User Journey

1. **ANTLR Implementation :**
   - Began by understanding how SQL grammar is defined and parsed using ANTLR's SQLite.g4.
   - Validated grammar with simple SQL queries and progressively tested more complex ones.
   - Ensured the parser could handle both single and multi-level nested queries accurately.
2. **Traversal :**
   - Implemented recursive tree traversal to explore SQL parse trees using ANTLR's parser rules.
   - Identified select_core nodes to isolate and extract complete SELECT statements.
   - Differentiated between outer and inner queries by maintaining depth and context tracking.
   - Designed clean extraction functions to reconstruct readable SQL from token streams.
3. **Cache Mechanism :**
   - Implemented a Python dictionary-based cache to store and retrieve query plans efficiently.
   - Normalized SQL queries into structural patterns for cache key consistency.
   - Differentiated cache hits and misses, logging metrics for accuracy and performance tracking.
   - Simulated real-world behavior with dummy plan generation, complexity scoring, and planning delays.

Developer Experience and Challenge :

1. **Handling corner cases in Sub-queries :**
   a. Parenthesis tracking and rule-based traversal used to carefully detect boundaries of nested sub-queries.
   b. Ensured that even malformed or incomplete queries didn't break the parser or normalization flow but leads to improper tree diagram.
2. **Literal replacement Challenges :**
   a. Initially I used regex to replace literals and getting expected results but regex sometimes fails to detect exact foundries and leads to wrong parenthesis tracking.
   b. Solve this issue after using ANTLR as it trackes each element as Node in a tree.
   c. By doing Depth First Search in tree we can detect which query has nested (inner) statements that need to be normalize.
3. **Performance Comparison :**
   a. Simulated dummy delay to show complex nested queries take longer time for plan generation after cache miss.
   b. Test suite of ~20 queries which included simple to complex nested queries to check how cache mechanism saves time by skipping expensive operation of plan generation.