SQL Detective: Technical Documentation

A Portfolio-Ready SQL Learning Game

Executive Summary

SQL Detective is a full-stack web application that teaches SQL through crime investigation gameplay. Players solve a bank heist case by writing SQL queries against realistic crime databases in an immersive 3D noir detective environment.

Attribute	Details
Project Type	Full-Stack Web Application
Duration	Single Development Sprint
Tech Stack	Flask (Python), Three.js, SQLite
SQL Concepts	7 Progressive Levels
Target Users	SQL Learners (Beginner to Advanced)

Table of Contents

---

1. Problem Statement

Challenge

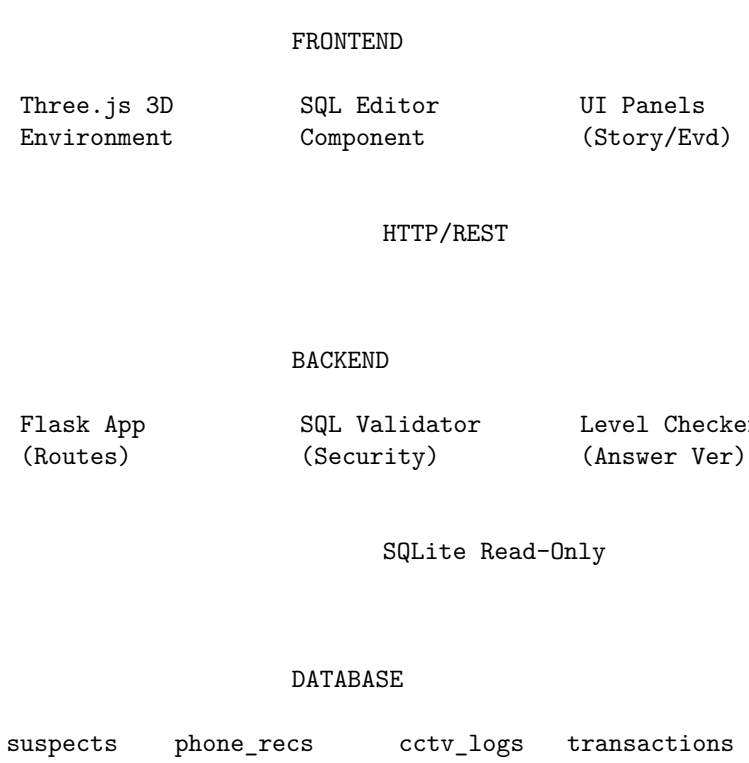
Traditional SQL learning methods (textbooks, tutorials) lack engagement and real-world context. Students struggle to retain concepts without hands-on practice with meaningful data.

Solution

An gamified learning experience that: - Provides **contextual motivation** through crime investigation narratives - Offers **immediate feedback** on query correctness - Progressively introduces **7 core SQL concepts** - Creates an **immersive environment** that makes learning memorable

---

2. Solution Architecture



Technology Choices

Component	Technology	Rationale
Backend	Flask (Python)	Lightweight, easy to deploy, excellent SQLite integration

Component	Technology	Rationale
Database	SQLite	Zero-config, file-based, perfect for read-only educational app
3D Engine	Three.js	Industry standard, runs in browser, no plugins required
Styling	Vanilla CSS	Maximum control, no framework dependencies

---

3. Database Design

Entity-Relationship Diagram

```
   SUSPECTS              LOCATIONS           CRIME_SCENES

id (PK)               id (PK)            location_id
name                  name                  case_number
age                   type                  crime_type
occupation            address               date_time
criminal_rec



PHONE_RECORDS           CCTV_LOGS           TRANSACTIONS

caller_id(FK)         person_id(FK)         account_id(FK)
receiver_id           location_id           amount
timestamp             timestamp             timestamp
duration              confidence            type
```

Data Volume

Table	Records	Purpose
suspects	10	Persons of interest profiles
locations	10	City locations (bank, diner, etc.)
phone_records	25+	Call/SMS logs with timestamps
bank_transactions	18+	Financial activity records

Table	Records	Purpose
cctv_logs	18+	Surveillance sightings
crime_scenes	3	Crime incident details

Design Decisions

1. **Realistic Relationships**: Foreign keys link suspects to their activities
2. **Temporal Data**: All records include timestamps for time-based queries
3. **Graduated Complexity**: Early levels use 1 table, later levels require JOINs across 3+ tables

---

4. Backend Implementation

Service Architecture

```
backend/
  app.py                  # Application entry, route registration
  config.py               # Configuration management
  services/
      sql_validator.py    # Query security validation
      query_executor.py   # Safe query execution
      level_checker.py    # Answer verification logic
  levels/
      level_config.py     # 7 level definitions
  routes/
      game.py             # Level/progress endpoints
      query.py            # Query execution endpoints
```

Key API Endpoints

Method	Endpoint	Purpose
GET	`/api/game/levels`	Retrieve all level metadata
GET	`/api/game/levels/<id>`	Get specific level details
POST	`/api/query/execute`	Execute user's SQL query
POST	`/api/query/check`	Verify answer correctness
GET	`/api/game/tables`	Get available tables for level
GET	`/api/game/progress`	Get player progress state

Query Validation Pipeline

```python
def validate_query(query: str) -> Tuple[bool, str]:
    # 1. Check for empty input
```

```
    # 2. Verify starts with SELECT or WITH
    # 3. Scan for blocked keywords (DROP, DELETE, etc.)
    # 4. Detect multiple statements
    # 5. Check query length limits
    # 6. Scan for suspicious patterns
    return (is_valid, error_message)
```

---

5. Frontend Implementation

3D Scene Components

```
DetectiveRoom
  Room Structure
      Floor (wood planks texture)
      Walls (dark wood panels)
      Ceiling
  Interactive Objects
      Desk (opens case file panel)
      Evidence Board (shows table schemas)
      Computer Terminal (opens SQL editor)
  Ambient Objects
      Bookshelf with colored books
      Filing cabinet
      Venetian blinds with light rays
  Effects
       Desk lamp (warm point light)
       Monitor glow (green point light)
       Dust particles (animated points)
```

UI Component Hierarchy

```
index.html
  Loading Screen (animated intro)
  HUD
      Level Indicator
      Action Buttons (Help, Menu)
  Panels
      Story Panel (case narrative)
      Evidence Panel (table viewer)
      SQL Editor Panel
          Editor (textarea + line numbers)
          Action Buttons (Execute, Submit)
          Results Display (table format)
  Modals
       Level Complete
```

```
Help
Menu
```

Interaction Flow

```
User clicks 3D object


Raycaster detects intersection


Object userData.action identified

          "open_story" → Show Story Panel
          "open_evidence" → Show Evidence Panel
          "open_terminal" → Show SQL Editor
```

---

6. Security Implementation

Threat Model

Threat	Mitigation
SQL Injection	Keyword blocking + read-only mode
Data Modification	SELECT-only queries enforced
DoS via expensive queries	5-second timeout limit
Schema discovery	Limited table access per level

Blocked SQL Keywords

```
BLOCKED_KEYWORDS = [
    'DROP', 'DELETE', 'UPDATE', 'INSERT', 'ALTER',
    'CREATE', 'TRUNCATE', 'GRANT', 'REVOKE', 'EXEC',
    'EXECUTE', 'PRAGMA', 'ATTACH', 'DETACH', 'VACUUM'
]
```

Read-Only Enforcement

```
# Connect with URI mode and read-only flag
conn = sqlite3.connect(
    f'file:{db_path}?mode=ro',
    uri=True,
    timeout=5
)
```

---

7. Game Design & Pedagogy

Level Progression

Level	Title	SQL Concepts	Scaffolding
1	The Missing Witness	SELECT, WHERE	Single table, basic filters
2	The Midnight Call	BETWEEN, ORDER BY, LIMIT	Time ranges, sorting
3	The Connection	INNER JOIN	Multi-table relationships
4	The Pattern	GROUP BY, HAVING	Aggregation, filtering groups
5	The Money Trail	Subqueries	Nested queries, AVG
6	The Movement	CTEs	WITH clause, organized queries
7	The Final Piece	Window Functions	RANK, LAG, PARTITION BY

Pedagogical Approach

1. **Narrative Context**: Each query solves a story problem
2. **Immediate Feedback**: Correct/incorrect with helpful hints
3. **Exploration Mode**: Execute any query to explore data
4. **Progressive Disclosure**: Tables unlock as levels progress
5. **No Penalty for Experimentation**: Unlimited attempts

---

8. Technical Challenges & Solutions

Challenge 1: 3D Object Interaction

Problem: Detecting clicks on 3D objects with raycasting **Solution**: Store action metadata in `userData`, traverse mesh hierarchy for hit detection

```
object.userData.action = 'open_terminal';
object.userData.interactive = true;
```

Challenge 2: Answer Verification Without Exact Match

Problem: Multiple correct queries can produce same result **Solution**: Compare result sets, not query strings; ignore row order

```
user_set = set(tuple(row) for row in user_result)
expected_set = set(tuple(row) for row in expected_result)
return user_set == expected_set
```

Challenge 3: Session Management Without Database

Problem: Track player progress without user accounts **Solution**: Flask session with client-side cookies

```
session['current_level'] = level_id
session['completed_levels'] = [1, 2, 3]
```

---

9. Testing & Verification

Test Results

Test Category	Status	Notes
Database Initialization	Pass	Schema + seed data load correctly
SQL Validation	Pass	Blocked keywords rejected
Query Execution	Pass	Results return in <100ms
3D Scene Rendering	Pass	All objects visible
Object Interaction	Pass	Panels open on click
Level Progression	Pass	Correct answers unlock next level

Browser Compatibility

Browser	Tested	WebGL Support
Chrome		Full
Firefox		Full
Edge		Full
Safari		Requires testing

---

10. Future Roadmap

Phase 2 Enhancements

- ☐ Sound effects and ambient audio
- ☐ User accounts with saved progress
- ☐ Leaderboard system

Phase 3 Features

- ☐ Additional crime case scenarios
- ☐ PostgreSQL/MySQL dialect modes
- ☐ Multiplayer collaborative solving

Phase 4 Scale

- ☐ Mobile responsive design
- ☐ Query performance analytics
- ☐ Teacher dashboard for classrooms

---

11. Resume Highlights

Copy these bullet points for your resume:

SQL Detective Game | *Full-Stack Developer* | 2024

- Designed and implemented interactive SQL learning game covering 7 progressive concepts from SELECT to Window Functions, processing user queries in real-time
- Built secure query execution engine with SQL injection prevention, keyword blocking, and read-only database enforcement
- Created immersive 3D noir detective environment using Three.js with interactive objects, dynamic lighting, and particle effects
- Developed Flask REST API for query validation, answer verification with fuzzy matching, and session-based progress tracking
- Architected normalized database schema with 7 tables modeling realistic crime investigation data relationships

---

Appendix A: File Structure

```
SQL Based Detective Game/
  backend/
      app.py
```

```
    config.py
    database/
        schema.sql
        seed_data.sql
        detective_game.db
    routes/
        game.py
        query.py
    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
```

Appendix B: Sample Queries

Level 1 - Expected Answer

```sql
SELECT * FROM suspects
WHERE age > 30 AND criminal_record = 1;
```

Level 3 - Expected Answer

```sql
SELECT s.name, s.occupation, l.name as location, c.timestamp
FROM suspects s
INNER JOIN cctv_logs c ON s.id = c.person_id
INNER JOIN locations l ON c.location_id = l.id
WHERE l.name = 'Downtown Bank';
```

Level 7 - Expected Answer

```sql
SELECT
    account_id, amount, timestamp,
    RANK() OVER (PARTITION BY account_id ORDER BY amount DESC) as amount_rank,
    LAG(amount) OVER (PARTITION BY account_id ORDER BY timestamp) as prev_amount,
    amount - LAG(amount) OVER (PARTITION BY account_id ORDER BY timestamp) as amount_change
FROM bank_transactions
WHERE account_id IN (SELECT id FROM suspects WHERE criminal_record = 1);
```

Document Version: 1.0
Last Updated: January 2026
Author: Ananya B

This documentation is intended for sharing with interviewers, portfolio reviewers, and self-reference. The project demonstrates proficiency in full-stack development, database design, security implementation, and interactive 3D web development.