Complete SQL Learning Guide - Beginner to Advanced

Week 1: SQL Foundations & Database Basics

Day 1-2: Database Creation & Basic Structure

Concepts:

- Creating databases and using them
- Understanding database objects
- Basic query writing sequence

Practice:

```sql
-- Creating your first database
CREATE DATABASE velocityBPI;
USE velocityBPI;

-- Query writing sequence (remember this order!)
SELECT [columns]
FROM [table]
WHERE [conditions]
ORDER BY [columns];
```

Day 3-4: Table Creation & Data Types

Concepts:

- CREATE TABLE syntax
- Data types (int, varchar, bigint, float, date)
- IDENTITY columns for auto-increment

Practice:

```sql
```

```sql
-- Create a student table
CREATE TABLE student (
    studentID int IDENTITY(1,1),
    studentName varchar(20),
    studentAge int,
    studentHeight int
);

-- Storage size understanding
-- tinyint: 1 byte (0-255)
-- smallint: 2 bytes
-- int: 4 bytes
-- bigint: 8 bytes
```

Day 5-7: Data Insertion Methods

Concepts:

- Three ways to insert data
- Best practices for data insertion

Practice:

```sql
-- Method 1: Simple insert
INSERT INTO student VALUES (1, 'manish', 24, 168);

-- Method 2: Multiple values (BEST PRACTICE)
INSERT INTO student (studentName, studentAge, studentHeight)
VALUES
    ('sid', 21, 169),
    ('harshita', 21, 149),
    ('akash', 26, 166);

-- Method 3: Using IDENTITY (most common)
INSERT INTO student VALUES ('romit', 24, 179);
```

Week 2: Data Filtering & Retrieval

Day 1-2: WHERE Clause Basics

Concepts:

- Comparison operators (=, <>, >, <, >=, <=)

- Filtering specific data

Practice:

```sql
-- Basic filtering
SELECT * FROM student WHERE studentAge = 24;
SELECT * FROM student WHERE studentAge > 24;
SELECT * FROM student WHERE studentName <> 'manish';
```

Day 3-4: Advanced WHERE Operators

Concepts:

- IN operator for multiple values
- BETWEEN for ranges
- NULL handling

Practice:

```sql
-- IN operator
SELECT * FROM student WHERE studentAge IN (23, 27);
SELECT * FROM student WHERE studentName IN ('manish', 'ramesh');

-- BETWEEN operator
SELECT * FROM student WHERE studentAge BETWEEN 23 AND 25;

-- NULL handling (CRITICAL CONCEPT!)
SELECT * FROM student WHERE studentHeight IS NULL;
SELECT * FROM student WHERE studentHeight IS NOT NULL;
-- Remember: NULL = NULL doesn't work!
```

Day 5-7: Pattern Matching & Logical Operators

Concepts:

- LIKE operator with wildcards
- AND, OR, NOT operators
- Underscore vs percentage wildcards

Practice:

```sql
```

```sql
-- LIKE operator patterns
SELECT * FROM student WHERE studentName LIKE 's%';    -- Starts with 's'
SELECT * FROM student WHERE studentName LIKE '%s';    -- Ends with 's'
SELECT * FROM student WHERE studentName LIKE '%sh%';  -- Contains 'sh'
SELECT * FROM student WHERE studentName LIKE '_a%';   -- Second letter is 'a'

-- Logical operators
SELECT * FROM student
WHERE studentName LIKE 'm%' AND studentAge > 20;

-- Finding even/odd records
SELECT * FROM student WHERE studentID % 2 = 0;  -- Even IDs
```

Week 3: Data Manipulation & Table Structure

Day 1-2: ORDER BY & Data Sorting

Concepts:

- Ascending (default) and descending order
- Multiple column sorting

Practice:

sql

```sql
-- Basic sorting
SELECT * FROM student ORDER BY studentAge;       -- ASC is default
SELECT * FROM student ORDER BY studentAge DESC;

-- Multiple column sorting
SELECT * FROM student
ORDER BY studentAge DESC, studentHeight DESC;
```

Day 3-4: Table Modification (ALTER)

Concepts:

- Adding columns
- Dropping columns
- Modifying table structure

Practice:

sql

```sql
-- Adding columns
ALTER TABLE student ADD gender CHAR(1);
ALTER TABLE student ADD phoneNumber BIGINT;
ALTER TABLE student ADD DOB DATE;

-- Dropping columns
ALTER TABLE student DROP COLUMN gender;
ALTER TABLE student DROP COLUMN DOB;
```

Day 5-7: Data Updates & Modifications

Concepts:

- UPDATE statement
- Conditional updates
- Multiple column updates

Practice:

```sql
-- Single column update
UPDATE student
SET studentHeight = 170
WHERE studentID = 8;

-- Multiple column update
UPDATE student
SET studentAge = 24,
   studentHeight = 140
WHERE studentName = 'gaurav';

-- Bulk updates
UPDATE student SET courseID = 1 WHERE studentID IN (1, 2);
```

Week 4: Relationships & Joins

Day 1-2: Foreign Keys & Relationships

Concepts:

- Creating related tables
- Understanding relationships
- Referential integrity

Practice:

```sql
-- Create course table
CREATE TABLE course (
    courseID int,
    courseName varchar(20)
);

INSERT INTO course VALUES
    (1, 'Power BI'),
    (2, 'Testing'),
    (3, 'Python'),
    (4, 'Java');

-- Add foreign key to student
ALTER TABLE student ADD courseID int;
```

Day 3-4: Basic Joins (INNER & LEFT)

Concepts:

- INNER JOIN (matching records only)
- LEFT JOIN (all from left table)
- Join syntax and logic

Practice:

```sql
-- INNER JOIN - only matching records
SELECT studentName, courseName
FROM student
INNER JOIN course ON student.courseID = course.courseID;

-- LEFT JOIN - all students, even without courses
SELECT studentName, courseName
FROM student
LEFT JOIN course ON student.courseID = course.courseID;
```

Day 5-7: Complete Join Types

Concepts:

- RIGHT JOIN
- FULL OUTER JOIN

- Self joins

Practice:

sql

```sql
-- RIGHT JOIN - all courses, even without students
SELECT studentName, courseName
FROM student
RIGHT JOIN course ON student.courseID = course.courseID;

-- FULL OUTER JOIN - all records from both tables
SELECT studentName, courseName
FROM student
FULL OUTER JOIN course ON student.courseID = course.courseID;

-- SELF JOIN - joining table with itself
SELECT emp.empName as Employee, man.empName as Manager
FROM employees emp
LEFT JOIN employees man ON emp.managerID = man.empID;
```

Week 5: Data Integrity & Constraints

Day 1-2: Basic Constraints

Concepts:

- NOT NULL constraint
- UNIQUE constraint
- Data validation

Practice:

sql

```sql
-- NOT NULL constraint
CREATE TABLE sampleNotNull (
    id int NOT NULL,
    name varchar(20) NOT NULL,
    age int
);

-- UNIQUE constraint
CREATE TABLE sampleUnique (
    id int UNIQUE,
    name varchar(20) NOT NULL,
    age int
);
```

Day 3-4: Advanced Constraints

Concepts:

- CHECK constraint
- DEFAULT values
- Custom validation rules

Practice:

sql

```sql
-- CHECK constraint
CREATE TABLE sampleCheck (
    id int NOT NULL CHECK(id > 3),
    name varchar(20) NOT NULL,
    age int
);

-- Email validation with CHECK
CREATE TABLE users (
    id int NOT NULL,
    name varchar(20) NOT NULL,
    email varchar(30) CHECK(email LIKE '%@gmail.com'),
    age int
);

-- DEFAULT constraint
CREATE TABLE sampleDefault (
    id int NOT NULL,
    name varchar(20) DEFAULT 'Unknown',
    age int DEFAULT 23
);
```

Day 5-7: Primary & Foreign Keys

Concepts:

- PRIMARY KEY (UNIQUE + NOT NULL)
- FOREIGN KEY relationships
- Referential integrity

Practice:

sql

```sql
-- Primary key table
CREATE TABLE department (
    deptID int PRIMARY KEY,
    deptName varchar(20)
);

-- Foreign key table
CREATE TABLE employee (
    empID int,
    empName varchar(20),
    empAge int,
    empDeptID int FOREIGN KEY REFERENCES department(deptID)
);
```

Week 6: Performance & Automation

Day 1-2: Indexes

Concepts:

- Clustered vs Non-clustered indexes
- Performance optimization
- When to use indexes

Practice:

```sql
-- Non-clustered index (like book index)
CREATE INDEX idx_department_id ON department(deptID);

-- Clustered index (like dictionary - data ordered)
CREATE CLUSTERED INDEX idx_employee_id ON employee(empID);

-- Remember:
-- - 1 clustered index per table
-- - 999 non-clustered indexes per table
```

Day 3-4: Stored Procedures Basics

Concepts:

- Creating reusable code blocks
- Parameters and execution
- Code organization

Practice:

```sql
-- Simple stored procedure
CREATE PROCEDURE spGetEmployees
AS
BEGIN
    SELECT * FROM employee
    LEFT JOIN department ON empDeptID = deptID;
END;

-- Execute procedure
EXEC spGetEmployees;
```

Day 5-7: Parameterized Procedures

Concepts:

- Input parameters
- Default values
- Dynamic procedures

Practice:

```sql
-- Procedure with parameters
CREATE PROCEDURE spGetStudent(@StudentID int)
AS
BEGIN
    SELECT * FROM student WHERE studentID = @StudentID;
END;

-- Multiple parameters with default
CREATE PROCEDURE spGetStudentByAge(@StudentID int, @Age int = 22)
AS
BEGIN
    SELECT * FROM student
    WHERE studentID = @StudentID OR studentAge = @Age;
END;

-- Execute with parameters
EXEC spGetStudent 1;
EXEC spGetStudentByAge 2, 24;
EXEC spGetStudentByAge 2;  -- Uses default age
```

Week 7: Advanced Querying

Day 1-2: UNION Operations

Concepts:

- UNION vs UNION ALL
- Combining result sets
- Data compatibility

Practice:

```sql
-- UNION ALL - keeps duplicates, maintains order
SELECT * FROM test1
UNION ALL
SELECT * FROM test2;

-- UNION - removes duplicates, sorts data
SELECT * FROM test1
UNION
SELECT * FROM test2;
```

Day 3-4: Views

Concepts:

- Virtual tables
- Data security and abstraction
- Updateable vs read-only views

Practice:

```sql
```

```sql
-- Simple view
CREATE VIEW vwStudent AS
SELECT studentName, studentAge FROM student;

-- Complex view with joins
CREATE VIEW vwStudentCourse AS
SELECT studentName, courseName
FROM student
LEFT JOIN course ON student.courseID = course.courseID;

-- Using views
SELECT * FROM vwStudent WHERE studentAge < 25;
```

Day 5-7: Subqueries

Concepts:

- Nested queries
- Inner queries vs main queries
- Correlated vs non-correlated subqueries

Practice:

```sql
-- Simple subquery
SELECT * FROM student
WHERE studentID IN (
    SELECT studentID FROM student WHERE courseID = 1
);

-- Finding second highest salary
SELECT MAX(salary) FROM payment
WHERE salary < (SELECT MAX(salary) FROM payment);

-- Subquery in FROM clause
SELECT studentName FROM (
    SELECT studentName, studentAge FROM student
) AS subTable;
```

Week 8: Advanced Analytics

Day 1-2: Window Functions - Ranking

Concepts:

- ROW_NUMBER(), RANK(), DENSE_RANK()
- PARTITION BY clause
- Advanced analytics

Practice:

```sql
-- Row numbering with partitioning
SELECT *,
    ROW_NUMBER() OVER (PARTITION BY studentCourse ORDER BY studentAge) as rn,
    RANK() OVER (PARTITION BY studentCourse ORDER BY studentAge) as rnk,
    DENSE_RANK() OVER (PARTITION BY studentCourse ORDER BY studentAge) as dense_rnk
FROM student;

-- Without partitioning
SELECT *,
    ROW_NUMBER() OVER (ORDER BY studentAge) as overall_rank
FROM student;
```

Day 3-4: Date Functions

Concepts:

- GETDATE(), DATEPART(), DATEADD(), DATEDIFF()
- Date manipulation and calculations
- Formatting dates

Practice:

```sql
-- Date functions
SELECT *,
    DATEPART(YYYY, DOB) as birthYear,
    DATEPART(MM, DOB) as birthMonth,
    DATEDIFF(YEAR, DOB, GETDATE()) as currentAge,
    DATEADD(YEAR, 1, DOB) as nextBirthday,
    FORMAT(DOB, 'dd-MM-yyyy') as formattedDate
FROM student;
```

Day 5-7: Conditional Logic

Concepts:

- CASE WHEN statements

- IF-ELSE logic
- Conditional updates

Practice:

```sql
-- CASE WHEN in SELECT
SELECT *,
   CASE
      WHEN studentAge <= 22 THEN 'Young'
      WHEN studentAge <= 25 THEN 'Adult'
      ELSE 'Mature'
   END as ageCategory
FROM student;

-- IF-ELSE in procedures
DECLARE @flag int = 2;
IF @flag = 2
   SELECT 'Condition Met';
ELSE
   SELECT 'Condition Not Met';
```

Week 9: Advanced Techniques

Day 1-2: Common Table Expressions (CTEs)

Concepts:

- Temporary result sets
- Improved readability
- CTE vs subqueries vs views

Practice:

```sql
```

```sql
-- Basic CTE
WITH studentCTE AS (
    SELECT studentName, studentAge FROM student
)
SELECT * FROM studentCTE;

-- CTE with column aliases
WITH studentCTE(sName, sAge, currentDate) AS (
    SELECT studentName, studentAge, GETDATE() FROM student
)
SELECT * FROM studentCTE;
```

Day 3-4: Temporary Tables

Concepts:

- Local (#) vs Global (##) temp tables
- Temporary data storage
- Performance considerations

Practice:

```sql
-- Local temporary table
CREATE TABLE #localTemp (
    id int IDENTITY(1,1),
    name varchar(20)
);

-- Global temporary table
CREATE TABLE ##globalTemp (
    id int IDENTITY(1,1),
    name varchar(20)
);

-- Creating temp table from existing data
SELECT * INTO ##studentBackup FROM student;
```

Day 5-7: Advanced Updates & Deletes

Concepts:

- Update based on JOIN
- DELETE with CTEs

- Removing duplicates

Practice:

```sql
-- Update using JOIN
UPDATE s
SET s.studentCourse = c.courseName
FROM student s
LEFT JOIN course c ON s.courseID = c.courseID;

-- Delete duplicates using CTE
WITH duplicateCTE AS (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY id ORDER BY id) as rn
    FROM ##temp
)
DELETE FROM duplicateCTE WHERE rn > 1;
```

Week 10: Loops & Advanced Programming

Day 1-2: Variables & Control Flow

Concepts:

- DECLARE variables
- SET vs SELECT for assignment
- Variable scope

Practice:

```sql
-- Variable declaration and usage
DECLARE @count int = 2;
DECLARE @name varchar(20);
SET @name = 'John';

-- Using variables in queries
SELECT * FROM student WHERE studentID = @count;
```

Day 3-4: WHILE Loops

Concepts:

- Loop structures

- Conditional looping
- Loop control

Practice:

```sql
-- Simple counter loop
DECLARE @counter int = 1;
WHILE @counter <= 5
BEGIN
    PRINT 'Counter: ' + CAST(@counter AS varchar);
    SET @counter = @counter + 1;
END;

-- Loop through records
DECLARE @id int = 1;
WHILE @id <= 3
BEGIN
    SELECT * FROM student WHERE studentID = @id;
    SET @id = @id + 1;
END;
```

Day 5-7: Data Type Conversions

Concepts:

- CAST vs CONVERT
- Implicit vs explicit conversion
- String manipulation

Practice:

```sql
-- Type conversion
DECLARE @a varchar(2) = '2';
DECLARE @b varchar(2) = '3';
SELECT CAST(@a AS int) * CAST(@b AS int) AS result;

-- Date conversion
SELECT CONVERT(date, GETDATE()) AS today;
SELECT CAST(GETDATE() AS date) AS today;
```

Practice Projects & Real-World Applications

Project 1: Student Management System

Build a complete system with:

- Student registration
- Course enrollment
- Grade management
- Reporting with joins and window functions

Project 2: Employee Database

Create an HR system with:

- Employee hierarchy (self-joins)
- Department management
- Salary analysis using window functions
- Performance tracking

Project 3: Sales Analysis

Develop a sales reporting system:

- Customer management
- Order processing
- Sales analytics with CTEs
- Trend analysis using date functions

Study Tips & Best Practices

1. **Practice Daily**: Spend 1-2 hours daily on SQL practice
2. **Build Projects**: Apply concepts in real scenarios
3. **Use Sample Databases**: Practice with AdventureWorks, Northwind
4. **Focus on Performance**: Always consider query optimization
5. **Document Your Code**: Use comments and meaningful names
6. **Test Thoroughly**: Always test edge cases and null values
7. **Learn Error Handling**: Understand common SQL errors
8. **Version Control**: Keep track of your database changes

Resources for Continued Learning

- **Online Platforms**: SQLBolt, W3Schools, Khan Academy

- **Practice Sites**: HackerRank, LeetCode SQL problems
- **Books**: "Learning SQL" by Alan Beaulieu
- **Documentation**: Microsoft SQL Server Documentation
- **Communities**: Stack Overflow, Reddit r/SQL

Remember: SQL mastery comes with consistent practice and real-world application. Focus on understanding concepts rather than memorizing syntax!