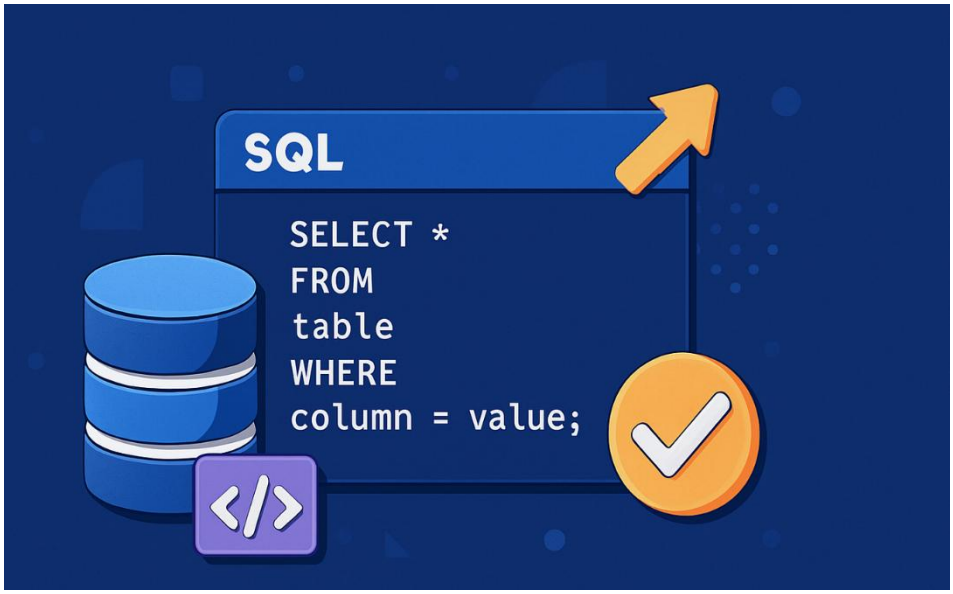
SQL Fundamentals

From Databases to Queries - A Complete Cybersecurity Guide



Introduction: Why SQL Matters in Cybersecurity

Databases are everywhere. Web applications? Databases. Security systems? Databases. User authentication? Databases. SIEM tools? Databases. Whether you're exploiting SQL injection vulnerabilities, investigating security incidents, analyzing logs, or building secure applications, you'll encounter databases.

Offensive perspective: SQL helps you understand injection attacks, extract sensitive data from compromised systems, and manipulate database queries.

Defensive perspective: SQL enables threat hunting through logs, implementing access controls, detecting suspicious patterns, and securing database configurations.

Part 1: Understanding Databases

What is a Database?

An organized collection of structured information that's easily accessible, manipulatable, and analyzable. Examples:

- User authentication (usernames, passwords, sessions)
- Social media content (posts, comments, likes, shares)
- E-commerce transactions (orders, inventory, payments)
- Streaming recommendations (watch history, preferences)

Two Primary Database Types

Relational Databases (SQL):

- Structured data in tables with rows and columns
- Strict schema enforced (data must match defined structure)
- Relationships between tables (user → orders → products)
- ACID compliance (Atomicity, Consistency, Isolation, Durability)
- Best for: Financial transactions, user authentication, e-commerce

Example structure:

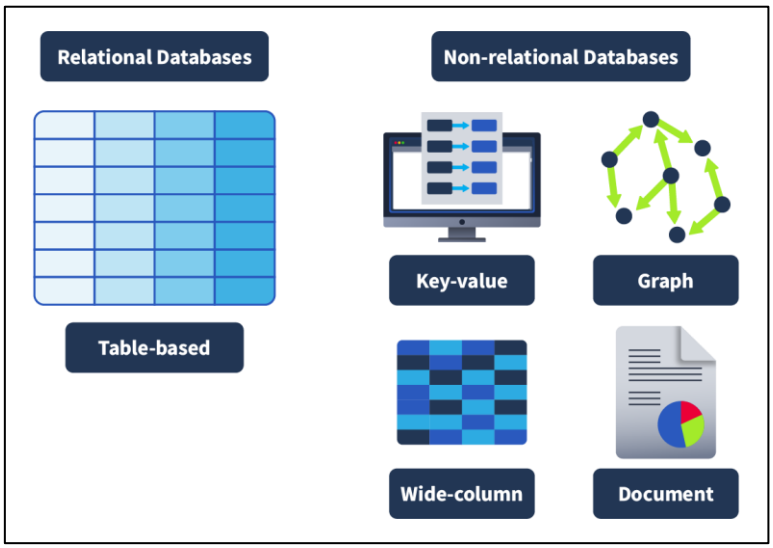
```
first_name | last_name | email | username | password_hash
```

Non-Relational Databases (NoSQL):

- Flexible, non-tabular format (documents, key-value, graphs)
- No strict schema (data structure can vary)
- Highly scalable horizontally
- Best for: Social media, real-time analytics, IoT data

Example (MongoDB document):

```
{
        _id: ObjectId("123abc"),
        name: { first: "Thomas", last: "Anderson" },
        occupation: ["The One"],
        skills: ["kung-fu", "bullet-dodging"]
}
```



Relational Database Structure

Tables - Collections of related data (e.g., 'Books', 'Users', 'Orders')

Columns - Data fields with specific types (id, name, email, created_date)

Rows - Individual records (one book, one user, one order)

Common data types:

`INT` - Whole numbers (1, 42, -5)

`VARCHAR(n)` - Text strings up to n characters

`DATE` - Dates (2024-01-27)

`DECIMAL/FLOAT` - Numbers with decimals (19.99)

`BOOLEAN` - True/False

Primary and Foreign Keys

Primary Key:

• Uniquely identifies each record

• Cannot be NULL or duplicate

• Example: student_id, order_id, isbn

• Only ONE per table

Books table:

```
book_id (PK) | title | author_id (FK)
```

Foreign Key:

• Links tables together (creates relationships)

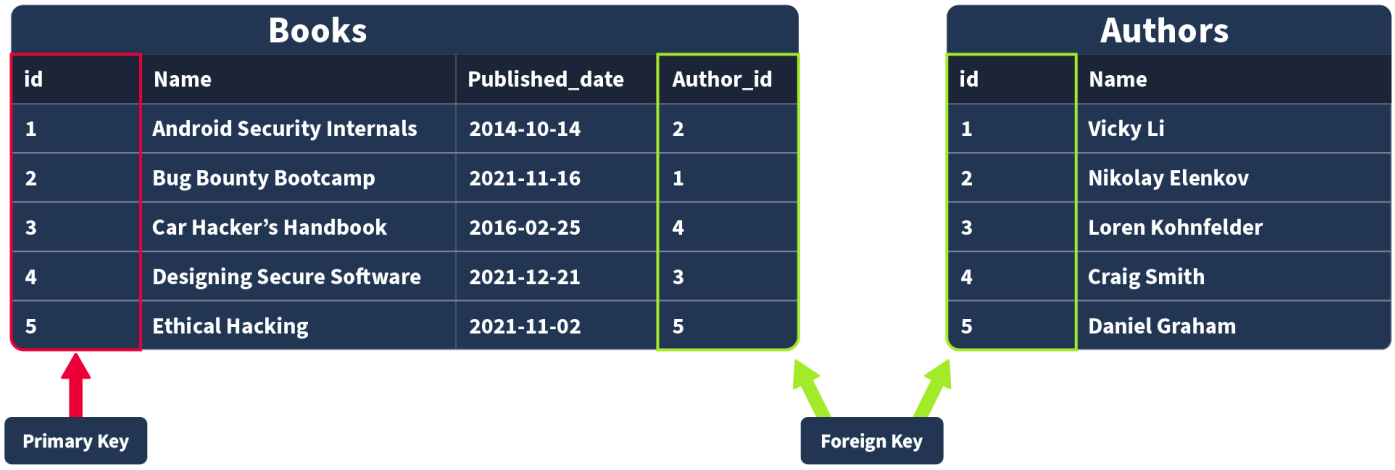
• References primary key in another table

• Multiple allowed per table

Example relationship:

```
Books.author_id → Authors.id
```

This links each book to its author!



Part 2: SQL - The Language of Databases

What is SQL?

SQL (Structured Query Language) is the programming language used to communicate with relational databases. It's managed through Database Management Systems (DBMS):

• MySQL - Most popular open-source

• PostgreSQL - Advanced features

• Microsoft SQL Server - Enterprise

• MariaDB - MySQL fork

• SQLite - Embedded, file-based

Why Learn SQL?

Fast: Query millions of records in milliseconds

Easy to Learn: Reads like plain English (SELECT, WHERE, FROM)

Reliable: Strict structure ensures data accuracy

Flexible: Powerful querying and analysis capabilities

Part 3: Database and Table Statements

Database Operations

CREATE DATABASE - Create new database

```
CREATE DATABASE thm_bookstore;
```

SHOW DATABASES - List all databases

```
SHOW DATABASES;
```

USE - Select active database

```
USE thm_bookstore;
```

DROP DATABASE - Delete database (PERMANENT!)

```
DROP DATABASE old_database;
```

Table Operations

CREATE TABLE

```
CREATE TABLE books (

book_id INT AUTO_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

author VARCHAR(100),

publication_date DATE,   price DECIMAL(10,2) );
```

Breakdown:

• book_id: Integer, auto-increments, unique identifier

• title: Text up to 255 characters, required (NOT NULL)

• publication_date: Date format

• price: Decimal with 2 decimal places

SHOW TABLES

```
SHOW TABLES;
```

DESCRIBE / DESC

```
DESCRIBE books;
```

Shows column names, data types, nullability, keys

ALTER TABLE

```
ALTER TABLE books ADD page_count INT;
```

Adds new column to existing table

DROP TABLE

```
DROP TABLE old_table;
```

Part 4: CRUD Operations - Managing Data

CRUD = Create, Read, Update, Delete (fundamental database operations)

01. CREATE - INSERT Statement

```
INSERT INTO books (title, author, publication_date, price)
        VALUES ('Hacking: The Art of Exploitation', 'Jon Erickson', '2008-02-07', 49.95);
```

Result: New row added to books table

02. READ - SELECT Statement

`SELECT * FROM books;`

→ Returns ALL columns from ALL rows

`SELECT title, price FROM books;`

→ Returns only title and price columns

03.UPDATE - Modify Existing Data

`UPDATE books SET price = 39.95 WHERE book_id = 1;`

Warning: Always use WHERE! Without it, ALL rows get updated!

04.DELETE - Remove Data

`DELETE FROM books WHERE book_id = 1;`

CRITICAL: `DELETE FROM books;` (no WHERE) deletes EVERYTHING!

Part 5: SQL Clauses - Filtering and Organizing

01. WHERE - Filter Results

```
SELECT * FROM books WHERE price > 30;
```

02. DISTINCT - Remove Duplicates

```
SELECT DISTINCT author FROM books;
```

03. ORDER BY - Sort Results

```
SELECT * FROM books ORDER BY price DESC;
```

ASC = ascending (1, 2, 3), DESC = descending (3, 2, 1)

04. GROUP BY - Aggregate Data

```
SELECT category, COUNT(*) FROM books GROUP BY category;
```

05. HAVING - Filter After Aggregation

```
SELECT category, COUNT(*) FROM books GROUP BY category HAVING COUNT(*) > 2;
```

Part 6: Operators - Logic and Comparison

Comparison Operators

= (equal), != (not equal), < (less than), > (greater than), <= (less/equal), >= (greater/equal)

Logical Operators

AND - All conditions must be true

```
SELECT * FROM books WHERE price > 20 AND category = 'Security';
```

OR - At least one condition true

```
SELECT * FROM books WHERE author = 'Erickson' OR price < 25;
```

NOT - Reverses condition

```
SELECT * FROM books WHERE NOT category = 'Fiction';
```

LIKE - Pattern matching

```
SELECT * FROM books WHERE title LIKE '%Hacking%';
```

% = any characters, _ = single character

BETWEEN - Range check

```
SELECT * FROM books WHERE price BETWEEN 20 AND 50;
```

Part 7: SQL Functions - Data Manipulation

Aggregate Functions

COUNT(*) - Count rows

```
SELECT COUNT(*) FROM books;
```

SUM() - Total of numeric column

```
SELECT SUM(price) FROM books;
```

AVG() - Average value

MIN() / MAX() - Minimum/Maximum

String Functions

CONCAT() - Combine strings

```
SELECT CONCAT(author, ' - ', title) FROM books;
```

SUBSTRING() - Extract portion

LENGTH() - String length

Best Practices & Security

- **Always use WHERE with UPDATE/DELETE**
- **Test queries with SELECT first**
- **Use transactions for critical operations**
- **Backup before major changes**
- **Never trust user input (SQL injection prevention)**
- **Use parameterized queries/prepared statements**
- **Implement least privilege access controls**

Conclusion

SQL is fundamental to modern computing and cybersecurity. Whether analyzing security logs, investigating breaches, or building applications, database skills are essential. Master these fundamentals: database/table management, CRUD operations, clauses, operators, and functions. Practice on real databases, understand relationships, and always prioritize security. SQL proficiency opens doors to SOC analysis, penetration testing, application security, and database administration.

Query wisely, secure thoroughly, analyze relentlessly. 🗄️ 🔒