SQL Detective: The Data Crime Investigation Game (3D Edition)

Professional Technical Documentation

Version: 1.0
Date: January 2026
Author: Ananya B
Document Type: Technical Portfolio Documentation

Table of Contents

1. Executive Summary
2. Project Overview
3. System Architecture
4. Technology Stack
5. Database Design
6. Game Design and Level Progression
7. 3D UI and User Experience Design
8. Backend Logic
9. Frontend Logic
10. Security and Data Safety
11. Development Workflow
12. Deployment and Usage
13. Challenges and Solutions
14. Performance and Optimization
15. Future Enhancements
16. Resume and Interview Highlights

---

1. Executive Summary

Project at a Glance

SQL Detective is a full-stack web application that gamifies SQL education through crime investigation gameplay. Players assume the role of a data detective solving a bank heist by querying realistic crime databases within an immersive 3D noir environment.

Problem Statement

Traditional SQL learning methods rely on static exercises and disconnected datasets. Students often struggle to: - Understand when to apply specific SQL concepts - Retain query patterns without meaningful context - Stay motivated through repetitive practice

Solution

This project addresses these challenges by: - Embedding SQL practice within a compelling crime narrative - Providing immediate feedback on query correctness - Creating an immersive 3D environment that increases engagement - Progressively introducing concepts from SELECT to Window Functions

Why This Project is Resume-Worthy

Aspect	Demonstration
Full-Stack Development	Flask backend, Three.js frontend, SQLite database
Security Engineering	SQL injection prevention, read-only enforcement
3D Graphics	Interactive Three.js environment with raycasting
Database Design	Normalized schema with realistic crime data
Game Design	Progressive difficulty with pedagogical scaffolding
API Design	RESTful endpoints for query execution and validation

Unique Value Proposition

Unlike existing SQL trainers, SQL Detective combines: - Narrative-driven learning with investigation storytelling - 3D spatial interaction rather than static web forms - Real query execution against actual databases - Security-first architecture suitable for production deployment

---

2. Project Overview

Game Concept

Players enter a noir detective office rendered in 3D. They interact with objects in the room to: - Read case files containing crime narratives - View evidence boards showing database schemas - Write SQL queries at a computer terminal - Solve crimes by producing correct query results

Storyline

A bank heist has occurred at Downtown Bank. The player must analyze phone records, CCTV footage, and financial transactions to identify the perpetrators. Each level reveals new evidence and requires increasingly sophisticated SQL skills.

Target Audience

Audience	Use Case
SQL Beginners	Learning fundamental query syntax
Intermediate Learners	Practicing JOINs and aggregations
Advanced Users	Mastering CTEs and window functions
Educators	Teaching SQL in classroom settings
Interviewers	Assessing candidate SQL proficiency

Learning Objectives

Upon completing all seven levels, players will be able to: - Write filtered queries using WHERE clauses - Sort and limit result sets - Join multiple tables using foreign key relationships - Aggregate data with GROUP BY and HAVING - Construct nested subqueries - Organize complex queries using Common Table Expressions - Analyze data patterns using window functions

---

3. System Architecture

High-Level Architecture Diagram

```
+------------------------------------------------------------------+
|                           BROWSER                                 |
|  +-------------------+  +------------------+  +----------------+  |
|  |   Three.js 3D     |  |   SQL Editor     |  |  UI Panels     |  |
|  |   Scene Renderer  |  |   Component      |  |  (Story/Evd)   |  |
|  +--------+----------+  +--------+---------+  +-------+--------+  |
|           |                      |                    |           |
|           +----------------------+--------------------+           |
|                                  |                                |
+----------------------------------+--------------------------------+
                                   | HTTP REST API
                                   v
+------------------------------------------------------------------+
|                        FLASK SERVER                               |
|  +----------------+  +------------------+  +-------------------+  |
|  | Route Handlers |  | SQL Validator    |  | Level Checker     |  |
|  | (game.py)      |  | (security layer) |  | (answer verify)   |  |
|  +-------+--------+  +--------+---------+  +---------+---------+  |
|          |                    |                      |            |
|          +--------------------+----------------------+            |
|                               |                                   |
+-------------------------------+-----------------------------------+
                                | SQLite Read-Only Connection
                                v
+------------------------------------------------------------------+
|                         SQLITE DATABASE                            |
|  +----------+ +-------------+ +----------+ +-------------------+   |
|  | suspects | | phone_recs  | | cctv_logs| | bank_transactions|   |
|  +----------+ +-------------+ +----------+ +-------------------+   |
+-------------------------------------------------------------------+
```

Data Flow

1. User clicks 3D object in browser
2. JavaScript identifies action type via raycasting
3. UI panel opens (story, evidence, or SQL editor)
4. User writes and submits SQL query
5. Frontend sends POST request to Flask API
6. Backend validates query for security threats
7. If valid, query executes against read-only SQLite
8. Results return to frontend for display
9. Answer checker compares results to expected output
10. Feedback displays and level progression updates

Component Responsibilities

Component	Responsibility
Three.js Scene	3D rendering, object interaction, camera control
UI Panels	Story display, table schemas, SQL editing
Flask Routes	HTTP request handling, session management
SQL Validator	Security enforcement, keyword blocking
Query Executor	Safe database access, timeout protection
Level Checker	Answer comparison, hint generation

---

4. Technology Stack

Backend: Flask (Python)

Selection Rationale: - Lightweight framework with minimal boilerplate - Native SQLite support through Python standard library - Simple deployment without complex configuration - Excellent for API-first architectures

Key Usage: - Blueprint-based route organization - Session-based progress tracking - Static file serving for frontend

Database: SQLite

Selection Rationale: - Zero-configuration embedded database - File-based storage simplifies deployment - Read-only mode enforcement at connection level - Sufficient performance for educational workloads

Key Usage: - URI-mode connection with read-only flag - Parameterized timeout for query execution - Row factory for column name access

Frontend: Three.js

Selection Rationale: - Industry-standard WebGL abstraction - Runs in browser without plugins - Rich ecosystem for 3D interaction - Procedural geometry eliminates asset dependencies

Key Usage: - Scene graph for room construction - Raycaster for click detection - OrbitControls for camera movement - Point lights for atmospheric lighting

Supporting Technologies

Technology	Purpose
HTML5	Document structure
CSS3	Styling with CSS variables
JavaScript ES6	Frontend logic with modules
Flask-CORS	Cross-origin request handling

---

5. Database Design

Entity Relationship Overview

The database models a crime investigation scenario with interconnected entities representing suspects, their activities, and locations.

```
SUSPECTS ----< PHONE_RECORDS >---- SUSPECTS (caller/receiver)
    |
    +----< BANK_TRANSACTIONS
    |
    +----< CCTV_LOGS >---- LOCATIONS
                               |
                        CRIME_SCENES
```

Table Purposes

Table	Purpose	Key Columns
suspects	Person profiles	id, name, age, occupation, criminal_record
locations	City places	id, name, type, address
phone_records	Call/SMS logs	caller_id, receiver_id, timestamp, duration

Table	Purpose	Key Columns
bank_transactions	Financial data	account_id, amount, timestamp, type
cctv_logs	Surveillance	person_id, location_id, timestamp, confidence
crime_scenes	Incidents	case_number, crime_type, location_id, date_time
case_progress	Player state	player_id, current_level, completed levels

Schema Snippet: Suspects Table

```
CREATE TABLE suspects (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(100) NOT NULL,
    age INTEGER NOT NULL,
    gender VARCHAR(20),
    occupation VARCHAR(100),
    address VARCHAR(200),
    phone_number VARCHAR(20),
    criminal_record INTEGER DEFAULT 0,
    notes TEXT
);
```

Data Integrity

- Primary keys enforce entity uniqueness
- Foreign keys link related records
- NOT NULL constraints ensure required data
- DEFAULT values provide sensible fallbacks

Sample Data Volume

Table	Record Count
suspects	10
locations	10
phone_records	25
bank_transactions	18
cctv_logs	18
crime_scenes	3

6. Game Design and Level Progression

Progression Philosophy

The game follows a scaffolded learning approach where each level builds upon previously introduced concepts. Early levels use single tables with simple filters, while later levels require multi-table joins and analytical functions.

Level Structure

Level	Title	SQL Concepts	Complexity
1	The Missing Witness	SELECT, WHERE, AND	Beginner
2	The Midnight Call	BETWEEN, ORDER BY, LIMIT	Beginner
3	The Connection	INNER JOIN	Intermediate
4	The Pattern	GROUP BY, HAVING, COUNT	Intermediate
5	The Money Trail	Subqueries, AVG	Advanced
6	The Movement	CTEs (WITH clause)	Advanced
7	The Final Piece	Window Functions	Expert

Sample Missions

Level 1: Find all suspects over 30 years old with prior criminal records.

Level 3: Identify suspects captured on CCTV at the Downtown Bank location.

Level 7: Analyze transaction patterns to detect suspicious spikes in spending.

Pedagogical Features

- Contextual hints available on request
- Multiple correct query formulations accepted
- Immediate feedback on syntax errors
- Progressive table access prevents overwhelming beginners

---

7. 3D UI and User Experience Design

Environment Concept

The player enters a noir detective office rendered in 3D. The room features: - A wooden desk with case files and a desk lamp - An evidence board on the back wall - A computer terminal for SQL queries - Venetian blinds suggesting a window - A bookshelf with reference materials

Interactive Elements

Object	Interaction	Result
Case File (desk)	Click	Opens story panel with mission
Evidence Board	Click	Displays available table schemas
Computer Terminal	Click	Opens SQL editor overlay
Help Button (HUD)	Click	Shows gameplay instructions

UI/UX Decisions

Dark Theme: Noir aesthetic reduces eye strain and enhances focus on content.

Terminal Aesthetic: SQL editor styled as green-on-black terminal reinforces coding context.

Minimal Text: Visual storytelling prioritized over lengthy instructions.

Immediate Feedback: Query results appear instantly with row counts and execution time.

Accessibility Considerations

- High contrast color scheme
- Keyboard navigation support for panels
- Clear error messages with actionable hints
- Resizable text in SQL editor

Why 3D Improves Engagement

- Spatial context creates memorable learning environment
- Object interaction mimics real-world investigation
- Visual variety prevents monotony of form-based learning
- Exploration rewards curiosity with environmental details

---

8. Backend Logic

SQL Validation Approach

All incoming queries pass through a multi-stage validation pipeline:

1. **Empty Check:** Reject blank queries
2. **Statement Type:** Verify query starts with SELECT or WITH
3. **Keyword Scan:** Block dangerous keywords (DROP, DELETE, etc.)
4. **Multi-Statement Detection:** Reject queries containing semicolons mid-query
5. **Length Limit:** Cap queries at 5000 characters
6. **Pattern Matching:** Block suspicious constructs (LOAD_FILE, SLEEP, etc.)

Query Execution Flow

```
User Query
    |
    v
Sanitize (trim, normalize)
    |
    v
Validate (security checks)
    |
    +-- Invalid --> Return error message
    |
    v
Execute (read-only connection)
    |
    +-- Timeout --> Return timeout error
    |
    v
Format Results (columns, rows)
    |
    v
Return JSON Response
```

Security Measures

- Read-only database connection mode
- Query timeout (5 seconds)
- Blocked keyword list
- Multi-statement prevention
- No raw SQL interpolation

Error Handling

User-friendly error messages replace technical SQLite errors: - “no such table” becomes “Table not found: [name]” - “no such column” becomes “Column not found” - Syntax errors include the problematic portion

---

9. Frontend Logic

Three.js Scene Structure

```
Scene
  +-- Ambient Light
  +-- Main Room Light
  +-- Detective Room Group
  |     +-- Floor
  |      +-- Walls
  |      +-- Desk Group (interactive)
  |      +-- Evidence Board (interactive)
  |      +-- Computer Terminal (interactive)
  |      +-- Bookshelf
  |      +-- Filing Cabinet
  |      +-- Venetian Blinds
  |      +-- Dust Particles
  +-- Camera
  +-- OrbitControls
```

Interaction Handling

Click detection uses Three.js Raycaster:

1. Convert mouse position to normalized device coordinates
2. Cast ray from camera through mouse position
3. Check intersection with interactive objects
4. Read action type from object userData
5. Dispatch appropriate panel opening function

SQL Editor Integration

The editor component provides: - Line number display (updates on input) - Syntax-aware placeholder text - Keyboard shortcuts (Ctrl+Enter to execute) - Tab key indentation support - Clear button for reset

Animation and Feedback Logic

- Successful answer: Green glow effect, success message
- Incorrect answer: Red highlight, shake animation, hints display
- Loading state: Button spinner during query execution
- Panel transitions: Slide-in animation with opacity fade

---

10. Security and Data Safety

Read-Only Enforcement

Database connections use SQLite URI mode with read-only flag:

```
sqlite3.connect('file:database.db?mode=ro', uri=True)
```

This prevents any write operations at the connection level.

SQL Injection Prevention

Multiple layers protect against injection attacks:

Layer	Protection
Keyword Blocking	Rejects DROP, DELETE, UPDATE, INSERT, ALTER
Statement Validation	Only SELECT and WITH statements allowed
Multi-Query Prevention	Semicolons mid-query rejected
Pattern Matching	Known attack patterns blocked

Blocked Keyword List

```
DROP, DELETE, UPDATE, INSERT, ALTER, CREATE, TRUNCATE,
GRANT, REVOKE, EXEC, EXECUTE, PRAGMA, ATTACH, DETACH
```

Execution Limits

- Query timeout: 5 seconds
- Maximum result rows: 1000
- Maximum query length: 5000 characters

---

11. Development Workflow

Step-by-Step Process

1. **Database Design**
   - Created schema with normalized tables
   - Wrote seed data with realistic crime narrative
   - Tested relationships and queries
2. **Backend Core**
   - Implemented Flask application structure
   - Built SQL validation service
   - Created query executor with safety measures
3. **Level Configuration**
   - Defined all seven levels with stories
   - Wrote expected queries for each level
   - Implemented answer checking logic
4. **API Development**
   - Created game routes for levels and progress
   - Built query routes for execution and checking
   - Added table schema endpoints
5. **Frontend Foundation**
   - Set up Three.js scene with room geometry
   - Implemented camera and controls
   - Added atmospheric lighting
6. **Interactive Objects**
   - Created desk, evidence board, computer models
   - Implemented raycasting for click detection

    - Registered objects with action handlers

7. **UI Panels**
    - Built story panel with case file display
    - Created evidence panel with table viewer
    - Developed SQL editor with results display
8. **Integration and Polish**
    - Connected frontend to backend API
    - Added feedback animations
    - Implemented level progression

Testing Strategy

Test Type	Scope
Unit Tests	SQL validator, level checker
Integration Tests	API endpoints, database operations
Manual Testing	UI interactions, level completion
Security Testing	Injection attempts, blocked keywords

---

12. Deployment and Usage

Prerequisites

- Python 3.8 or higher
- Modern web browser with WebGL support

Installation Steps

```
# Clone repository
git clone <repository-url>
cd sql-detective-game

# Install Python dependencies
pip install -r requirements.txt

# Navigate to backend
cd backend

# Run application
python app.py
```

Accessing the Game

Open browser and navigate to:

```
http://localhost:5000
```

Folder Structure

```
sql-detective-game/
|
+-- backend/
|   +-- app.py              # Flask entry point
|   +-- config.py           # Configuration
|   +-- database/
|   |   +-- schema.sql      # Table definitions
|   |   +-- seed_data.sql   # Initial data
|   +-- routes/
|   |   +-- game.py         # Game API
|   |   +-- query.py        # Query API
|   +-- services/
|   |   +-- sql_validator.py
|   |   +-- query_executor.py
|   |   +-- level_checker.py
|   +-- levels/
|       +-- level_config.py
|
+-- frontend/
|   +-- index.html
|   +-- css/
|   |   +-- main.css
|   |   +-- sql-editor.css
|   |   +-- ui-panels.css
|   +-- js/
|       +-- main.js
|       +-- scene/
|       |   +-- DetectiveRoom.js
|       +-- api/
|           +-- gameAPI.js
|
+-- requirements.txt
+-- README.md
```

---

13. Challenges and Solutions

Challenge 1: 3D Object Click Detection

Problem: Determining which 3D object was clicked required translating 2D mouse coordinates to 3D space.

Solution: Implemented Three.js Raycaster with object userData to store action identifiers. Parent objects register as interactive, and child meshes inherit the action type during traversal.

Challenge 2: Answer Verification Without Exact Query Match

Problem: Multiple valid SQL queries can produce the same result set. String comparison would reject correct alternatives.

Solution: Compare result sets rather than query strings. Convert results to normalized tuples, create sets, and check set equality. Order-sensitive levels use list comparison instead.

Challenge 3: Secure SQL Execution

Problem: User-submitted SQL could potentially damage the database or extract sensitive information.

Solution: Multi-layer security: - Read-only database connection - Keyword blocking at validation layer - Query timeout to prevent resource exhaustion - No table access beyond current level

Design Trade-offs

Decision	Trade-off	Rationale
Procedural geometry	Less visual detail	No external asset dependencies
Session storage	No persistence across browsers	Simplifies architecture
SQLite	Limited concurrent users	Sufficient for educational use

---

14. Performance and Optimization

SQL Optimization

- Indexed columns used in WHERE clauses
- Limited result set sizes (max 1000 rows)
- Query timeout prevents long-running operations
- Read-only mode eliminates write overhead

UI Performance

- Dust particles use BufferGeometry for efficiency
- Shadow maps limited to key light sources
- Orbit controls use damping for smooth movement
- Panel animations use CSS transforms (GPU accelerated)

Network Optimization

- Single HTTP request per query execution
- JSON responses minimized to essential data
- Static assets served with appropriate caching headers

Measured Performance

Metric	Value
Query execution	Under 100ms typical
3D scene load	Under 2 seconds
Panel transitions	300ms animation

---

15. Future Enhancements

Phase 2: Enhanced Gameplay

- Sound effects for interactions and feedback
- Background ambient audio (rain, city sounds)
- More detailed 3D models
- Additional camera angles

Phase 3: Extended Content

- Additional crime case scenarios
- Harder difficulty modes
- Timed challenge mode
- Achievement system

Phase 4: Social Features

- User accounts with saved progress
- Global leaderboard
- Multiplayer collaborative solving
- Share solutions with others

Phase 5: Platform Expansion

- Mobile responsive design
- PostgreSQL/MySQL dialect options
- Offline mode with local storage
- Teacher dashboard for classroom use

---

16. Resume and Interview Highlights

Resume Bullet Points

Use these for your resume (adjust as needed):

- Designed and implemented interactive SQL learning game with 7 progressive levels covering SELECT through Window Functions, processing real-time database queries
- Built secure query execution engine featuring SQL injection prevention, keyword blocking, and read-only database enforcement
- Created immersive 3D detective environment using Three.js with interactive objects, raycasting-based click detection, and atmospheric lighting
- Developed Flask REST API with endpoints for query validation, answer verification using result set comparison, and session-based progress tracking
- Architected normalized SQLite database schema with 7 interconnected tables modeling realistic crime investigation data

Interview Explanation Script

When asked about this project:

"SQL Detective is a gamified SQL learning platform I built to teach database querying through crime investigation. Players solve a bank heist by writing SQL queries in an immersive 3D detective office.

The backend uses Flask with a security-first architecture. Every query passes through validation that blocks dangerous keywords and ensures read-only execution. I compare result sets rather than query strings so multiple correct formulations are accepted.

The frontend uses Three.js for the 3D environment. Players click objects like the desk to read case files, the evidence board to see table schemas, and the computer to write queries. Raycasting handles click detection.

The game has seven levels progressing from basic SELECT to window functions, each teaching a specific SQL concept through narrative context."

Sample Interview Questions and Answers

Q: How did you handle SQL injection prevention?

A: I implemented multiple security layers. First, queries must start with SELECT or WITH. Second, I block a list of dangerous keywords like DROP, DELETE, and UPDATE. Third, I detect multiple statements by looking for semicolons mid-query. Finally, the database connection itself is read-only at the SQLite level.

Q: How do you verify if a query answer is correct?

A: I execute both the user's query and the expected query, then compare the result sets. I normalize the data by converting rows to tuples, then compare as sets for order-insensitive matching or as lists for order-sensitive levels. This allows multiple valid query formulations to be accepted.

Q: Why did you choose Three.js for the frontend?

A: Three.js is the industry standard for WebGL abstraction. It runs in browsers without plugins, has excellent documentation, and supports procedural geometry so I could build the environment without external 3D assets. The raycaster API made click detection straightforward.

Q: What was the most challenging part of this project?

A: Balancing security with usability. I needed to allow arbitrary SELECT queries while preventing any data modification or injection attacks. The solution was layered defenses: validation-time keyword blocking, pattern matching for known attack vectors, and connection-time read-only enforcement.

Document Information

Document Title: SQL Detective Technical Documentation
Version: 1.0
Last Updated: January 2026
Total Sections: 16
Intended Audience: Technical Interviewers, Developers, Portfolio Reviewers

---

This documentation is designed for professional sharing. The project demonstrates proficiency in full-stack development, database design, security engineering, 3D graphics programming, and educational game design.