SQL Complete Study Material - Zero to Hero

Chapter 1: Introduction to Databases and SQL

What is a Database?

A database is a structured collection of data stored electronically. Think of it as a digital filing system where information is organized in tables.

Real-world example: A school database might contain:

- Student information (names, IDs, grades)
- Course information (course names, credits, professors)
- Enrollment records (which students are in which courses)

What is SQL?

SQL (Structured Query Language) is the standard language for:

- **Querying** data (asking questions)
- **Inserting** new data
- **Updating** existing data
- **Deleting** unwanted data
- **Creating** database structures

Database Terminology

Table: A collection of related data organized in rows and columns

employees table:

id	name	position	salary
1	Jhon	Developer	70000
2	Sarah	Manager	85000

Row (Record): A single entry in a table (like John's complete information) **Column (Field):** A specific attribute (like "name" or "salary") **Primary Key:** A unique identifier for each row (like employee ID)

Chapter 2: Basic SELECT Statements

The SELECT Statement Structure

SELECT column_name(s)

FROM table_name;

Examples with Explanations

Example 1: Select all columns

SELECT * FROM employees;

- `*` means "all columns"
- This returns every piece of information about every employee

Example 2: Select specific columns

SELECT first_name, salary FROM employees;

- Only shows names and salaries
- Useful when you don't need all information

Example 3: Select with aliases (renaming columns)

SELECT first_name AS "Employee Name", salary AS "Monthly Pay"

FROM employees;

- `AS` creates a temporary name for display
- Makes output more readable

Practice Exercises - Chapter 2

1. Write a query to show all information about customers
2. Show only customer names and cities
3. Display product names with a more readable column header
4. Select employee emails and hire dates

Chapter 3: Filtering Data with WHERE

The WHERE Clause

Used to filter rows based on conditions.

Basic Syntax:

SELECT column_name(s)

FROM table_name

WHERE condition;

Comparison Operators

- `=` Equal to
- `!=` or `<>` Not equal to
- `>` Greater than
- `<` Less than
- `>=` Greater than or equal
- `<=` Less than or equal

Examples with Explanations

Example 1: Exact match

SELECT * FROM employees WHERE department = 'Sales';

Explanation: Shows all employees who work in the Sales department

Example 2: Numeric comparison

SELECT name, salary FROM employees WHERE salary > 50000;

Explanation: Shows names and salaries of employees earning more than $50,000

Example 3: Text patterns with LIKE

SELECT * FROM customers WHERE first_name LIKE 'J%';

Explanation:

- `LIKE` is used for pattern matching
- `%` means "any characters"
- `J%` means "starts with J"
- `%son` would mean "ends with son"
- `%mid%` would mean "contains mid"

Logical Operators

AND - Both conditions must be true

SELECT * FROM employees

WHERE department = 'IT' AND salary > 60000;

OR - At least one condition must be true

SELECT * FROM customers

WHERE country = 'USA' OR country = 'Canada';

NOT - Opposite of the condition

SELECT * FROM orders WHERE NOT status = 'Cancelled';

IN - Match any value in a list

SELECT * FROM customers WHERE country IN ('USA', 'UK', 'Canada');

BETWEEN - Within a range

SELECT * FROM products WHERE price BETWEEN 10 AND 100;

Practice Exercises - Chapter 3

1. Find all employees hired after 2020
2. Show customers from either 'USA' or 'UK'
3. Display products with price less than $50
4. Find employees whose email contains 'gmail'
5. Show orders with amount between $100 and $500

Chapter 4: Sorting and Limiting Results

ORDER BY Clause

Used to sort results in ascending or descending order.

Syntax:

SELECT column_name(s)

FROM table_name

ORDER BY column_name ASC|DESC;

Examples

Example 1: Sort by salary (highest first)

SELECT name, salary FROM employees

ORDER BY salary DESC;

Example 2: Sort by multiple columns

SELECT * FROM employees

ORDER BY department ASC, salary DESC;

Explanation: First sorts by department A-Z, then within each department, sorts by salary highest to lowest

LIMIT Clause

Used to restrict the number of rows returned.

Example: Top 5 highest-paid employees

SELECT name, salary FROM employees

ORDER BY salary DESC

LIMIT 5;

Practice Exercises - Chapter 4

1. Show all products sorted by price (cheapest first)
2. Display top 3 most recent orders
3. List customers alphabetically by last name
4. Show 10 most expensive products

Chapter 5: Aggregate Functions

What are Aggregate Functions?

Functions that perform calculations on multiple rows and return a single result.

Common Aggregate Functions

COUNT() - Counts rows

```
SELECT COUNT(*) FROM employees;

-- Result: Total number of employees


SELECT COUNT(DISTINCT department) FROM employees;

-- Result: Number of different departments
```

SUM() - Adds up values

```
SELECT SUM(salary) FROM employees;

-- Result: Total of all salaries


SELECT SUM(quantity) FROM order_items;

-- Result: Total items sold
```

AVG() - Calculates average

```
SELECT AVG(salary) FROM employees;

-- Result: Average salary


SELECT AVG(price) FROM products;

-- Result: Average product price
```

MAX() and MIN() - Find highest and lowest values

SELECT MAX(salary), MIN(salary) FROM employees;

-- Result: Highest and lowest salaries

SELECT MAX(order_date) FROM orders;

-- Result: Most recent order date

Combining Aggregates with WHERE

SELECT AVG(salary) FROM employees

WHERE department = 'IT';

-- Average salary in IT department only

Practice Exercises - Chapter 5

1. Count total number of customers
2. Find the most expensive product
3. Calculate average order amount
4. Sum all product stock quantities
5. Find earliest employee hire date

Chapter 6: GROUP BY and HAVING

GROUP BY Clause

Groups rows with the same values and allows aggregate functions on each group.

Example: Average salary by department

SELECT department, AVG(salary) as avg_salary

FROM employees

GROUP BY department;

Result:

department	avg_salary
sales	65000
it	78000
HR	55000

More GROUP BY Examples

Example: Count employees by department

SELECT department, COUNT(*) as employee_count

FROM employees

GROUP BY department;

Example: Total sales by customer

SELECT customer_id, SUM(total_amount) as total_spent

FROM orders

GROUP BY customer_id;

HAVING Clause

Used to filter groups (like WHERE but for groups).

Example: Departments with more than 5 employees

```sql
SELECT department, COUNT(*) as employee_count

FROM employees

GROUP BY department

HAVING COUNT(*) > 5;
```

WHERE vs HAVING:

- **WHERE** filters individual rows before grouping
- **HAVING** filters groups after grouping

Practice Exercises - Chapter 6

1. Count orders by status
2. Average product price by category
3. Find customers who spent more than $1000 total
4. Show departments with average salary > $60000

Chapter 7: JOINs - Combining Tables

Why JOINs?

Real-world data is spread across multiple related tables. JOINs combine data from these tables.

Types of JOINs

INNER JOIN

Returns rows that have matching values in both tables.

Example: Employee names with department names

SELECT e.first_name, e.last_name, d.department_name

FROM employees e

INNER JOIN departments d ON e.department_id = d.department_id;

Explanation:

- `e` and `d` are table aliases (shortcuts)
- `ON` specifies how tables are related
- Only employees with valid departments are shown

LEFT JOIN

Returns all rows from the left table, plus matched rows from the right table.

Example: All employees with their departments (even if no department)

SELECT e.first_name, d.department_name

FROM employees e

LEFT JOIN departments d ON e.department_id = d.department_id;

RIGHT JOIN

Returns all rows from the right table, plus matched rows from the left table.

Example: All departments with their employees (even empty departments)

```sql
SELECT e.first_name, d.department_name

FROM employees e

RIGHT JOIN departments d ON e.department_id = d.department_id;
```

Complex JOIN Example

Orders with customer and product information:

```sql
SELECT

    c.first_name + ' ' + c.last_name as customer_name,

    p.product_name,

    oi.quantity,

    o.order_date

FROM orders o

INNER JOIN customers c ON o.customer_id = c.customer_id

INNER JOIN order_items oi ON o.order_id = oi.order_id

INNER JOIN products p ON oi.product_id = p.product_id;
```

Practice Exercises - Chapter 7

1. Show employee names with their manager names
2. List all orders with customer information
3. Display products with their order quantities
4. Find customers who haven't placed any orders

Chapter 8: Subqueries

What is a Subquery?

A query inside another query. Used when you need the result of one query to help with another.

Simple Subquery Example

Find employees earning more than average:

SELECT first_name, last_name, salary

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

Step-by-step explanation:

1. Inner query calculates: `SELECT AVG(salary) FROM employees` → Result: 70000
2. Outer query becomes: `WHERE salary > 70000`
3. Shows employees with salary > 70000

Subquery with IN

Find customers who have placed orders:

SELECT first_name, last_name

FROM customers

WHERE customer_id IN (SELECT customer_id FROM orders);

Practice Exercises - Chapter 8

1. Find products more expensive than average price
2. Show customers who placed orders in 2023
3. Find employees in departments with more than 3 people

Chapter 9: Advanced Features

CASE Statements

Used for conditional logic (like if-else).

Example: Categorize employees by salary

```
SELECT
    first_name,
    salary,
    CASE
        WHEN salary < 50000 THEN 'Low'
        WHEN salary < 80000 THEN 'Medium'
        ELSE 'High'
    END as salary_category
FROM employees;
```

UNION

Combines results from multiple SELECT statements.

Example: All email addresses from customers and employees

```
SELECT email FROM customers
UNION
SELECT email FROM employees;
```

Window Functions (Advanced)

Perform calculations across related rows.

Example: Rank employees by salary within department

```
SELECT
```

```sql
    first_name,
    department_id,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) as salary_rank
FROM employees;
```

Chapter 10: Data Modification

INSERT - Adding New Data

-- Insert single row

INSERT INTO customers (first_name, last_name, email)

VALUES ('John', 'Doe', 'john@email.com');

-- Insert multiple rows

INSERT INTO products (product_name, price, category) VALUES

('Laptop', 999.99, 'Electronics'),

('Mouse', 19.99, 'Electronics');

UPDATE - Modifying Existing Data

-- Update single record

UPDATE employees

SET salary = 75000

WHERE employee_id = 101;

-- Update multiple records

UPDATE products

SET price = price * 1.1

WHERE category = 'Electronics';

DELETE - Removing Data

-- Delete specific records

DELETE FROM orders

WHERE status = 'Cancelled';

-- Delete all records (be careful!)

DELETE FROM temp_table;

Chapter 11: Database Design and Normalization

What is Database Design?

Database design is the process of organizing data efficiently and logically. Poor design leads to data redundancy, inconsistency, and performance issues.

Database Normalization

Normalization is the process of organizing data to reduce redundancy and improve data integrity.

First Normal Form (1NF)

Rule: Each column should contain atomic (indivisible) values, and each row should be unique.

Bad Example (Not 1NF):

students table:

Id	Name	subject
1	Jhon	Maths, Science
2	Jane	English, History

Good Example (1NF):

students table:

Id	**Name**
1	Jhon
2	Jane

student_subjects table:

Id	Student_id	Subject
1	1	Math
2	1	Science
3	2	English
4	2	History

Second Normal Form (2NF)

Rule: Must be in 1NF, and all non-key columns must depend on the entire primary key.

Example: In an order_details table with composite primary key (order_id, product_id), customer_name shouldn't be there because it only depends on order_id, not the full key.

Third Normal Form (3NF)

Rule: Must be in 2NF, and no non-key column should depend on another non-key column.

Bad Example:

employees table:

Id	Name	Dept	Dept_location	Salary
1	Jhon	Sales	New York	50000
2	Jane	Sales	New York	55000

Problem: dept_location depends on dept, not on employee_id

Good Example (3NF):

employees table:

ID	Name	dept_id	Salary
1	Jhon	1	50000
2	Jane	1	55000

departments table:

Dept_id	Dept_location
1	New York
2	London

Entity Relationship (ER) Diagrams

Visual representation of database structure showing:

- **Entities** (tables) - rectangles
- **Attributes** (columns) - ovals
- **Relationships** - diamonds

Types of Relationships

1. **One-to-One (1:1):** Each record in table A relates to exactly one record in table B
    - Example: Employee ↔ Employee_Details
2. **One-to-Many (1:M):** One record in table A can relate to many records in table B
    - Example: Department → Employees
3. **Many-to-Many (M:M):** Records in both tables can relate to multiple records in the other
    - Example: Students ↔ Courses (requires junction table)

Chapter 12: Constraints and Data Integrity

Primary Key Constraints

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

Foreign Key Constraints

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Check Constraints

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    price DECIMAL(10,2) CHECK (price > 0),
    stock_quantity INT CHECK (stock_quantity >= 0)
);
```

Unique Constraints

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(255) UNIQUE,
```

```
    username VARCHAR(50) UNIQUE
);
```

NOT NULL Constraints

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE
);
```

Chapter 13: Indexes and Performance

What are Indexes?

Indexes are database objects that improve query performance by creating shortcuts to data.

Analogy: Like an index in a book - instead of reading every page to find "SQL", you look in the index and jump directly to page 245.

Creating Indexes

```
-- Create index on frequently searched column
CREATE INDEX idx_customer_email ON customers(email);


-- Composite index for multiple columns
CREATE INDEX idx_order_date_status ON orders(order_date, status);


-- Unique index
CREATE UNIQUE INDEX idx_product_code ON products(product_code);
```

When to Use Indexes

Good for:

- Columns frequently used in WHERE clauses
- Columns used in JOIN conditions
- Columns used in ORDER BY

Avoid indexing:

- Small tables (< 1000 rows)
- Columns that change frequently
- Tables with heavy INSERT/UPDATE activity

Query Performance Tips

```
-- Good: Uses index

SELECT * FROM customers WHERE customer_id = 123;



-- Bad: Function prevents index usage

SELECT * FROM customers WHERE UPPER(email) = 'JOHN@EMAIL.COM';



-- Good: Index-friendly

SELECT * FROM customers WHERE email = 'john@email.com';
```

Chapter 14: Views and Stored Procedures

Views

A view is a virtual table based on a query. It doesn't store data but shows results dynamically.

```
-- Create a view

CREATE VIEW employee_summary AS

SELECT

    e.first_name + ' ' + e.last_name as full_name,

    d.department_name,

    e.salary

FROM employees e

JOIN departments d ON e.department_id = d.department_id;



-- Use the view like a table

SELECT * FROM employee_summary WHERE salary > 60000;
```

Benefits of Views:

- Simplify complex queries
- Security (hide sensitive columns)
- Consistent interface for applications

Stored Procedures

Reusable SQL code blocks that can accept parameters.

```
-- Create stored procedure

CREATE PROCEDURE GetEmployeesByDepartment(IN dept_name VARCHAR(50))

BEGIN

    SELECT e.first_name, e.last_name, e.salary

    FROM employees e
```

```
    JOIN departments d ON e.department_id = d.department_id
    WHERE d.department_name = dept_name;
END;

-- Execute stored procedure
CALL GetEmployeesByDepartment('Sales');
```

Chapter 15: Transactions and ACID Properties

What are Transactions?

A transaction is a sequence of SQL operations treated as a single unit of work.

```
-- Start transaction
BEGIN TRANSACTION;

-- Multiple operations
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

-- Commit if everything succeeds
COMMIT;

-- Or rollback if something fails
-- ROLLBACK;
```

ACID Properties

A - Atomicity: All operations succeed or all fail **C - Consistency:** Database remains in valid state **I - Isolation:** Concurrent transactions don't interfere **D - Durability:** Committed changes are permanent

---

Chapter 16: Common Table Expressions (CTEs)

What are CTEs?

Temporary named result sets that exist only during query execution.

```
-- Simple CTE
WITH sales_summary AS (
    SELECT
        customer_id,
        SUM(total_amount) as total_sales
    FROM orders
    GROUP BY customer_id
)
SELECT
    c.first_name,
    c.last_name,
    s.total_sales
FROM customers c
JOIN sales_summary s ON c.customer_id = s.customer_id
WHERE s.total_sales > 1000;
```

Recursive CTEs

Used for hierarchical data like organizational charts.

```
-- Find all employees under a manager
WITH RECURSIVE employee_hierarchy AS (
    -- Base case: Start with the manager
    SELECT employee_id, first_name, manager_id, 0 as level
    FROM employees
```

```sql
    WHERE employee_id = 101

    UNION ALL

    -- Recursive case: Find direct reports
    SELECT e.employee_id, e.first_name, e.manager_id, eh.level + 1
    FROM employees e
    JOIN employee_hierarchy eh ON e.manager_id = eh.employee_id
)
SELECT * FROM employee_hierarchy;
```

Chapter 17: JSON and Modern SQL Features

Working with JSON Data

Many modern databases support JSON data types.

```
-- PostgreSQL JSON examples

CREATE TABLE products (

    id INT PRIMARY KEY,

    name VARCHAR(100),

    attributes JSON

);


-- Insert JSON data

INSERT INTO products VALUES

(1, 'Laptop', '{"brand": "Apple", "ram": "16GB", "storage": "512GB"}');


-- Query JSON data

SELECT name, attributes->>'brand' as brand

FROM products

WHERE attributes->>'ram' = '16GB';
```

Window Functions (Advanced Analytics)

Perform calculations across related rows without grouping.

```
-- Running total of sales

SELECT

    order_date,

    total_amount,

    SUM(total_amount) OVER (ORDER BY order_date) as running_total
```

```
FROM orders;

-- Rank within groups
SELECT
    department_id,
    first_name,
    salary,
    RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) as dept_rank
FROM employees;
```

Important Concepts Summary

Data Types to Know

- **Numeric:** INT, DECIMAL(10,2), FLOAT
- **Text:** VARCHAR(50), TEXT, CHAR(10)
- **Date/Time:** DATE, DATETIME, TIMESTAMP
- **Boolean:** BOOLEAN (TRUE/FALSE)
- **Modern:** JSON, XML, ARRAY

SQL Standards and Dialects

- **ANSI SQL:** Standard that all databases follow
- **PostgreSQL:** Advanced features, strong JSON support
- **MySQL:** Popular for web applications
- **SQL Server:** Microsoft's enterprise solution
- **Oracle:** Enterprise-grade with advanced features

Security Best Practices

1. **Use parameterized queries** to prevent SQL injection
2. **Grant minimum necessary permissions** to users
3. **Encrypt sensitive data** at rest and in transit
4. **Regular backups** and disaster recovery plans
5. **Audit database access** and changes