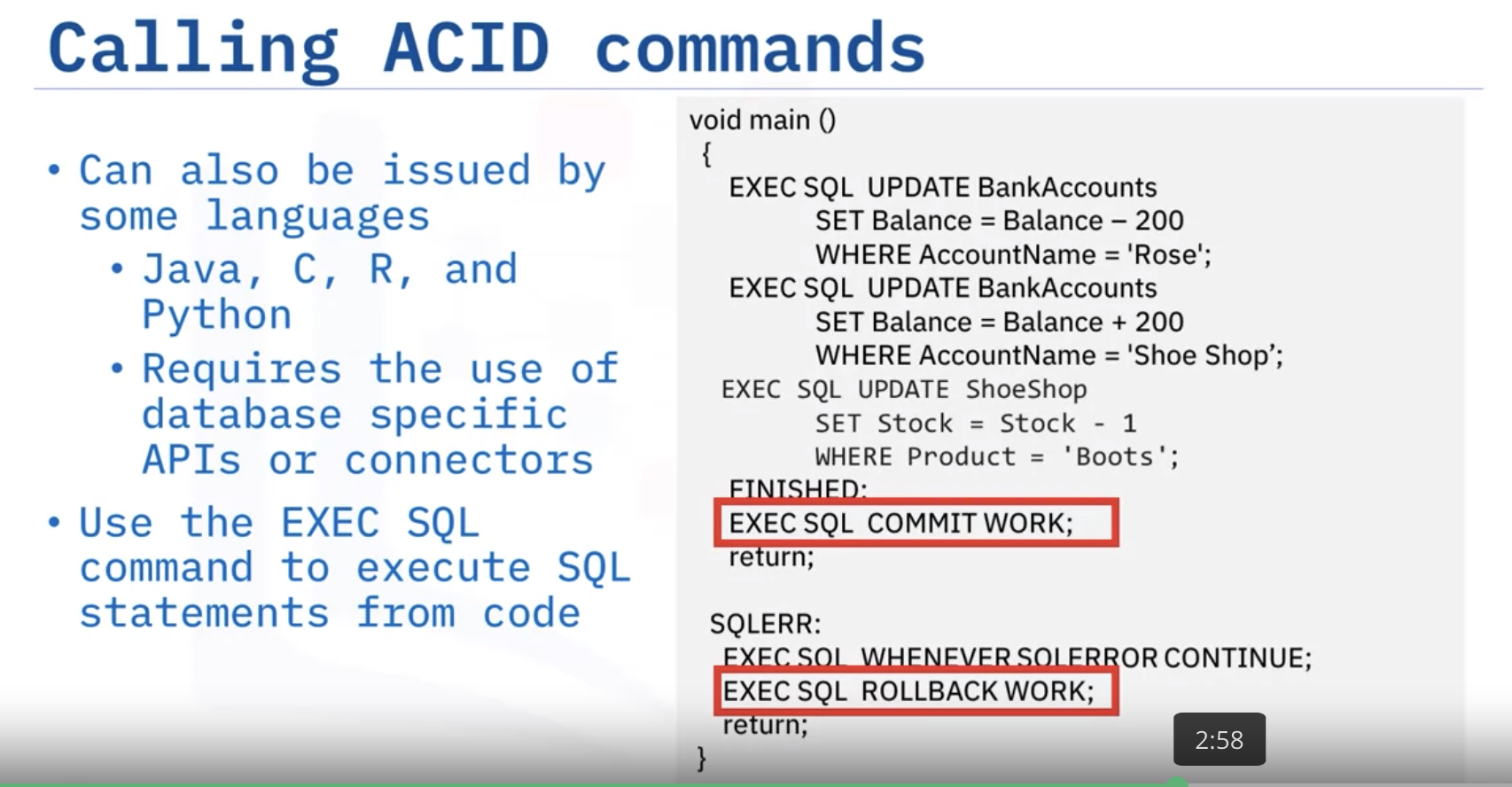




**ACID Transactions**

* A transaction represents a complete unit of work, which can be one or more SQL statements.
* An ACID transaction is one where all the SQL statements must complete successfully or none at all. This ensures the database is always in a consistent state.
* ACID stands for Atomic, Consistent, Isolated, Durable.
* SQL commands BEGIN, COMMIT, and ROLLBACK are used to manage ACID transactions.
* SQL commands can be called from languages like C, R and Python.





**Summary & Highlights**

* Views are a dynamic mechanism for presenting data from one or more tables.A transaction represents a complete unit of work, which can be one or more SQL statements.
* An ACID transaction is one where all the SQL statements must complete successfully, or none at all.
* A stored procedure is a set of SQL statements that are stored and executed on the database server, allowing you to send one statement as an alternative to sending multiple statements.
* You can write stored procedures in many different languages like SQL PL, PL/SQL, Java, and C.

<https://author-ide.skills.network/render?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZF9pbnN0cnVjdGlvbnNfdXJsIjoiaHR0cHM6Ly9jZi1jb3Vyc2VzLWRhdGEuczMudXMuY2xvdWQtb2JqZWN0LXN0b3JhZ2UuYXBwZG9tYWluLmNsb3VkL0lCTURldmVsb3BlclNraWxsc05ldHdvcmstREIwMjAxRU4tU2tpbGxzTmV0d29yay9sYWJzL0NoZWF0U2hlZXQvU1FMLUNoZWF0LVNoZWV0LUZvci1WaWV3cy1TdG9yZWQtUHJvY2VkdXJlcy1UcmFuc2FjdGlvbnMtYW5kLUpPSU5fU3RhdGVtZW50cy5tZCIsInRvb2xfdHlwZSI6Imluc3RydWN0aW9uYWwtbGFiIiwiYWRtaW4iOmZhbHNlLCJpYXQiOjE2NzY1NTExMTJ9.3HKJGz1dWhGfhHXtnK8GTzz4LxXNMwT5StHcDfdTk8s>

**JOIN Statements**

**Join Overview**

* You can use the JOIN operator to combine rows from two or more tables
* The tables being joined are related by a common column, which is usually the primary key of one table, and appears as a foreign key in the other table
* There are two types of joins; inner joins and outer joins.

**Inner Join**

* Inner joins return only the rows from the tables that have matching value in a common column, usually the primary key of one table that exists as a foreign key in the second table.
* Rows from joined tables that do not have a matching value do not appear in the result.

**Outer Join**

There are many varieties of outer join that you can use to refine your result set.

* Left outer joins return all rows from the left table, and all the rows form the right table that match that an inner join would return and all the rows in the first table that do not have a match in the second table.
* Right outer joins return all the rows that an inner join would return and all the rows in the second table that do not have a match in the first table.
* Full outer joins return all matching rows from both tables and all the rows from both tables that don’t have a match.

<https://stackoverflow.com/questions/2559194/difference-between-and-and-where-in-joins#:~:text=One%20way%20to%20look%20at,restrict%20the%20number%20of%20results>.

**Summary & Highlights**

* A join combines the rows from two or more tables based on a relationship between certain columns in these tables.
* To combine data from three or more different tables, you simply add new joins to the SQL statement.
* There are two types of table joins: inner join and outer join; and three types of outer joins: left outer join, right outer join, and full outer join.
* The most common type of join is the inner join, which matches the results from two tables and returns only the rows that match.
* You can use an alias as shorthand for a table or column name.
* You can use a self-join to compare rows within the same table.

<https://author-ide.skills.network/render?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZF9pbnN0cnVjdGlvbnNfdXJsIjoiaHR0cHM6Ly9jZi1jb3Vyc2VzLWRhdGEuczMudXMuY2xvdWQtb2JqZWN0LXN0b3JhZ2UuYXBwZG9tYWluLmNsb3VkL0lCTURldmVsb3BlclNraWxsc05ldHdvcmstREIwMjAxRU4tU2tpbGxzTmV0d29yay9sYWJzL0NoZWF0U2hlZXQvU1FMLUNoZWF0LVNoZWV0LUpPSU5fU3RhdGVtZW50cy5tZCIsInRvb2xfdHlwZSI6InRoZWlhIiwiYWRtaW4iOmZhbHNlLCJpYXQiOjE2ODMyNTk3MzV9.vUsKugs8KUZM87jr_jqFTcgDd_UcOGAVXddl_PSd-v4>

<https://www.coursera.org/learn/sql-data-science/ungradedWidget/j9W07/sql-cheat-sheet-views-stored-procedures-transactions-and-join-statements>