Table of Contents

Week 7: Introduction to Databases and SQL

Teaching Guide & Hands-on Project

📋 Session Overview

Duration: 3 Hours
Teaching: 75 minutes | **Hands-on Practice:** 45 minutes

🎯 Learning Objectives

By the end of this session, students will be able to:
1. Understand relational database concepts and terminology
2. Install and configure PostgreSQL database
3. Write basic SQL queries (SELECT, INSERT, UPDATE, DELETE)
4. Use WHERE clauses for filtering data
5. Perform JOIN operations to combine data from multiple tables
6. Design database schemas with proper relationships
7. Create tables with primary and foreign keys
8. Understand data types and constraints in PostgreSQL

📚 Quick Review (5 minutes)

Quick Questions:
- What's the difference between checked and unchecked exceptions?
- What does try-with-resources do?
- Why is file persistence important?
- What happens if we don't close a file?

Key Review Points:
- Exceptions handle errors gracefully
- Files provide data persistence
- Always close resources (or use try-with-resources)

🕐 Detailed Lesson Plan (120 Minutes)

Part 1: Database Concepts (15 minutes)

What is a Database? (5 minutes)

Definition:
- Organized collection of structured data
- Stored electronically in a computer system
- Managed by a Database Management System (DBMS)
- Provides efficient storage, retrieval, and manipulation of data

Why Use Databases Instead of Files?

```
FILES:
✗ No query capability
✗ Data redundancy
✗ No concurrent access
✗ No data integrity
✗ Limited search capability

DATABASES:
✓ Powerful query language (SQL)
✓ Data integrity and validation
✓ Multiple user access
✓ Data security
✓ Backup and recovery
✓ Relationships between data
```

Relational Database Concepts (10 minutes)

Key Terms:

```
┌─────────────────────────────────────────────────┐
│ TABLE (Relation)                                │
│                                                 │
│ ┌──────────┬──────────┬──────────┬──────────┐   │
│ │ ID (PK)  │ Name     │ Age      │ Email    │ ← Row
│ ├──────────┼──────────┼──────────┼──────────┤   │
│ │ 1        │ Alice    │ 20       │ a@e.com  │   │
│ │ 2        │ Bob      │ 22       │ b@e.com  │   │
│ │ 3        │ Charlie  │ 21       │ c@e.com  │   │
│ └──────────┴──────────┴──────────┴──────────┘   │
│   ↑                                             │
│ Column (Attribute)                              │
└─────────────────────────────────────────────────┘
```

```
Primary Key (PK): Uniquely identifies each row
Foreign Key (FK): References primary key in another table
```

Database Terminology:

- **Table (Relation):** Collection of related data organized in rows and columns
- **Row (Tuple/Record):** Single entry in a table
- **Column (Attribute/Field):** Specific piece of information
- **Primary Key:** Unique identifier for each row
- **Foreign Key:** Creates relationship between tables
- **Schema:** Structure/design of the database
- **Query:** Request for data from database

Example Schema - University Database:

STUDENTS Table:

student_id	name	age	email	major
S001	Alice	20	a@u.edu	CS
S002	Bob	22	b@u.edu	Math

↑ Primary Key

COURSES Table:

course_id	course_name	credits
CS101	Intro to Programming	3
MATH101	Calculus I	4

ENROLLMENTS Table:

student_id	course_id	grade
S001	CS101	A
S001	MATH101	B
S002	CS101	A

↑ ↑
Foreign Keys (create relationships)

Part 2: PostgreSQL Installation and Setup (10 minutes)

Installing PostgreSQL (5 minutes)

Step-by-step guide for students:

1. **Download PostgreSQL**
   - Visit: https://www.postgresql.org/download/
   - Choose your operating system
   - Download the installer
2. **Install PostgreSQL**
   - Run the installer
   - Choose installation directory
   - **Remember the password you set for 'postgres' user!**
   - Default port: 5432 (keep default)
   - Install all components including pgAdmin
3. **Verify Installation**
   - Open pgAdmin 4 (graphical interface)
   - Or use command line: `psql --version`

Using pgAdmin (5 minutes)

Quick Tour:
- Left panel: Server hierarchy
- Right panel: Query tool and results
- Top menu: Database operations

Connecting to PostgreSQL:
1. Open pgAdmin 4
2. Expand "Servers"
3. Right-click "PostgreSQL" → Connect
4. Enter password
5. You're connected!

Part 3: SQL Basics - Data Definition Language (DDL) (15 minutes)

Creating a Database (3 minutes)

SQL Example:

```sql
-- Create a new database
CREATE DATABASE university;

-- Note: In pgAdmin, right-click "Databases" → Create → Database
-- Name it "university"
```

Creating Tables (12 minutes)

Example 1: Simple Student Table

```sql
-- Connect to university database first
-- Then create table

CREATE TABLE students (
    student_id VARCHAR(10) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    age INTEGER CHECK (age ≥ 16 AND age ≤ 100),
    email VARCHAR(100) UNIQUE,
    major VARCHAR(50),
    gpa DECIMAL(3, 2) CHECK (gpa ≥ 0.0 AND gpa ≤ 4.0),
    enrollment_date DATE DEFAULT CURRENT_DATE
);
```

Common Data Types in PostgreSQL:

```sql
-- Text types
VARCHAR(n)      -- Variable length string, max n characters
TEXT            -- Unlimited length text
CHAR(n)         -- Fixed length string

-- Numeric types
INTEGER or INT  -- Whole numbers
DECIMAL(p, s)   -- Decimal numbers (precision, scale)
NUMERIC         -- Same as DECIMAL
REAL            -- Floating point
SERIAL          -- Auto-incrementing integer

-- Date and Time
DATE            -- Date only (YYYY-MM-DD)
TIME            -- Time only
TIMESTAMP       -- Date and time
TIMESTAMPTZ     -- Date and time with timezone

-- Boolean
BOOLEAN         -- TRUE or FALSE

-- Other
```

```
JSON            -- JSON dataUUID            -- Universally unique identifier
```

Common Constraints:

```
PRIMARY KEY     -- Unique identifier, cannot be NULL
NOT NULL        -- Value must be provided
UNIQUE          -- No duplicate values allowed
CHECK           -- Custom validation rule
DEFAULT         -- Default value if none provided
FOREIGN KEY     -- References another table
```

Example 2: Multiple Related Tables

```sql
-- Courses table
CREATE TABLE courses (
    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    credits INTEGER NOT NULL CHECK (credits > 0),
    department VARCHAR(50)
);

-- Enrollments table (junction table)
CREATE TABLE enrollments (
    enrollment_id SERIAL PRIMARY KEY,
    student_id VARCHAR(10) NOT NULL,
    course_id VARCHAR(10) NOT NULL,
    enrollment_date DATE DEFAULT CURRENT_DATE,
    grade CHAR(1) CHECK (grade IN ('A', 'B', 'C', 'D', 'F')),
    FOREIGN KEY (student_id) REFERENCES students(student_id)
        ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
        ON DELETE CASCADE,
    UNIQUE(student_id, course_id)  -- Student can't enroll twice in same course
);
```

Important Notes:
- `SERIAL` auto-generates unique numbers
- `FOREIGN KEY` creates relationship between tables
- `ON DELETE CASCADE` means if parent is deleted, child records are too
- `UNIQUE(col1, col2)` creates composite unique constraint

Part 4: SQL Basics - Data Manipulation Language (DML) (20 minutes)

INSERT - Adding Data (5 minutes)

Example 3: Inserting Records

```sql
-- Insert single student
INSERT INTO students (student_id, name, age, email, major, gpa)
VALUES ('S001', 'Alice Johnson', 20, 'alice@university.edu', 'Computer Science', 3.8);

-- Insert multiple students at once
INSERT INTO students (student_id, name, age, email, major, gpa)
VALUES
    ('S002', 'Bob Smith', 22, 'bob@university.edu', 'Mathematics', 3.5),
    ('S003', 'Charlie Brown', 21, 'charlie@university.edu', 'Physics', 3.9),
    ('S004', 'Diana Prince', 20, 'diana@university.edu', 'Computer Science', 3.7),
    ('S005', 'Eve Wilson', 23, 'eve@university.edu', 'Chemistry', 3.6);

-- Insert without specifying all columns (uses defaults)
INSERT INTO students (student_id, name, age, email, major)
VALUES ('S006', 'Frank Castle', 24, 'frank@university.edu', 'Engineering');
-- gpa will be NULL, enrollment_date will be today

-- Insert courses
INSERT INTO courses (course_id, course_name, credits, department)
VALUES
    ('CS101', 'Introduction to Programming', 3, 'Computer Science'),
    ('CS201', 'Data Structures', 4, 'Computer Science'),
    ('MATH101', 'Calculus I', 4, 'Mathematics'),
    ('PHYS101', 'Physics I', 4, 'Physics'),
    ('CHEM101', 'General Chemistry', 3, 'Chemistry');

-- Insert enrollments
INSERT INTO enrollments (student_id, course_id, grade)
VALUES
    ('S001', 'CS101', 'A'),
    ('S001', 'MATH101', 'B'),
    ('S002', 'MATH101', 'A'),
    ('S002', 'CS101', 'B'),
    ('S003', 'PHYS101', 'A'),
    ('S004', 'CS101', 'A'),
    ('S004', 'CS201', 'B');
```

SELECT - Retrieving Data (8 minutes)

Example 4: Basic SELECT Queries

```sql
-- Select all columns, all rows
SELECT * FROM students;

-- Select specific columns
```

```sql
SELECT name, major, gpa FROM students;          -- Select with calculated columns
SELECT name, gpa, (gpa * 25) AS percentage FROM students;
-- Select distinct values (no duplicates)SELECT DISTINCT major FROM students;
-- Count rowsSELECT COUNT(*) FROM students;SELECT COUNT(*) AS total_students FROM students;
         -- Aggregate functionsSELECT AVG(gpa) AS average_gpa FROM students;
SELECT MAX(gpa) AS highest_gpa FROM students;SELECT MIN(gpa) AS lowest_gpa FROM students;
SELECT SUM(credits) AS total_credits FROM courses;          -- Order results
SELECT name, gpa FROM students ORDER BY gpa DESC;
SELECT name, age FROM students ORDER BY age ASC, name ASC;          -- Limit results
SELECT * FROM students LIMIT 3;
SELECT * FROM students ORDER BY gpa DESC LIMIT 5;   -- Top 5 by GPA
```

Example 5: WHERE Clause - Filtering Data

```sql
-- Simple conditions
SELECT * FROM students WHERE major = 'Computer Science';
SELECT * FROM students WHERE age ≥ 21;
SELECT * FROM students WHERE gpa > 3.5;

-- Multiple conditions with AND
SELECT * FROM students
WHERE major = 'Computer Science' AND gpa ≥ 3.7;

-- Multiple conditions with OR
SELECT * FROM students
WHERE major = 'Computer Science' OR major = 'Mathematics';

-- NOT operator
SELECT * FROM students WHERE NOT major = 'Computer Science';

-- IN operator (multiple values)
SELECT * FROM students
WHERE major IN ('Computer Science', 'Mathematics', 'Physics');

-- BETWEEN operator
SELECT * FROM students WHERE age BETWEEN 20 AND 22;
SELECT * FROM students WHERE gpa BETWEEN 3.5 AND 4.0;

-- LIKE operator (pattern matching)
SELECT * FROM students WHERE name LIKE 'A%';       -- Starts with A
SELECT * FROM students WHERE name LIKE '%son';     -- Ends with son
SELECT * FROM students WHERE email LIKE '%@university.edu';

-- IS NULL / IS NOT NULL
SELECT * FROM students WHERE gpa IS NULL;
SELECT * FROM students WHERE gpa IS NOT NULL;

-- Combining conditions
SELECT * FROM students
WHERE (major = 'Computer Science' OR major = 'Mathematics')
```

```
  AND gpa ≥ 3.5  AND age ≤ 22;
```

UPDATE - Modifying Data (4 minutes)

Example 6: Updating Records

```
-- Update single column for one student
UPDATE students
SET gpa = 3.9
WHERE student_id = 'S001';

-- Update multiple columns
UPDATE students
SET age = 21, email = 'alice.new@university.edu'
WHERE student_id = 'S001';

-- Update based on condition
UPDATE students
SET major = 'Computer Science and Engineering'
WHERE major = 'Computer Science';

-- Update with calculation
UPDATE enrollments
SET grade = 'A'
WHERE student_id = 'S002' AND course_id = 'CS101';

-- Update multiple rows
UPDATE students
SET gpa = gpa + 0.1
WHERE gpa < 3.0;  -- Boost GPA for struggling students

-- IMPORTANT: Always use WHERE clause!
-- Without WHERE, ALL rows are updated!
```

⚠ Warning:

```
-- DANGEROUS: Updates ALL students!
UPDATE students SET gpa = 4.0;  -- Don't do this!
```

DELETE - Removing Data (3 minutes)

Example 7: Deleting Records

```
-- Delete specific student
DELETE FROM students WHERE student_id = 'S006';

-- Delete based on condition
```

```sql
DELETE FROM students WHERE gpa < 2.0;          -- Delete multiple rows
DELETE FROM enrollments WHERE grade = 'F';         -- Delete all enrollments for a student
DELETE FROM enrollments WHERE student_id = 'S005';         -- DANGEROUS: Delete all rows!
-- DELETE FROM students;  -- Don't do this without WHERE!
```

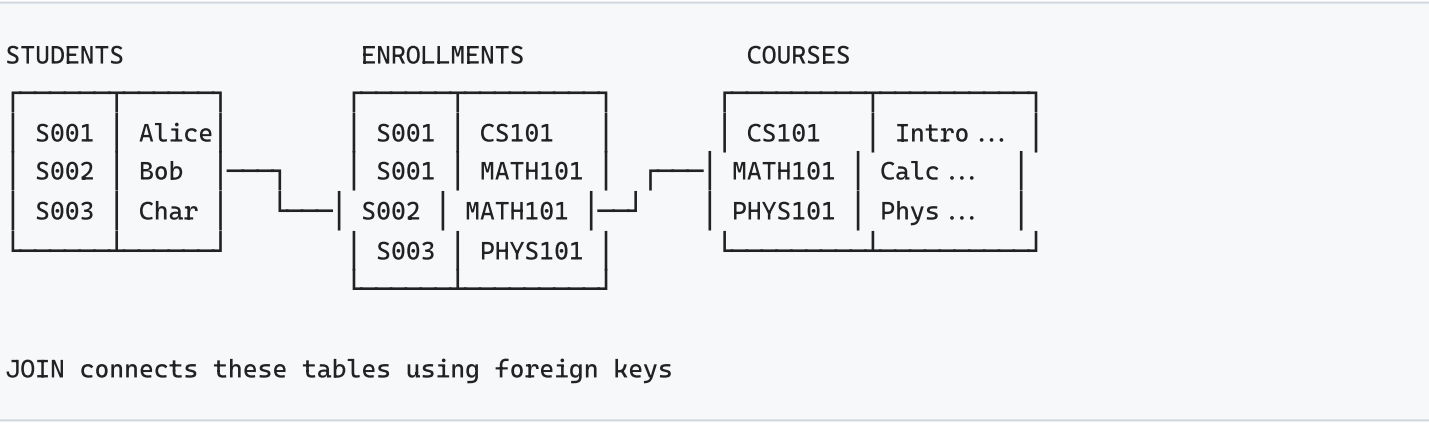
⚠️ **Warning:** Always use WHERE clause unless you want to delete ALL rows!

Part 5: JOIN Operations (15 minutes)

Understanding JOINs (5 minutes)

Why JOINs?
- Combine data from multiple tables
- Retrieve related information
- Avoid data duplication

Visual Representation:



INNER JOIN (4 minutes)

Example 8: Basic INNER JOIN

```sql
-- Get student names with their enrolled courses
SELECT students.name, courses.course_name
FROM students
INNER JOIN enrollments ON students.student_id = enrollments.student_id
INNER JOIN courses ON enrollments.course_id = courses.course_id;

-- Using table aliases (shorter syntax)
SELECT s.name, c.course_name, e.grade
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id;

-- With WHERE clause
SELECT s.name, c.course_name, e.grade
```

```
FROM students sINNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_idWHERE s.major = 'Computer Science';
-- With ORDER BYSELECT s.name, c.course_name, e.gradeFROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_idORDER BY s.name, c.course_name;
```

LEFT JOIN (3 minutes)

Example 9: LEFT JOIN

```
-- Get all students, including those not enrolled in any course
SELECT s.name, c.course_name
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
LEFT JOIN courses c ON e.course_id = c.course_id;

-- Find students with no enrollments
SELECT s.student_id, s.name
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
WHERE e.enrollment_id IS NULL;
```

GROUP BY and HAVING (3 minutes)

Example 10: Grouping and Aggregation

```
-- Count courses per student
SELECT s.name, COUNT(e.course_id) AS course_count
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
ORDER BY course_count DESC;

-- Average GPA by major
SELECT major, AVG(gpa) AS avg_gpa, COUNT(*) AS student_count
FROM students
GROUP BY major
ORDER BY avg_gpa DESC;

-- Students taking more than 1 course
SELECT s.name, COUNT(e.course_id) AS course_count
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
HAVING COUNT(e.course_id) > 1;

-- Courses with average grade
SELECT c.course_name, COUNT(e.enrollment_id) AS enrollment_count
FROM courses c
```

```
LEFT JOIN enrollments e ON c.course_id = e.course_idGROUP BY c.course_id, c.course_name
ORDER BY enrollment_count DESC;
```

💻 HANDS-ON PROJECT: University Database Design (45 minutes)

Project Overview

Design and implement a complete university database with:

1. Students table
2. Courses table
3. Teachers table
4. Enrollments table
5. Departments table

Then write queries to retrieve useful information.

Step 1: Database Schema Design

```
-- Create the database
CREATE DATABASE university_db;

-- Connect to it in pgAdmin, then create tables:

-- Departments table
CREATE TABLE departments (
    department_id SERIAL PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE,
    building VARCHAR(50),
    phone VARCHAR(20)
);

-- Teachers table
CREATE TABLE teachers (
    teacher_id VARCHAR(10) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    department_id INTEGER,
    hire_date DATE DEFAULT CURRENT_DATE,
    salary DECIMAL(10, 2),
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

```
    student_id VARCHAR(10) PRIMARY KEY,    name VARCHAR(100) NOT NULL,
    age INTEGER CHECK (age ≥ 16 AND age ≤ 100),    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),    major VARCHAR(50),
    gpa DECIMAL(3, 2) CHECK (gpa ≥ 0.0 AND gpa ≤ 4.0),    department_id INTEGER,
    enrollment_date DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (department_id) REFERENCES departments(department_id));
-- Courses table (enhanced)CREATE TABLE courses (    course_id VARCHAR(10) PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,    description TEXT,
    credits INTEGER NOT NULL CHECK (credits > 0 AND credits ≤ 6),    department_id INTEGER,
    teacher_id VARCHAR(10),    max_students INTEGER DEFAULT 30,
    FOREIGN KEY (department_id) REFERENCES departments(department_id),
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id));
-- Enrollments table (enhanced)CREATE TABLE enrollments (    enrollment_id SERIAL PRIMARY KEY,
    student_id VARCHAR(10) NOT NULL,    course_id VARCHAR(10) NOT NULL,
    enrollment_date DATE DEFAULT CURRENT_DATE,
    grade CHAR(1) CHECK (grade IN ('A', 'B', 'C', 'D', 'F', 'W')),    semester VARCHAR(20),
    year INTEGER,
    FOREIGN KEY (student_id) REFERENCES students(student_id) ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(course_id) ON DELETE CASCADE,
    UNIQUE(student_id, course_id, semester, year));
```

Step 2: Insert Sample Data

```
-- Insert departments
INSERT INTO departments (department_name, building, phone)
VALUES
    ('Computer Science', 'Engineering Building', '555-0101'),
    ('Mathematics', 'Science Hall', '555-0102'),
    ('Physics', 'Science Hall', '555-0103'),
    ('Chemistry', 'Chemistry Building', '555-0104'),
    ('Engineering', 'Engineering Building', '555-0105');

-- Insert teachers
INSERT INTO teachers (teacher_id, name, email, department_id, salary)
VALUES
    ('T001', 'Dr. John Smith', 'jsmith@university.edu', 1, 85000.00),
    ('T002', 'Dr. Sarah Johnson', 'sjohnson@university.edu', 1, 90000.00),
    ('T003', 'Dr. Michael Brown', 'mbrown@university.edu', 2, 82000.00),
    ('T004', 'Dr. Emily Davis', 'edavis@university.edu', 3, 88000.00),
    ('T005', 'Dr. Robert Wilson', 'rwilson@university.edu', 4, 86000.00);

-- Insert students
INSERT INTO students (student_id, name, age, email, major, gpa, department_id)
VALUES
    ('S001', 'Alice Johnson', 20, 'alice@university.edu', 'Computer Science', 3.8, 1),
    ('S002', 'Bob Smith', 22, 'bob@university.edu', 'Mathematics', 3.5, 2),
    ('S003', 'Charlie Brown', 21, 'charlie@university.edu', 'Physics', 3.9, 3),
    ('S004', 'Diana Prince', 20, 'diana@university.edu', 'Computer Science', 3.7, 1),
```

```
    ('S005', 'Eve Wilson', 23, 'eve@university.edu', 'Chemistry', 3.6, 4),
    ('S006', 'Frank Castle', 24, 'frank@university.edu', 'Engineering', 3.4, 5),
    ('S007', 'Grace Hopper', 19, 'grace@university.edu', 'Computer Science', 4.0, 1),
    ('S008', 'Henry Ford', 21, 'henry@university.edu', 'Engineering', 3.3, 5);
-- Insert courses
INSERT INTO courses (course_id, course_name, description, credits, department_id, teacher_id)
VALUES
    ('CS101', 'Introduction to Programming', 'Learn Java programming basics', 3, 1, 'T001'),
    ('CS201', 'Data Structures', 'Advanced data structures and algorithms', 4, 1, 'T002'),
    ('CS301', 'Database Systems', 'Relational databases and SQL', 3, 1, 'T002'),
    ('MATH101', 'Calculus I', 'Differential calculus', 4, 2, 'T003'),
    ('MATH201', 'Linear Algebra', 'Matrices and vector spaces', 3, 2, 'T003'),
    ('PHYS101', 'Physics I', 'Mechanics and thermodynamics', 4, 3, 'T004'),
    ('CHEM101', 'General Chemistry', 'Introduction to chemistry', 3, 4, 'T005'),
    ('ENG101', 'Engineering Fundamentals', 'Basic engineering principles', 3, 5, 'T001');
        -- Insert enrollments
INSERT INTO enrollments (student_id, course_id, grade, semester, year)VALUES
    ('S001', 'CS101', 'A', 'Fall', 2024),    ('S001', 'MATH101', 'B', 'Fall', 2024),
    ('S001', 'CS201', 'A', 'Spring', 2025),    ('S002', 'MATH101', 'A', 'Fall', 2024),
    ('S002', 'CS101', 'B', 'Fall', 2024),    ('S003', 'PHYS101', 'A', 'Fall', 2024),
    ('S004', 'CS101', 'A', 'Fall', 2024),    ('S004', 'CS201', 'B', 'Spring', 2025),
    ('S005', 'CHEM101', 'A', 'Fall', 2024),    ('S006', 'ENG101', 'B', 'Fall', 2024),
    ('S007', 'CS101', 'A', 'Fall', 2024),    ('S007', 'CS201', 'A', 'Spring', 2025),
    ('S007', 'CS301', NULL, 'Spring', 2025);  -- Currently enrolled, no grade yet
```

Step 3: Practice Queries

Students should write these queries:

```sql
-- Query 1: List all students with their departments
SELECT s.name, s.major, d.department_name
FROM students s
LEFT JOIN departments d ON s.department_id = d.department_id
ORDER BY d.department_name, s.name;

-- Query 2: Find all courses taught by each teacher
SELECT t.name AS teacher_name, c.course_name, c.credits
FROM teachers t
LEFT JOIN courses c ON t.teacher_id = c.teacher_id
ORDER BY t.name;

-- Query 3: Show student enrollments with course details
SELECT s.name AS student_name, c.course_name, e.grade, e.semester, e.year
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id
ORDER BY s.name, e.year, e.semester;
```

```sql
-- Query 4: Find honor roll students (GPA ≥ 3.5)SELECT name, major, gpaFROM students
WHERE gpa ≥ 3.5ORDER BY gpa DESC;          -- Query 5: Count students per department
SELECT d.department_name, COUNT(s.student_id) AS student_countFROM departments d
LEFT JOIN students s ON d.department_id = s.department_id
GROUP BY d.department_id, d.department_nameORDER BY avg_gpa DESC;
```

Step 4: Advanced Queries Challenge

Challenge students with these:

```sql
-- Challenge 1: Find students who got 'A' in all their courses
SELECT s.name, COUNT(e.enrollment_id) AS total_courses
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
WHERE e.grade = 'A'
GROUP BY s.student_id, s.name
HAVING COUNT(e.enrollment_id) = (
    SELECT COUNT(*)
    FROM enrollments
    WHERE student_id = s.student_id AND grade IS NOT NULL
);

-- Challenge 2: Courses with above-average enrollment
SELECT c.course_name, COUNT(e.enrollment_id) AS enrollment_count
FROM courses c
LEFT JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
HAVING COUNT(e.enrollment_id) > (
    SELECT AVG(course_count)
    FROM (
        SELECT COUNT(enrollment_id) AS course_count
        FROM enrollments
        GROUP BY course_id
    ) AS avg_enrollments
);

-- Challenge 3: Students and their class rank by GPA
SELECT name, major, gpa,
    RANK() OVER (ORDER BY gpa DESC) AS overall_rank,
    RANK() OVER (PARTITION BY major ORDER BY gpa DESC) AS major_rank
FROM students
WHERE gpa IS NOT NULL
ORDER BY overall_rank;

-- Challenge 4: Teachers teaching the most credits
SELECT t.name, SUM(c.credits) AS total_credits_taught
FROM teachers t
INNER JOIN courses c ON t.teacher_id = c.teacher_id
```

```sql
GROUP BY t.teacher_id, t.nameORDER BY total_credits_taught DESC;
-- Challenge 5: Update all Computer Science students' GPA by 0.1UPDATE students
SET gpa = LEAST(gpa + 0.1, 4.0)  -- Don't exceed 4.0
WHERE major = 'Computer Science' AND gpa IS NOT NULL;
```

📝 Homework Assignment

Task 1: Create Library Database (Required)

Design and implement a library database with:

Tables:
1. **books** - book_id, title, author, isbn, publication_year, available_copies
2. **members** - member_id, name, email, phone, join_date
3. **borrowings** - borrowing_id, member_id, book_id, borrow_date, due_date, return_date

Requirements:
- Create all tables with appropriate constraints
- Insert at least 10 books, 5 members, and 8 borrowing records
- Write queries to:
- List all currently borrowed books
- Find overdue books
- Show members with no borrowings
- Count total borrowings per member
- Find most popular books

Task 2: Add More Complex Queries (Required)

For the university database, write queries to:
1. Find students taking courses from multiple departments
2. Calculate average salary by department for teachers
3. List all courses a specific student needs for graduation (assume 120 total credits needed)
4. Find students who dropped courses (grade = 'W')
5. Show teacher workload (number of courses taught)

Task 3: Design Your Own Schema (Challenge)

Design a database for one of these scenarios:
- **E-commerce:** Products, customers, orders, order_items
- **Hospital:** Patients, doctors, appointments, prescriptions
- **Social Media:** Users, posts, comments, likes

Create tables, insert sample data, and write at least 10 meaningful queries.

🎯 Assessment Checklist

Students should be able to:
- [ ] Understand relational database concepts
- [ ] Install and connect to PostgreSQL
- [ ] Create databases and tables
- [ ] Use appropriate data types for columns
- [ ] Define primary keys and foreign keys
- [ ] Insert data into tables
- [ ] Write SELECT queries with WHERE, ORDER BY, LIMIT
- [ ] Use aggregate functions (COUNT, AVG, SUM, MIN, MAX)
- [ ] Update existing records
- [ ] Delete records safely
- [ ] Perform INNER JOIN operations
- [ ] Perform LEFT JOIN operations
- [ ] Use GROUP BY and HAVING clauses
- [ ] Write complex multi-table queries

🔧 Common Student Errors & Solutions

Error 1: Syntax Error in Query

```
-- Wrong
SELECT name age FROM students;

-- Correct
SELECT name, age FROM students;
```

Solution: Always use commas between column names

Error 2: Ambiguous Column Name in JOIN

```
-- Error: column "name" is ambiguous
SELECT name, course_name
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id;

-- Solution: Use table aliases
SELECT s.name, c.course_name
```

```
FROM students sINNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id;
```

Error 3: Forgetting WHERE Clause

```
-- DANGER: Updates ALL students!
UPDATE students SET gpa = 4.0;

-- Correct: Update specific student
UPDATE students SET gpa = 4.0 WHERE student_id = 'S001';
```

Solution: Always double-check UPDATE and DELETE statements

Error 4: Using WHERE Instead of HAVING

```
-- Wrong: WHERE can't use aggregate functions
SELECT major, AVG(gpa)
FROM students
WHERE AVG(gpa) > 3.5
GROUP BY major;

-- Correct: Use HAVING for aggregates
SELECT major, AVG(gpa)
FROM students
GROUP BY major
HAVING AVG(gpa) > 3.5;
```

Error 5: Foreign Key Violation

```
-- Error: violates foreign key constraint
INSERT INTO enrollments (student_id, course_id)
VALUES ('S999', 'CS101');  -- S999 doesn't exist!

-- Solution: Ensure referenced records exist first
```

📚 Additional Resources

Official Documentation:

- PostgreSQL Documentation

- PostgreSQL Tutorial
- SQL Syntax Reference

Practice Platforms:

- SQLZoo - Interactive SQL tutorial
- LeetCode Database - SQL practice problems
- HackerRank SQL - SQL challenges
- Mode Analytics SQL Tutorial

Tools:

- pgAdmin 4 - PostgreSQL GUI
- DBeaver - Universal database tool
- DB Diagram - Design database schemas visually

🔜 Preview of Week 8

Next week we'll cover:
- **JDBC (Java Database Connectivity):** Connect Java to PostgreSQL
- **Executing SQL from Java:** PreparedStatement and Statement
- **ResultSet:** Processing query results
- **CRUD Operations:** Building a complete data access layer
- **SQL Injection Prevention:** Using PreparedStatement safely
- **Connection Management:** Best practices

Prepare: Make sure PostgreSQL is working and you understand basic SQL queries!

📌 Key Takeaways

> ***"Database = Organized Data Storage"***
> *Structured, queryable, and persistent data*

> ***"SQL = Standard Query Language"***
> *Universal language for relational databases*

> *"**Primary Key = Unique Identifier**"*
> *Every table should have one*

> *"**Foreign Key = Relationship**"*
> *Links tables together*

> *"**JOIN = Combine Tables**"*
> *Retrieve related data from multiple tables*

> *"**Always Use WHERE**"*
> *Especially with UPDATE and DELETE!*

🎯 Quick Reference Guide

Essential SQL Commands:

```sql
-- CREATE
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints
);

-- INSERT
INSERT INTO table_name (col1, col2) VALUES (val1, val2);

-- SELECT
SELECT col1, col2 FROM table_name WHERE condition;

-- UPDATE
UPDATE table_name SET col1 = value WHERE condition;

-- DELETE
DELETE FROM table_name WHERE condition;

-- JOIN
SELECT * FROM table1 t1
INNER JOIN table2 t2 ON t1.id = t2.foreign_id;

-- GROUP BY
SELECT col1, COUNT(*) FROM table_name GROUP BY col1;
```

```sql
-- ORDER BYSELECT * FROM table_name ORDER BY col1 DESC;
```

Common Data Types:

```sql
VARCHAR(n)      -- Variable text
INTEGER         -- Whole numbers
DECIMAL(p,s)    -- Decimals
DATE            -- Dates
BOOLEAN         -- True/false
SERIAL          -- Auto-increment
TEXT            -- Long text
TIMESTAMP       -- Date and time
```

Common Constraints:

```sql
PRIMARY KEY     -- Unique identifier
FOREIGN KEY     -- References another table
NOT NULL        -- Cannot be empty
UNIQUE          -- No duplicates
CHECK           -- Custom validation
DEFAULT         -- Default value
```

💻 Live Coding Exercise (If Time Permits)

Build a simple blog database together:

```sql
-- 1. Create tables
CREATE TABLE authors (
    author_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    join_date DATE DEFAULT CURRENT_DATE
);

CREATE TABLE posts (
    post_id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    content TEXT,
    author_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    views INTEGER DEFAULT 0,
    FOREIGN KEY (author_id) REFERENCES authors(author_id)
);
```

```
        CREATE TABLE comments (    comment_id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL,    author_id INTEGER NOT NULL,    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES posts(post_id) ON DELETE CASCADE,
    FOREIGN KEY (author_id) REFERENCES authors(author_id));
-- 2. Insert sample dataINSERT INTO authors (username, email) VALUES
    ('john_doe', 'john@blog.com'),    ('jane_smith', 'jane@blog.com'),
    ('bob_tech', 'bob@blog.com');        INSERT INTO posts (title, content, author_id) VALUES
    ('My First Post', 'This is my first blog post!', 1),
    ('SQL is Awesome', 'Learning SQL has been great ... ', 1),
    ('Java Tips', 'Here are some Java programming tips ... ', 2);
INSERT INTO comments (post_id, author_id, content) VALUES    (1, 2, 'Great post!'),
    (1, 3, 'Welcome to blogging!'),    (2, 2, 'I agree, SQL is powerful!');
-- 3. Useful queries-- Get all posts with author info
SELECT p.title, p.content, a.username, p.created_atFROM posts p
INNER JOIN authors a ON p.author_id = a.author_idORDER BY p.created_at DESC;
-- Get post with comment countSELECT p.title, a.username, COUNT(c.comment_id) AS comment_count
FROM posts pINNER JOIN authors a ON p.author_id = a.author_id
LEFT JOIN comments c ON p.post_id = c.post_idGROUP BY p.post_id, p.title, a.username;
-- Get all comments for a specific postSELECT p.title, a.username, c.content, c.created_at
FROM comments cINNER JOIN posts p ON c.post_id = p.post_id
INNER JOIN authors a ON c.author_id = a.author_idWHERE p.post_id = 1ORDER BY c.created_at;
```

🎓 Summary

This week, students learned:

1. ✅ Fundamental database concepts and terminology
2. ✅ How to install and use PostgreSQL
3. ✅ Creating tables with proper constraints
4. ✅ Inserting, updating, and deleting data
5. ✅ Writing SELECT queries with filtering and sorting
6. ✅ Using aggregate functions and grouping
7. ✅ Performing JOIN operations to combine tables
8. ✅ Designing complete database schemas

Next week: We'll connect Java to PostgreSQL using JDBC and build a complete data access layer!

Students should practice writing SQL queries and become comfortable with the database they created. This foundation is crucial for Week 7's JDBC integration.d.department_name
ORDER BY student_count DESC;

– Query 6: Average GPA by major
SELECT major, AVG(gpa) AS avg_gpa, COUNT*() AS student_count*
FROM students
WHERE gpa IS NOT NULL
GROUP BY major

```
HAVING COUNT() > 1
ORDER BY avg_gpa DESC;

– Query 7: Students enrolled in more than 2 courses
SELECT s.name, COUNT(e.enrollment_id) AS course_count
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.name
HAVING COUNT(e.enrollment_id) > 2
ORDER BY course_count DESC;

– Query 8: Most popular courses (by enrollment)
SELECT c.course_name, COUNT(e.enrollment_id) AS enrollment_count
FROM courses c
LEFT JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
ORDER BY enrollment_count DESC;

– Query 9: Students with A grades
SELECT s.name, c.course_name, e.grade
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id
WHERE e.grade = 'A'
ORDER BY s.name;

– Query 10: Teachers and their departments with salaries
SELECT t.name AS teacher_name, d.department_name, t.salary
FROM teachers t
LEFT JOIN departments d ON t.department_id = d.department_id
ORDER BY t.salary DESC;

– Query 11: Students not enrolled in any course
SELECT s.student_id, s.name, s.major
FROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id
WHERE e.enrollment_id IS NULL;

– Query 12: Courses with no enrollments
SELECT c.course_id, c.course_name, c.credits
FROM courses c
LEFT JOIN enrollments e ON c.course_id = e.course_id
WHERE e.enrollment_id IS NULL;

– Query 13: Total credits per student
SELECT s.name, SUM(c.credits) AS total_credits
FROM students s
```

```
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id
GROUP BY s.student_id, s.name
ORDER BY total_credits DESC;

– Query 14: Find Computer Science students taking CS courses
SELECT s.name, c.course_name, e.grade
FROM students s
INNER JOIN enrollments e ON s.student_id = e.student_id
INNER JOIN courses c ON e.course_id = c.course_id
WHERE s.major = 'Computer Science'
AND c.course_id LIKE 'CS%'
ORDER BY s.name, c.course_name;

– Query 15: Department with highest average student GPA
SELECT d.department_name, AVG(s.gpa) AS avg_gpa, COUNT(s.student_id) AS student_count
FROM departments d
INNER JOIN students s ON d.department_id = s.department_id
WHERE s.gpa IS NOT NULL
GROUP BY d.department_id,
```