🚀 Gemini SQL Assistant - Complete Integration Guide

📁 Project Structure

```
gemini-sql-assistant/
│
├── sqlm.py                 # AI Reasoning Core (Gemini Integration)
├── schema_awareness.py     # Schema Management Module
├── command_processor.py    # CLI Command Processor
├── db_executor.py          # Database Execution Module (NEW)
├── streamlit_app.py        # Beautiful Frontend (NEW)
│
├── schema.txt              # Auto-generated schema file
├── schema_metadata.json    # Schema versioning metadata
│
├── .env                    # Environment variables
├── requirements.txt        # Python dependencies
└── README.md               # Documentation
```

📦 Installation

Step 1: Install Dependencies

Create a `requirements.txt` file:

```txt
streamlit>=1.28.0
pandas>=2.0.0
plotly>=5.17.0
python-dotenv>=1.0.0
sqlglot>=19.0.0
sqlparse>=0.4.4
google-generativeai>=0.3.0
pymysql>=1.1.0
psycopg2-binary>=2.9.0
```

Install all dependencies:

```bash
```

```
pip install -r requirements.txt
```

Step 2: Configure Environment Variables

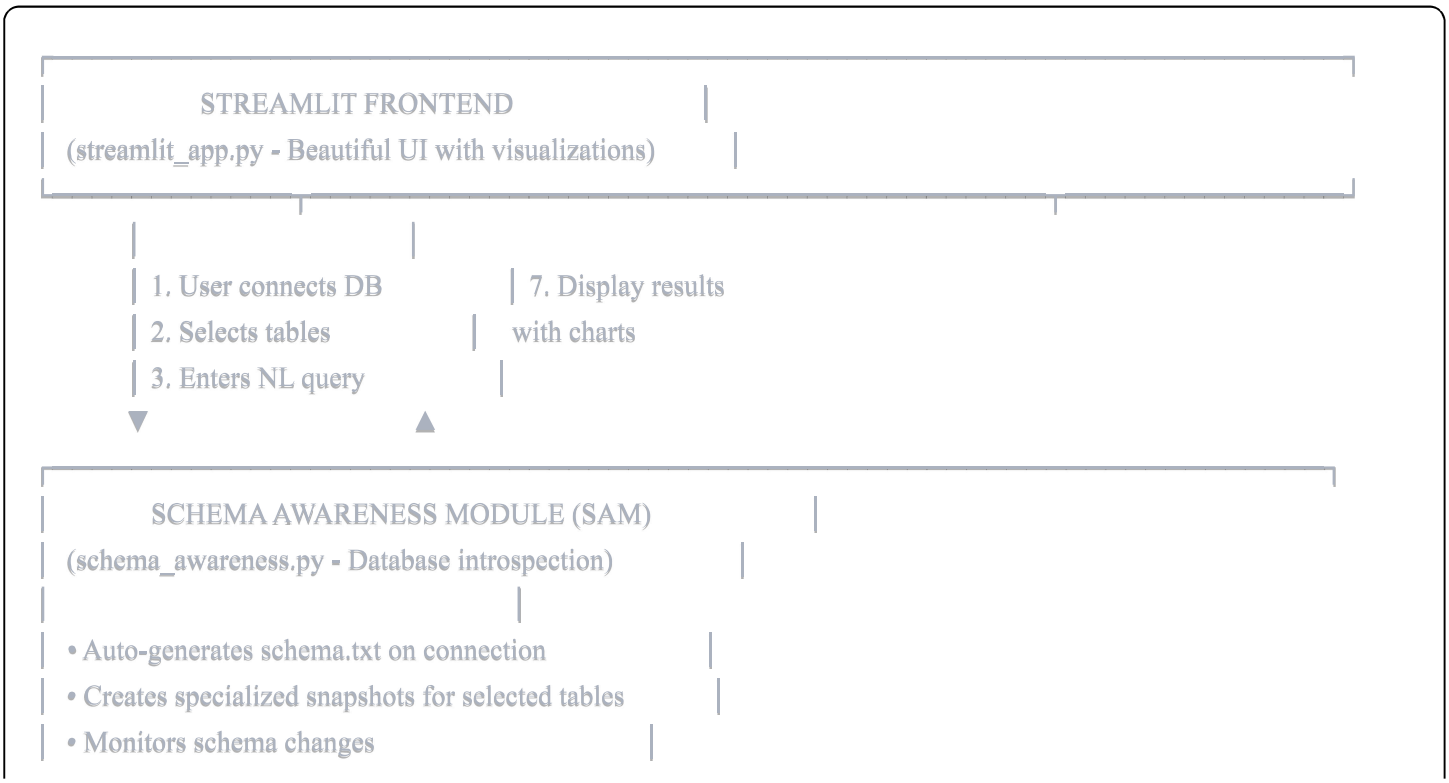
Create a `.env` file in the root directory:

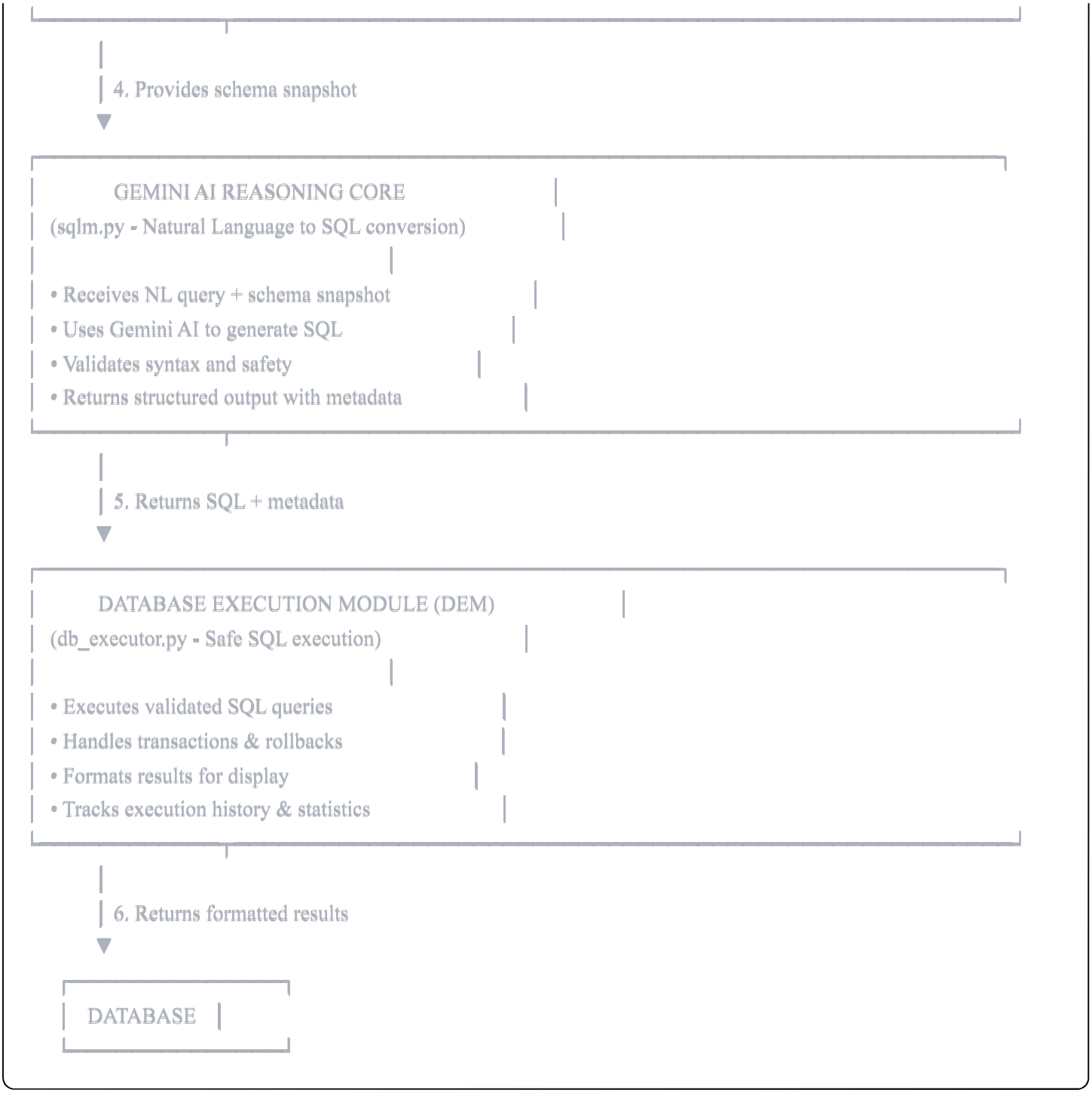
```
env

# Gemini AI Configuration
GEMINI_API_KEY=your_gemini_api_key_here
GEMINI_MODEL=models/gemini-2.5-pro
GEMINI_MAX_TOKENS=8192

# SQL Configuration
DEFAULT_DIALECT=mysql
MAX_SCHEMA_PROMPT_CHARS=14000
```

Get your Gemini API Key:

1. Visit Google AI Studio
2. Create a new API key
3. Copy and paste it into your `.env` file

🔄 System Architecture & Data Flow





🎯 How Each Module Works Together

Module 1: Schema Awareness Module (SAM)

Purpose: Automatically discovers and manages database schema

Key Functions:

- `connect_database()` - Connects to MySQL/PostgreSQL/SQLite
- `generate_full_schema()` - Scans DB and creates schema.txt

- `create_specialized_snapshot()` - Generates table-specific schemas
- `detect_schema_changes()` - Monitors for schema modifications

Integration Points:

- Frontend calls `connect_database()` when user connects
- Frontend calls `create_specialized_snapshot()` for selected tables
- Provides schema to Reasoning Core

Module 2: Gemini AI Reasoning Core

Purpose: Converts natural language to SQL using Gemini AI

Key Functions:

- `update_schema()` - Updates schema context
- `generate()` - Converts NL to SQL with safety checks
- `_call_llm()` - Communicates with Gemini API

Integration Points:

- Receives schema from SAM
- Receives NL query from Frontend
- Sends SQL output to Database Executor

Module 3: Database Execution Module (DEM)

Purpose: Safely executes SQL and formats results

Key Functions:

- `execute_query()` - Runs SQL with safety checks
- `format_results_for_display()` - Prepares data for frontend
- `get_statistics()` - Provides execution metrics

Integration Points:

- Receives SQL from Reasoning Core
- Executes against database connection from SAM

- Returns formatted results to Frontend

Module 4: Streamlit Frontend

Purpose: Beautiful UI for user interaction

Key Features:

- Database connection interface
- Table selection (all/some/none)
- Natural language query input
- Real-time SQL generation
- Data visualization with charts
- Query history tracking

Integration Points:

- Uses SAM for connection & schema
- Uses Reasoning Core for SQL generation
- Uses DEM for execution & results

🚀 Running the Application

Option 1: Run the Streamlit Frontend (Recommended)

```
bash

streamlit run streamlit_app.py
```

The app will open in your browser at `http://localhost:8501`

Option 2: Use the CLI Interface

```
bash

python command_processor.py
```

Option 3: Test Individual Modules

Test Schema Awareness:

```bash
python schema_awareness.py --db-type sqlite --database test.db
```

Test Reasoning Core:

```bash
python sqlm.py --schema schema.txt
```

Test with Mock Data:

```bash
python sqlm.py --run-test
```

📖 Usage Examples

Example 1: Basic SELECT Query

1. **Connect to database** (sidebar)
2. **Select tables:** "students" table
3. **Enter NL query:** "Show me all students whose surname starts with A"
4. **Click:** "Generate & Execute SQL"
5. **Result:**

```sql
SELECT * FROM students WHERE surname LIKE 'A%';
```

Example 2: Aggregate Query

1. **Select tables:** "classes" table
2. **Enter NL query:** "Count how many classes exist"
3. **Result:**

```sql
SELECT COUNT(*) FROM classes;
```

Example 3: Create Table (No table selection needed)

1. **Select:** "No Tables (CREATE, DROP, etc.)"
2. **Enter NL query:** "Create a new table called courses with columns id, name, credits"
3. **Enable:** "Allow Destructive Operations"
4. **Result:**

```sql
CREATE TABLE courses (
  id INT PRIMARY KEY,
  name VARCHAR(255),
  credits INT
);
```

Example 4: Join Query

1. **Select tables:** "students" and "classes"
2. **Enter NL query:** "Show me all students with their class names"
3. **Result:**

```sql
SELECT s.*, c.classname
FROM students s
JOIN classes c ON s.class_id = c.id;
```

🛡️ Safety Features

Built-in Safety Checks

1. **Destructive Operation Detection**
   - Automatically detects INSERT, UPDATE, DELETE, DROP, ALTER
   - Requires explicit user permission via checkbox
   - Blocks execution by default
2. **SQL Validation**
   - Uses `sqlglot` for syntax parsing
   - Detects potential Cartesian products

- Validates against schema

3. **Dry Run Mode**
   - Preview SQL without executing
   - Test queries safely
4. **Transaction Management**
   - Automatic rollback on errors for destructive ops
   - Prevents partial data corruption
5. **Schema Verification**
   - Only uses tables/columns that exist
   - Prevents SQL injection through schema validation

🎨 Frontend Features

User Interface

- **Modern Gradient Design** - Beautiful purple gradient theme
- **Responsive Layout** - Works on all screen sizes
- **Dark Sidebar** - Easy-to-read connection info
- **Smooth Animations** - Hover effects and transitions

Data Visualization

- **Automatic Chart Generation** - Bar, Line, Scatter, Pie charts
- **Interactive Tables** - Sortable, filterable dataframes
- **CSV Export** - Download results instantly
- **Real-time Updates** - Live query execution

Statistics Dashboard

- **Success Rate** - Track query performance
- **Execution Time** - Monitor query speed
- **Query History** - Review past queries
- **Visual Metrics** - Pie charts for status breakdown

Change AI Model
Edit `.env`:

```
env

GEMINI_MODEL=models/gemini-1.5-pro  # or other available models
```

Adjust Schema Context Size

Edit `.env`:

```
env

MAX_SCHEMA_PROMPT_CHARS=20000  # Increase for larger schemas
```

Modify Frontend Theme

Edit `streamlit_app.py` - Look for the CSS in `st.markdown()`:

```
python

# Change gradient colors
background: linear-gradient(135deg, #YOUR_COLOR_1 0%, #YOUR_COLOR_2 100%);
```

🐛 Troubleshooting

Issue: "Gemini API not available"

Solution:

- Check your `GEMINI_API_KEY` in `.env`
- Verify internet connection
- The system will use mock LLM if API unavailable

Issue: "Schema file not found"

Solution:

- Ensure you've connected to a database first
- SAM will auto-generate `schema.txt` on connection

Issue: "Connection failed"

Solution:

- Verify database credentials
- Check if database server is running
- Ensure correct port numbers (3306 for MySQL, 5432 for PostgreSQL)

Issue: "No tables found"

Solution:

- Verify database has tables
- Check user permissions
- Try refreshing schema

📊 Performance Tips

1. **For Large Databases:**
   - Use specialized snapshots (select specific tables)
   - Don't select "All Tables" unless needed
2. **For Complex Queries:**
   - Be specific in natural language
   - Mention column names if known
   - Use "Show me" or "Get" for clarity
3. **For Best Results:**
   - Start with simple queries to test
   - Enable destructive ops only when needed
   - Use dry run mode to preview SQL

🔐 Security Best Practices

1. **Never commit `.env` file** to version control
2. **Use environment variables** for all credentials
3. **Enable destructive operations** only when necessary

4. **Review generated SQL** before executing
5. **Use read-only database users** for SELECT-only access
6. **Keep API keys secure** and rotate regularly

📚 Additional Resources

- Gemini API Documentation
- Streamlit Documentation
- SQLGlot Documentation
- Project Repository

🎉 You're Ready!

Your Gemini SQL Assistant is now fully integrated and ready to use. Start by:

1. Running `streamlit run streamlit_app.py`
2. Connecting your database
3. Asking your first natural language question!

Enjoy the magic of AI-powered SQL! ✨🤖