

**Create a pdf which has detailed understanding of SQL with code examples. Topics to include are:**

SQL Basics: SELECT, WHERE, ORDER BY, GROUP BY, HAVING Joins: INNER, LEFT, RIGHT, FULL OUTER Subqueries & Nested Queries CTEs and Window Functions (RANK, ROW\_NUMBER, etc.) SQL for EDA: Case When, Aggregations Indexing and Query Optimization Using SQL with Pandas (via sqlite3 or SQLAlchemy

Certainly! Below is a comprehensive outline for a PDF guide that covers SQL fundamentals with **detailed explanations** and practical code examples, as per your topics list. You can use this format to create a PDF document.

**SQL: A Practical Guide with Code Examples**

**1. SQL Basics**

**SELECT, WHERE, ORDER BY, GROUP BY, HAVING**

* **SELECT** retrieves specific columns:

SELECT name, age FROM employees;

* **WHERE** filters rows before grouping:

SELECT \* FROM employees WHERE department = 'HR';

* **ORDER BY** sorts the output:

SELECT name, salary FROM employees ORDER BY salary DESC;

* **GROUP BY** groups rows with the same values for aggregate functions:

SELECT department, COUNT(\*) FROM employees GROUP BY department;

* **HAVING** filters groups, not individual rows:

SELECT department, COUNT(\*) as count  
FROM employees  
GROUP BY department  
HAVING count > 5;

**Execution order:** WHERE filters rows; GROUP BY forms groups; HAVING filters groups; ORDER BY sorts results[[1]](#fn1)[[2]](#fn2).

**2. Joins: INNER, LEFT, RIGHT, FULL OUTER**

* **INNER JOIN**: Returns rows with matching values in both tables.

SELECT employees.name, departments.dept\_name  
FROM employees  
INNER JOIN departments  
 ON employees.dept\_id = departments.id;

* **LEFT JOIN**: All left table rows and matched right table rows (NULL if no match).

SELECT e.name, d.dept\_name  
FROM employees e  
LEFT JOIN departments d  
 ON e.dept\_id = d.id;

* **RIGHT JOIN**: All right table rows and matched left table rows (NULL if no match).

SELECT e.name, d.dept\_name  
FROM employees e  
RIGHT JOIN departments d  
 ON e.dept\_id = d.id;

* **FULL OUTER JOIN**: All rows where there is a match in either table; NULLs where there is no match[[3]](#fn3)[[4]](#fn4).

SELECT e.name, d.dept\_name  
FROM employees e  
FULL OUTER JOIN departments d  
 ON e.dept\_id = d.id;

**3. Subqueries & Nested Queries**

* **Subquery**: Query within a query.

SELECT name FROM employees  
WHERE department\_id IN (  
 SELECT id FROM departments WHERE location = 'NY'  
);

* **Correlated Subquery**: Uses data from the outer query.

SELECT name  
FROM employees e  
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id);

* **Types**: Non-correlated (runs independently), Correlated (depends on outer query)[[5]](#fn5)[[6]](#fn6).

**4. CTEs and Window Functions**

* **CTE (Common Table Expression):**

WITH high\_salary AS (  
 SELECT name, salary FROM employees WHERE salary > 50000  
)  
SELECT \* FROM high\_salary;

* **ROW\_NUMBER, RANK, DENSE\_RANK**:

SELECT name, salary,  
 ROW\_NUMBER() OVER (ORDER BY salary DESC) AS row\_num,  
 RANK() OVER (ORDER BY salary DESC) AS rank,  
 DENSE\_RANK() OVER (ORDER BY salary DESC) AS dense\_rank  
FROM employees;

With **PARTITION BY** (e.g., ranking within departments):

SELECT name, department,  
 ROW\_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS row\_num  
FROM employees;

**5. SQL for EDA: CASE WHEN & Aggregations**

* **CASE WHEN** for conditional logic:

SELECT name,  
 CASE  
 WHEN salary >= 60000 THEN 'Highly Paid'  
 WHEN salary >= 40000 THEN 'Average'  
 ELSE 'Low'  
 END AS salary\_bracket  
FROM employees;

* **Aggregations**: COUNT, SUM, AVG, MIN, MAX

SELECT department, AVG(salary)  
FROM employees  
GROUP BY department;

* **Combining CASE with aggregations**:

SELECT department,  
 COUNT(CASE WHEN salary > 50000 THEN 1 END) AS high\_earners  
FROM employees  
GROUP BY department;

**6. Indexing & Query Optimization**

* **Creating an Index:**

CREATE INDEX idx\_salary ON employees(salary);

* **Index Usage**: Indexes help speed up WHERE, ORDER BY, and JOINs where indexed columns are involved.
* **Best Practices**:
  + Index columns that appear in WHERE, JOIN, or ORDER BY clauses.
  + Avoid over-indexing (can slow down updates/inserts).
* **Example for Query Optimization:**

SELECT \* FROM employees WHERE salary BETWEEN 60000 AND 90000;

This. uses an index on salary for fast range search[[7]](#fn7)[[8]](#fn8).

**7. Using SQL with Pandas (sqlite3, SQLAlchemy)**

**Read from SQL:**

import pandas as pd  
import sqlite3  
  
conn = sqlite3.connect('database.db')  
df = pd.read\_sql\_query("SELECT \* FROM employees", conn)  
conn.close()

**Write DataFrame to SQL:**

from sqlalchemy import create\_engine  
engine = create\_engine('sqlite:///database.db')  
df.to\_sql('employees', engine, if\_exists='replace')

**Execute SQL via SQLAlchemy:**

from sqlalchemy import text  
with engine.connect() as conn:  
 result = conn.execute(text("SELECT \* FROM employees"))  
 for row in result:  
 print(row)

**This guide gives a hands-on approach for quickly mastering SQL for analytics and advanced queries. Each section may be expanded with more code samples and exercises, as needed for the PDF.**

If you need this content formatted and exported as an actual PDF, you can use tools like Pandoc, LaTeX, or any Markdown to PDF converter—just copy the content above into your preferred tool and "Export as PDF."

⁂

1. <https://www.geekster.in/articles/group-by-having-clause-in-sql/>

1. <https://www.digitalocean.com/community/tutorials/how-to-use-groupby-and-orderby-in-sql>

1. <https://www.w3schools.com/sql/sql_join.asp>

1. <https://datalemur.com/sql-tutorial/sql-joins-inner-outer-left-right>

1. <https://www.scaler.com/topics/nested-sql-query/>

1. <https://www.tutorialspoint.com/nested-queries-in-sql>

1. <https://docs.oracle.com/en/database/other-databases/nosql-database/20.3/sqlreferencefornosql/examples-query-optimization.html>

1. <https://towardsdatascience.com/indexing-for-sql-query-optimization-139b57db9fc6/>