SQL Prompting Cheatsheet

Quick Reference for Using Claude Code with Databases

Query Generation Prompts

1. Simple SELECT with Filters

```
"Write a SQL query to find all customers in California with orders over $100."
```

What You Get: Proper WHERE clauses, JOINs if needed, correct column selection.

2. Aggregation & GROUP BY

```
"Show me total revenue by product category for the last quarter, sorted highest to lowest."
```

What You Get: SUM/COUNT aggregations, GROUP BY category, date filtering, ORDER BY.

3. Multi-Table JOINs

```
"Find all products that have never been ordered, include product name and category."
```

What You Get: LEFT JOIN with NULL check, or NOT EXISTS subquery, proper table relationships.

4. Subqueries & CTEs

```
"List customers who spent more than the average customer in 2025."
```

What You Get: Subquery calculating AVG, or CTE for readability, proper comparison logic.

5. Window Functions (Advanced)

```
"Show each salesperson's monthly sales with a running total and rank within their region."
```

What You Get: PARTITION BY, ORDER BY within window, ROW_NUMBER/RANK/SUM OVER functions.

Schema Design Prompts

1. From Business Requirements

```
"Design a database schema for a hotel reservation system. Include rooms, guests, bookings, payments,
```

What You Get: Complete table definitions, foreign keys, constraints, indexes, junction tables for many-to-many.

2. Add Table to Existing Schema

```
"Add a 'reviews' table to this e-commerce schema. Customers can review products they've purchased. Ir
```

What You Get: Table definition with foreign keys to customers and products, CHECK constraint on rating, indexes.

3. Modify Existing Schema

```
"Add audit fields (created_at, updated_at, created_by) to all tables in this schema."
```

What You Get: ALTER TABLE statements, default values (CURRENT_TIMESTAMP), triggers for auto-update if needed.

Optimization Prompts

1. Analyze Slow Query

```
"This query takes 15 seconds on a table with 1 million rows. Optimize it and explain why it's slow: [
```

What You Get: EXPLAIN analysis, index recommendations, query rewrite (JOIN vs subquery), cardinality estimates.

2. Index Recommendations

```
"Recommend indexes for this schema based on these common queries: [paste queries]"
```

What You Get: CREATE INDEX statements with column names, composite index suggestions, covering index opportunities.

3. Rewrite for Performance

```
"Rewrite this query to avoid the N+1 problem: [paste query with repeated subqueries]"
```

What You Get: Single query with proper JOINs, elimination of correlated subqueries, batch operations.

ETL & Data Transformation Prompts

1. Schema Migration

```
"Write a migration script to transform this denormalized table [paste schema] into a normalized schem
```

What You Get: Multi-step INSERT...SELECT statements, deduplication logic, referential integrity preservation.

2. Data Cleanup

```
"Write queries to clean this data: remove duplicates based on email, standardize phone number format
```

What You Get: UPDATE statements with CASE logic, regex patterns, deduplication with ROW_NUMBER.

3. Format Conversion

```
"Convert this CSV data structure to SQL INSERT statements for the following schema: [paste schema]"
```

What You Get: Properly escaped INSERT statements, type conversions, NULL handling.

Data Validation Prompts

1. Referential Integrity Check

```
"Write queries to find all orphaned records (foreign keys pointing to non-existent records) in this d
```

What You Get: LEFT JOIN with NULL checks for each foreign key relationship, reports by table.

2. Duplicate Detection

```
"Find duplicate customers based on email address. Show all columns for duplicates so I can manually r
```

What You Get: GROUP BY with HAVING COUNT(*) > 1, CTE to show full records.

3. Constraint Violations

```
"Find all records that violate these business rules: order total must be positive, due date must be a
```

What You Get: WHERE clauses checking each rule, separate queries per table/rule, count of violations.

4. Data Profiling

```
"Generate a data quality report for the customers table: count total rows, NULL values per column, di
```

What You Get: Multiple queries with aggregations, statistical summaries, identification of data quality issues.

5. Anomaly Detection

```
"Find unusual patterns in the orders table: orders with negative totals, orders older than 5 years, o
```

What You Get: Queries with outlier detection logic, thresholds, anomaly flagging.

Tips for Better SQL Output

□DO: Specify Database Engine

- × Bad: "Write a query to..."
- □Good: "Write a **PostgreSQL** query to..."

Why: Syntax varies between databases (LIMIT vs FETCH FIRST, || vs CONCAT, date functions).

□DO: Provide Schema Context

- × Bad: "Find customers who ordered products"
- □Good: "Using this schema [paste], find customers who ordered products"

Why: Claude Code needs table names, column names, and relationships to generate correct queries.

Pro Tip: Use `@file` mentions: `claude @schema.sql "write a query to..."`

□DO: Ask for EXPLAIN Analysis

- × Bad: "Optimize this query"

- □Good: "Optimize this query and show EXPLAIN output before and after"

Why: EXPLAIN shows actual query plan, index usage, and performance impact.

□DO: Request Parameterized Queries

- × Bad: "Find customer with email 'john@example.com'"
- □Good: "Find customer by email, use a parameterized query to prevent SQL injection"

Why: Security. Parameterized queries prevent injection attacks.

□DO: Specify Output Format

- × Bad: "Generate seed data"
- □Good: "Generate 50 rows of realistic seed data as INSERT statements with proper escaping"

Why: Clarifies whether you want raw data, SQL statements, CSV, JSON, etc.

□DO: Ask for Explanations

- × Bad: "Fix this query"
- □Good: "Fix this query and explain what was wrong and why your version is better"

Why: You learn the underlying principles, not just get working code.

Common Pitfalls to Avoid

× Not Specifying Database Engine

Problem: Gets generic SQL that might not work on your database.
Example: PostgreSQL uses `LIMIT`, SQL Server uses `TOP`, Oracle uses `FETCH FIRST`.
Fix: Always start with "Using [PostgreSQL/MySQL/SQLite/SQL Server]..."

× Missing Table/Column Context

Problem: Claude Code guesses table names and gets them wrong.
Example: You have `users` table, Claude assumes `customers`.
Fix: Paste schema or use `@file` to include schema definition.

× Accepting Generated SQL Without Testing

Problem: Trusts output blindly, discovers errors in production.
Example: Query works on small dataset, times out on production data.
Fix: Always run queries on test data, check EXPLAIN plans, verify results.

× Vague Requirements

Problem: "Make the query faster" doesn't give Claude Code enough direction.
Example: Claude optimizes the wrong part of the query.
Fix: Be specific: "This query is slow because of the subquery in WHERE. Rewrite using a JOIN."

× Ignoring Security

Problem: Uses string concatenation instead of parameterized queries.
Example: `SELECT * FROM users WHERE email = '${userInput}'` ← SQL injection risk!
Fix: Request parameterized queries explicitly or ask Claude Code to show safe parameter binding.

× Not Validating Data Types

Problem: Stores dates as VARCHAR, prices as FLOAT.
Example: Sorting dates alphabetically instead of chronologically.
Fix: Ask Claude Code to review schema and recommend proper data types.

× Skipping Indexes

Problem: Asks for schema but not index recommendations.
Example: Foreign keys without indexes cause slow JOINs.
Fix: Always ask: "Include index recommendations for common queries."

× Over-Reliance on SELECT *

Problem: Fetches all columns when only a few are needed.
Example: `SELECT * FROM orders JOIN customers JOIN products` pulls gigabytes of unnecessary data.
Fix: Ask Claude Code: "Select only the columns needed for [specific use case]."

× Not Testing Edge Cases

Problem: Query works for happy path, fails on NULLs or empty results.
Example: AVG calculation returns NULL when no rows match.
Fix: Ask: "Handle edge cases: empty results, NULL values, divide-by-zero."

Pro Tips

□Chain Requests for Complex Tasks

Don't ask for everything at once. Build incrementally: 1. "Design schema for X" 2. "Add indexes for these queries" 3. "Generate seed data" 4. "Write validation queries"

□Use @file Mentions for Schema

```
claude @schema.sql "Write a query to find customers with no orders"
```

Claude Code will read the schema file and use exact table/column names.

□Ask for Multiple Approaches

```
"Show me 3 ways to solve this: using JOIN, using subquery, using EXISTS. Which is fastest?"
```

Learn different techniques and understand tradeoffs.

□Request Realistic Test Data

```
"Generate 100 rows of realistic customer data: real-sounding names, valid email formats, US addresses
```

Way better than "Customer 1", "Customer 2"...

□Get Validation Queries Automatically

```
"After creating this schema, write 5 validation queries to check data integrity."
```

Ensures your data stays clean over time.

□Learn While You Build

```
"Explain why you chose this JOIN type instead of a subquery."
"Walk me through how this window function works step-by-step."
```

Turn Claude Code into a SQL tutor.

Example Workflow: End-to-End Database Development

```
# 1. Design Schema
claude "Design a PostgreSQL schema for a blog platform with users, posts, comments, tags, and categor

# 2. Get Indexes
claude "Add index recommendations for these queries: list posts by author, search posts by tag, show

# 3. Generate Seed Data
claude "Generate realistic seed data: 50 users, 200 posts, 500 comments, 20 tags. Use real-sounding r

# 4. Validation Queries
claude "Write data validation queries to check for: orphaned comments, duplicate emails, posts withou

# 5. Common Queries
claude "Generate queries for: top 10 posts by comment count, users with most posts, trending tags thi

# 6. Optimization
claude "These queries are slow [paste]. Optimize them and recommend indexes."
```

Quick Examples

Task	Prompt
Find duplicates	`"Find duplicate records in customers table based on email"`
Add column	`"Add a 'verified' boolean column to users table, default FALSE"`
Calculate running total	`"Show daily sales with running total using window function"`
Pivot data	`"Convert this data from rows to columns: [describe structure]"`
Recursive query	`"Query all employees and their managers, showing the full hierarchy"`
Full-text search	`"Add full-text search to the posts table (PostgreSQL)"`
JSON query	`"Query the JSONB column to find users with role = 'admin'"`
Bulk insert	`"Generate bulk INSERT for 1000 rows with transaction and error handling"`

Database-Specific Tips

PostgreSQL

- Mention JSONB, array types, CTEs, window functions
- Request `EXPLAIN ANALYZE` for actual execution stats
- Ask about pg_trgm for fuzzy search, PostGIS for geo queries

MySQL

- Use `EXPLAIN` (not ANALYZE)
- Request stored procedures if needed
- Mention InnoDB vs MyISAM when relevant

SQLite

- Specify limitations: no RIGHT JOIN, limited ALTER TABLE
- Great for local development and testing
- Request `.schema` output for compatibility checks

SQL Server

- Use `TOP` instead of `LIMIT`
- Request `SET STATISTICS TIME ON` for performance
- Mention SSMS-specific features if using that tool

Remember: Claude Code is a powerful SQL assistant, but you're still the developer. Always review generated queries, test them on real data, and understand what they're doing. Use Claude Code to learn, not just to copy-paste.

Last Updated: 2026-02-13
For: AIA Copilot Training — Claude Code Fundamentals (Day 2)