```sql
/*
╔══════════════════════════════════════════════════════════════════════════════╗
║                     SQL QUERY GUIDE: BEGINNER TO EXPERT                        ║
║                                                                                ║
║  Master SQL step-by-step with practical examples                             ║
║  Sample Tables: employees, departments, customers, orders, products           ║
╚══════════════════════════════════════════════════════════════════════════════╝
*/

-- ═══════════════════════════════════════════════════════════════════════════════
-- 📚 PART 1: BASIC QUERIES - Start Here!
-- Learn the fundamentals: selecting, filtering, and sorting data
-- ═══════════════════════════════════════════════════════════════════════════════

-- ─────────────────────────────────────────────────────────────────────────────
-- Query 1: Select Everything
-- Use this to see all columns and rows from a table
-- ─────────────────────────────────────────────────────────────────────────────
SELECT * FROM employees;


-- ─────────────────────────────────────────────────────────────────────────────
-- Query 2: Select Specific Columns
-- Choose only the columns you need (faster and cleaner!)
-- ─────────────────────────────────────────────────────────────────────────────
SELECT
    first_name,
    last_name,
    salary
FROM employees;


-- ─────────────────────────────────────────────────────────────────────────────
-- Query 3: Filter with WHERE
-- Find records that meet specific conditions
-- ─────────────────────────────────────────────────────────────────────────────
SELECT *
FROM employees
WHERE salary > 50000;


-- ─────────────────────────────────────────────────────────────────────────────
-- Query 4: Sort Results
-- Organize your data (ASC = ascending, DESC = descending)
-- ─────────────────────────────────────────────────────────────────────────────
SELECT
    first_name,
    salary
```

```
FROM employees
ORDER BY salary DESC;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 5: Multiple Conditions
-- Combine filters using AND (both must be true) or OR (either can be true)
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT *
FROM employees
WHERE department_id = 3
  AND salary > 60000;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 6: Pattern Matching
-- Find text that matches a pattern (% = any characters, _ = one character)
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT *
FROM employees
WHERE first_name LIKE 'J%';  -- Names starting with 'J'


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 7: Count Records
-- How many rows are in your table?
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT COUNT(*) AS total_employees
FROM employees;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 8: Find Unique Values
-- Remove duplicates to see distinct values
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT DISTINCT department_id
FROM employees;



-- ═══════════════════════════════════════════════════════════════════════════════════════
-- 📊 PART 2: INTERMEDIATE QUERIES - Level Up!
-- Aggregations, joins, subqueries, and conditional logic
-- ═══════════════════════════════════════════════════════════════════════════════════════

-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 9: Group and Aggregate
-- Calculate statistics for each group (COUNT, SUM, AVG, MIN, MAX)
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT
```

```
    department_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id;


-- ──────────────────────────────────────────────────────────────────────────────────────
-- Query 10: Filter Groups with HAVING
-- WHERE filters rows, HAVING filters groups (use after GROUP BY)
-- ──────────────────────────────────────────────────────────────────────────────────────
SELECT
    department_id,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 60000;


-- ──────────────────────────────────────────────────────────────────────────────────────
-- Query 11: INNER JOIN
-- Combine tables - only returns matching records from both tables
-- ──────────────────────────────────────────────────────────────────────────────────────
SELECT
    e.first_name,
    e.last_name,
    d.department_name
FROM employees e
INNER JOIN departments d
    ON e.department_id = d.department_id;


-- ──────────────────────────────────────────────────────────────────────────────────────
-- Query 12: LEFT JOIN
-- Returns ALL records from left table + matching records from right table
-- ──────────────────────────────────────────────────────────────────────────────────────
SELECT
    c.customer_name,
    o.order_id,
    o.total_amount
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id;


-- ──────────────────────────────────────────────────────────────────────────────────────
-- Query 13: Subquery
-- A query inside another query - useful for comparisons
-- ──────────────────────────────────────────────────────────────────────────────────────
SELECT
```

```sql
    first_name,
    last_name,
    salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);  -- Above average earners


-- ─────────────────────────────────────────────────────────────────────────────
-- Query 14: CASE Statement
-- Create conditional logic (like IF-THEN-ELSE in programming)
-- ─────────────────────────────────────────────────────────────────────────────
SELECT
    first_name,
    salary,
    CASE
        WHEN salary < 40000 THEN 'Low'
        WHEN salary < 70000 THEN 'Medium'
        ELSE 'High'
    END AS salary_level
FROM employees;


-- ─────────────────────────────────────────────────────────────────────────────
-- Query 15: String Functions
-- Manipulate text data (CONCAT, UPPER, LOWER, SUBSTRING, etc.)
-- ─────────────────────────────────────────────────────────────────────────────
SELECT
    CONCAT(first_name, ' ', last_name) AS full_name,
    UPPER(first_name) AS uppercase_name
FROM employees;



-- ═══════════════════════════════════════════════════════════════════════════════
-- 🚀 PART 3: ADVANCED QUERIES - Expert Level!
-- Window functions, CTEs, recursive queries, and complex analytics
-- ═══════════════════════════════════════════════════════════════════════════════

-- ─────────────────────────────────────────────────────────────────────────────
-- Query 16: Window Function - Ranking
-- Rank rows without grouping them (unlike GROUP BY)
-- ─────────────────────────────────────────────────────────────────────────────
SELECT
    first_name,
    salary,
    RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM employees;


-- ─────────────────────────────────────────────────────────────────────────────
-- Query 17: Partition and Rank
-- Rank within groups - best performer in each department
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT
    first_name,
    department_id,
    salary,
    ROW_NUMBER() OVER (
        PARTITION BY department_id
        ORDER BY salary DESC
    ) AS dept_rank
FROM employees;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 18: Running Total
-- Calculate cumulative sum as you go through rows
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT
    order_date,
    total_amount,
    SUM(total_amount) OVER (ORDER BY order_date) AS running_total
FROM orders;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 19: Common Table Expression (CTE)
-- Create temporary named result set - makes complex queries readable!
-- ─────────────────────────────────────────────────────────────────────────────────────
WITH dept_avg AS (
    SELECT
        department_id,
        AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
)
SELECT
    e.first_name,
    e.salary,
    d.avg_salary
FROM employees e
JOIN dept_avg d
    ON e.department_id = d.department_id
WHERE e.salary > d.avg_salary;  -- Employees earning above department average


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 20: Recursive CTE - Organizational Hierarchy
-- Navigate through hierarchical data (like org charts, family trees)
-- ─────────────────────────────────────────────────────────────────────────────────────
```

```sql
WITH RECURSIVE emp_hierarchy AS (
    -- Base case: Start with top-level managers
    SELECT
        employee_id,
        first_name,
        manager_id,
        1 AS level
    FROM employees
    WHERE manager_id IS NULL

    UNION ALL

    -- Recursive case: Add employees reporting to previous level
    SELECT
        e.employee_id,
        e.first_name,
        e.manager_id,
        eh.level + 1
    FROM employees e
    JOIN emp_hierarchy eh
        ON e.manager_id = eh.employee_id
)
SELECT * FROM emp_hierarchy;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 21: Top N per Group
-- Get the top 3 highest earners in each department
-- ─────────────────────────────────────────────────────────────────────────────────────
SELECT *
FROM (
    SELECT
        first_name,
        department_id,
        salary,
        ROW_NUMBER() OVER (
            PARTITION BY department_id
            ORDER BY salary DESC
        ) AS rn
    FROM employees
) ranked
WHERE rn <= 3;


-- ─────────────────────────────────────────────────────────────────────────────────────
-- Query 22: Customer Analytics with Segmentation
-- Real-world business intelligence: analyze and segment customers
-- ─────────────────────────────────────────────────────────────────────────────────────
WITH customer_stats AS (
    SELECT
```

```
        c.customer_id,
        c.customer_name,
        COUNT(o.order_id) AS total_orders,
        SUM(o.total_amount) AS total_spent
    FROM customers c
    LEFT JOIN orders o
        ON c.customer_id = o.customer_id
    GROUP BY c.customer_id, c.customer_name
)
SELECT
    customer_name,
    total_orders,
    total_spent,
    CASE
        WHEN total_orders >= 10 THEN 'VIP'
        WHEN total_orders >= 5 THEN 'Loyal'
        ELSE 'Regular'
    END AS segment
FROM customer_stats
ORDER BY total_spent DESC;


/*
╔══════════════════════════════════════════════════════════════════════════════════════════╗
║                           🎓 LEARNING PATH                               ║
╠══════════════════════════════════════════════════════════════════════════════════════════╣
║  1. Master Basic Queries (1-8) - Foundation is everything!             ║
║  2. Practice Intermediate (9-15) - This is where SQL gets powerful       ║
║  3. Challenge yourself with Advanced (16-22) - Become an expert          ║
║                                                                          ║
║  💡 Pro Tip: Run each query, modify it, break it, fix it - that's how    ║
║     you truly learn!                                                     ║
╚══════════════════════════════════════════════════════════════════════════════════════════╝
*/
```