List of API Called at Home Screen and Canine Screen

API_Home_Canine-screen.txt

Analysis of Queries Shared

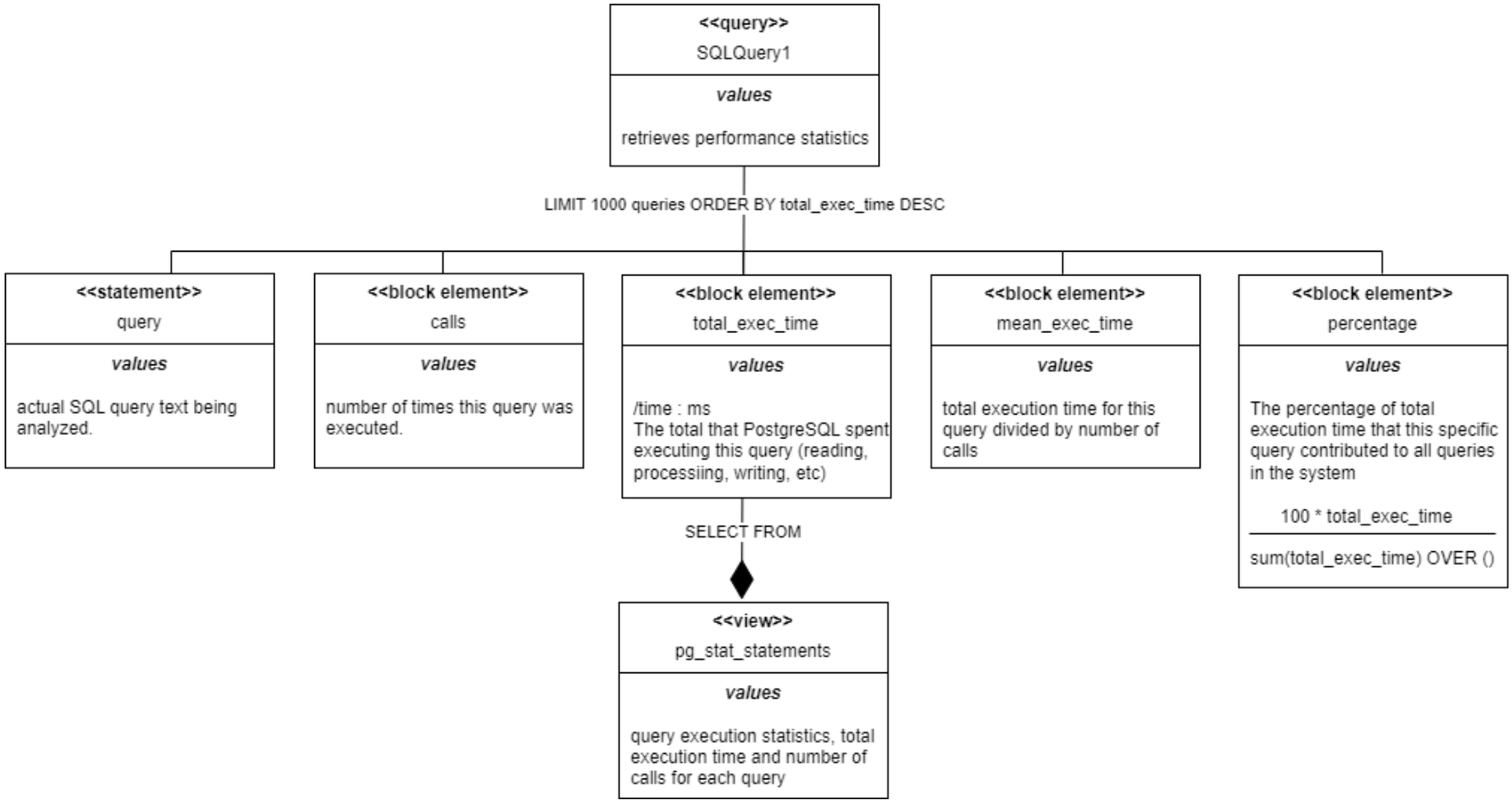
Query

SQLQuery1.sql

Purpose

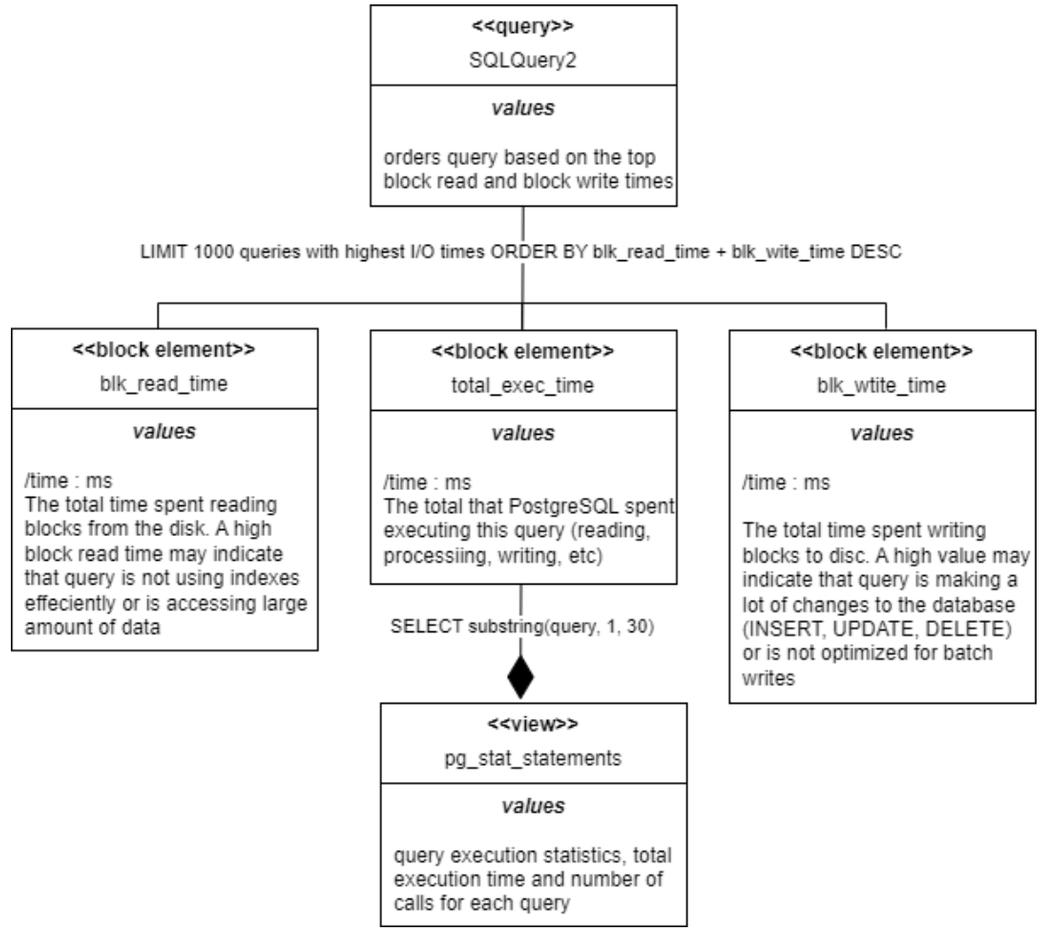
This query retrieves performance statistics for 1000 most time-consuming queries stored in the `pg_stat_statments` view in postgresSQL.

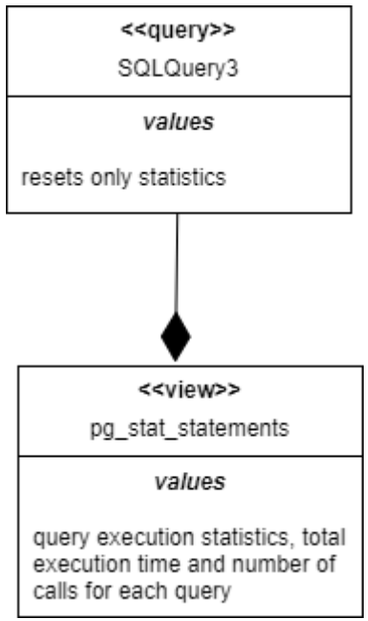
Flow

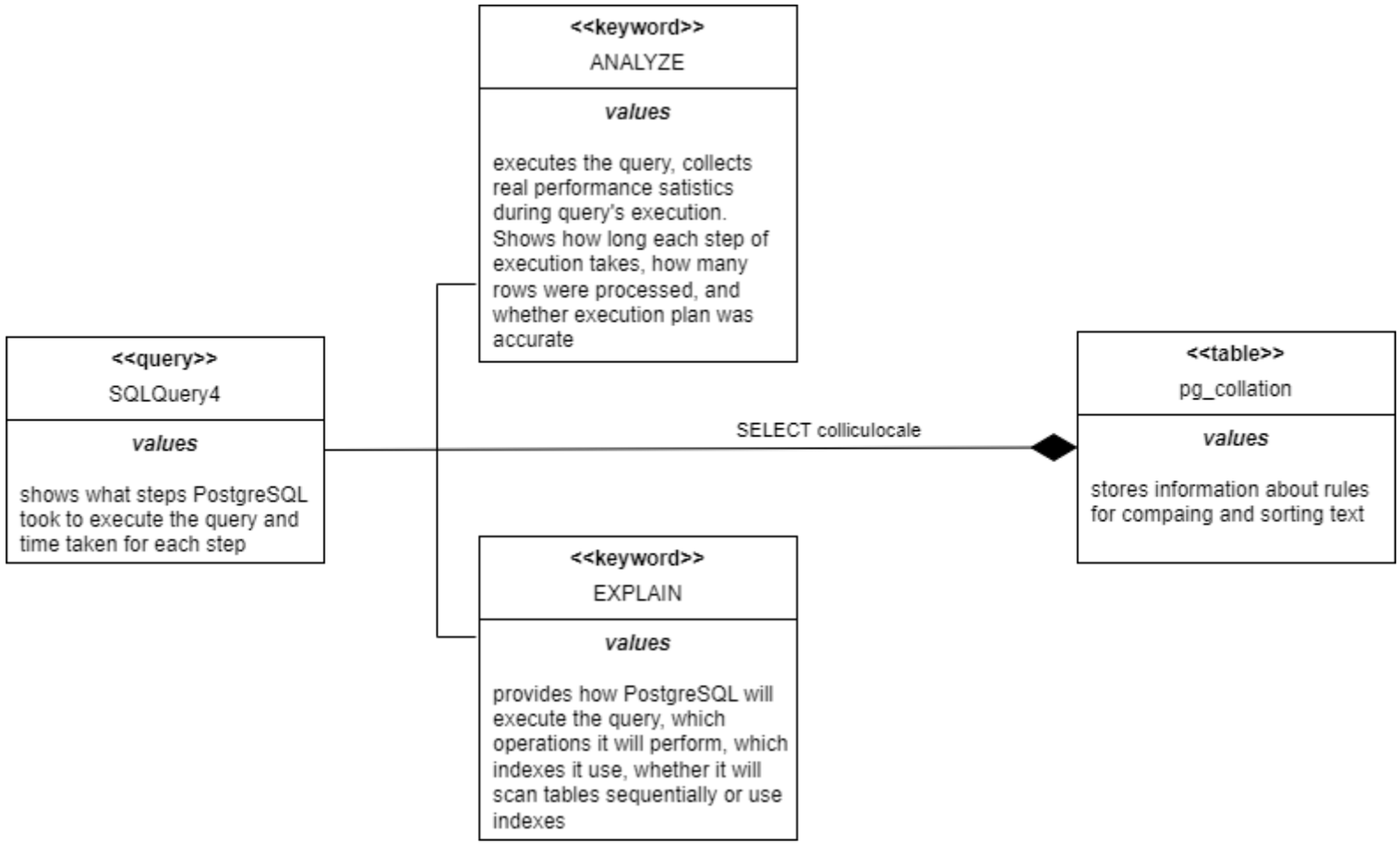


Output Example:

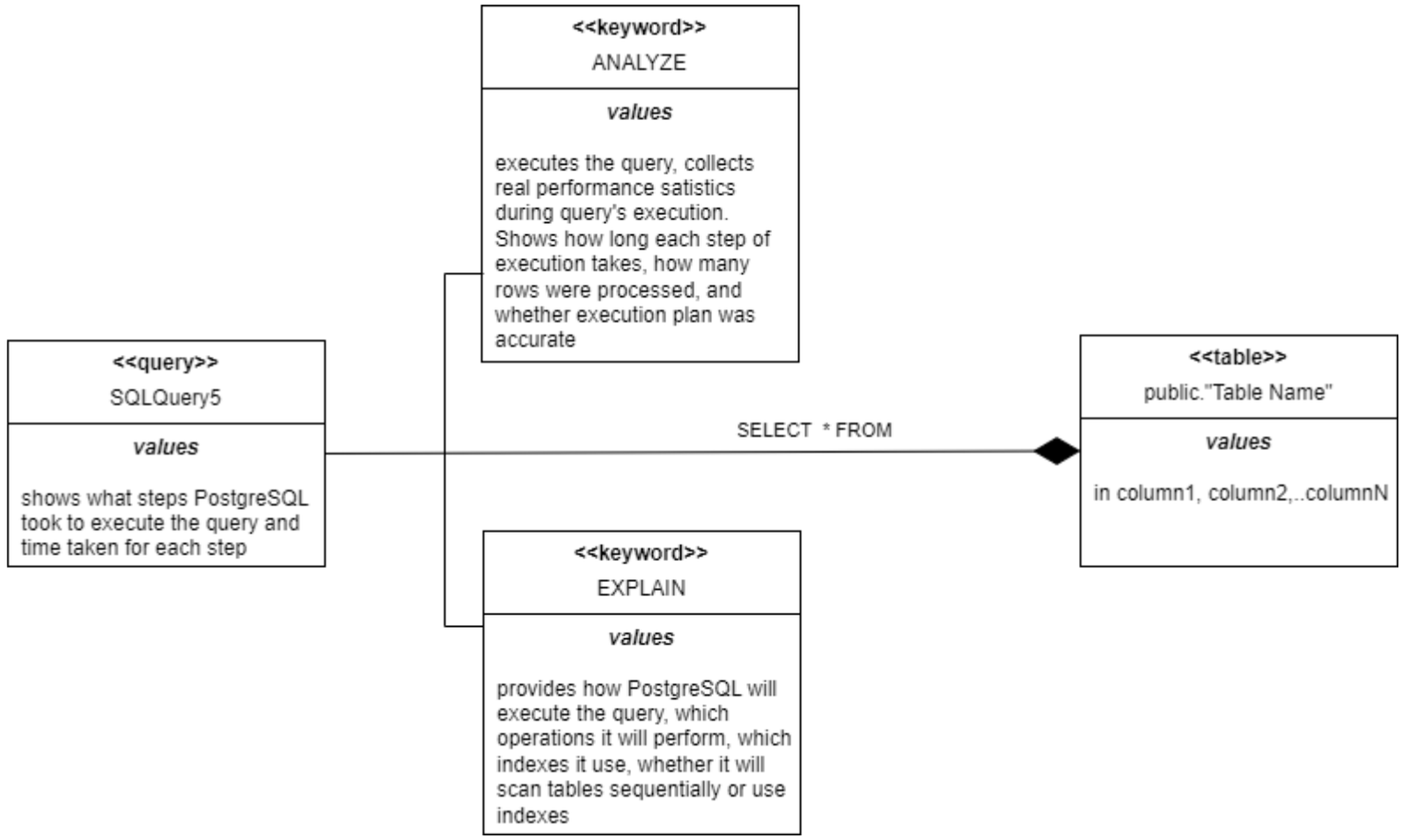
query	calls	total_time	mean_time	Percentage
SELECT * FROM “PetTrainingLog” where len_Calories = 3	500	1500.32	3.00	25.40
INSERT INTO PetDietary (PetID, IsFav)	1000	1300.72	1.30	22.04

<table>
<tr><td>Query</td><td> 
SQLQuery2.sql</td></tr>
<tr><td>Purpose</td><td>This query helps in identifying top 1000 queries that cause most disk I/O operations</td></tr>
<tr><td>Flow</td><td> 
</td></tr>
<tr><td>Output Example:</td><td>
<table>
<tr><th>query</th><th>total_time</th><th>blk_read_time</th><th>blk_wite_time</th></tr>
<tr><td>SELECT * FROM “AboutUs” where len_ourheader = 12</td><td>12000.45</td><td>8000.12</td><td>1000.55</td></tr>
<tr><td>INSERT INTO PetScheduler (len_ID, IsFav)</td><td>9500.25</td><td>5000.20</td><td>4000.80</td></tr>
</table>
</td></tr>
</table>

Query	SQLQuery3.sql
Purpose	This query is used to reset the statistics collected by pg_stat_statements extension in PostgreSQL, giving “clean slate” for monitoring queries.
Flow	
Output	Just a success message (or an error if something ges wrong)
When to use?	- After performance tuning: If you’ve optimized some queries or made changes to your databse schema - Periodic monitoring: You may periodically reset stats (e.g daily or weekly) to keep track of recent query performance and optimization - After large scale database operations: If you’ve loaded a large dataset or completed a maintenance task (like vaccuming or indexing) - Testing: If you’re running perforamance tests and want to clear old stats

Query	SQLQuery4.sql
Purpose	This query retrieves the colliculate from the system catalog table pg_collation. and with EXPLAIN ANALYZE, it also gives detailed information about the execution plan and actual performance. This allows you to see how the query is run internally by PostgreSQL, which can help you understand its performance characteristics
Flow	
Output Example	Seq Scan on pg_collation (cost=0.00..1.03 rows=3 width=64) (actual time=0.010..0.012 rows=3 loops=1) Output: colliculocale Planning Time: 0.086 ms Execution Time: 0.034 ms
Explanation of output	Seq Scan on pg_collation: This means that PostgreSQL decided to perform a sequenteial scan over the pg_collation table. Since pg_collation is a system catalog table Cost=0.00...1.03 : This is an estimate of the cost for executing the query, where 0.00 is the startup cost (the cost of getting the first row), and 1.03 os total cost of executing the query row=3: This shows that PostgreSQL estiamted the query would return 3 rows. actual time = 1.010..0.012; This is the actual time it took to execute this part of the query. It started 0.0010 ms and finished at 0.012 ms loops=1: This indicates how many times PostgreSQL executed this part of the query.

	Output: Colliculate: this shows that the output of this query will be the colliculocale column. Planning Time: 0.086 ms: This is the time PostgreSQL spent planning the query (deciding which operation to perform, whether to use indexes, etc) Execution Time: 0.034 ms: This is the total time it took to actually execute the query, incluing fetching the data

Query	SQLQuery5.sql
Purpose	This query shows whether PostgreSQL performed a sequential scan or index scan (if there's an index and a WHERE clause, which isn't used here), how many rows were processed and how long query took to execute.
Flow	
Output	Seq Scan on public."Table Name" (cost=0.00..35.50 rows=2550 width=64) (actual time=0.023..0.205 rows=2550 loops=1) Output: column1, column2, column3, ... Planning Time: 0.145 ms Execution Time: 0.310 ms
Explanation of output	Seq Scan on public."Table Name" : Seq Scan (Sequential Scan) indicates that PostgreSQL is scanning the table row by row. Since the query is retreiving all rows and column (SELECT *), PostgreSQLis doing a full table scan public."Table Name" refers to the table being scanned

	(cost=0.00..35.50 rows=250 width=64) : cost=0.00..35.50:These numbers represent the estimated execution cost. The first number (0.00) is the estimated startup cost. The first number (0.00) is the estimated start up cost (how much work it takes to get first row), the second number (35,50) is the total cost of retreiving all rows. rows=2550: this is the estimate of how many rows the query will return (2,550 in this case). width=64: This indicates the estimated width (in bytes) of each row, including all the columns that will be returned. In this example PostgreSQL estimates that each row will be 64 byte wide. (actual time=0.023..0.205 rows=2550 loops=1): actual time =0.023..0.205 : This shows the actual time taken to execute the scan. The first number (0.023 ms) is the time to start fetching rows, and the second number (0.205 ms) is the time to complete fetching all rows rows=2550: this is the actual number of rows retrieved loops=1: This indicates how many times this step was executed (in this case, it was executed once). Output:column1, column2, column3,... This line shows which columns are being returned by the query. Planning Time: 0.145 ms: This is the time PostgreSQL spent planning the query, which involves analyzing which scan method to use (sequential scan, index scan, etc) Execution Time: 0.310 ms: This is the total execution time of the query from start to finish, including scanning the rows and returning them

System tables to help monitor and optimize performance

1. pg_stat_activity – Monitoring Active Queries

Purpose	Shows information about the current activity of queries that are running in the database
Useful for	Identifying slow queries, blocking queries, and long-running transactions in real time
Example query	 SQLQuery6.sql
Columns of interest	- query: The current SQL query being executed - state: The state of query (active, idle, idle in transaction) - wait_event: Indicates if the query is waiting on locks or I/O

2. pg_stat_user_tables – Table Level Statistics

Purpose	Provides performance statistics for user tables in the database, such as number of scans, index scans, rows head, rows inserted, etc
Useful for	Analyzing performance of tables used by queries and identifying bottlenecks
Example query	 SQLQuery7.sql
Columns of interest	- seq_scan: Number of sequential scans on the table. - idx_scan: Number of index scans on the table. - n_tup_ins, n_tup_upd, n_tup_del: Number of rows inserted, updated, and deleted

3. `pg_stat_user_indexes` – Index Level Statistics

Purpose	Provides statistics on how often indexes are used, including the number of index scans, number of index tuples fetched, and the number of index rows returned
Useful for	Analyzing index performance and identifying unused or inefficient indexes
Example query	 SQLQuery8.sql
Columns of interest	- idx_scan: Number of scans performed on the index - idx_tup_read: Number of index entries read - idx_tup_fetch: Number of rows by index scans

4. `pg_statio_user_tables` – I/O Statistics for Tables

Purpose	Provides detailed statistics about the I/O behavior (disk reads, buffer hits, etc) for user tables
Useful for	Identifying tables with heavy disk usage or buffer hits, which can affect performance
Example query	 SQLQuery9.sql
Columns of interest	heap_blks_read: Number of disk blocks read for the table. heap_blks_hit: Number of buffer hits for the table (when data was found in memory) idx_blks_read, idx_blks_hit: similar statistics for indexes

5. `pg_statio_user_indexes` – I/O Statistics for Indexes

Purpose	Provides I/O statistics for indexes, showing how often index blocks are read from disk and found in memory
Useful for	Identifying indexes with high I/O, which could be candidates for optimization
Example query	 SQLQuery10.sql
Columns of interest	idx_blks_read: Number of disk blocks read for the index idx_blks_hit: Number of buffer hits for the index

6. pg_size_pretty(pg_total_relation_size('table name')) – Table Size

Purpose	Provides the total disk space used by a table, including all indexes and toast data.
Useful for	Checking the size of table in the database, when dealing with large tables for performance
Example query	 SQLQuery11.sql

Columns of interest	-

7. `pg_locks` – Lock Monitoring

Purpose	Shows information about locks held by transactions or queries
Useful for	Investigating performance issues caused by blocking queries or deadlocks
Example query	 SQLQuery12.sql
Columns of interest	locktype: Type of Lock (relation, low, etc) relation: The table or relation involved mode: The type of lock (e.g., RowExclusiveLock) granted: Whether the lock has been granted or is waiting

8. `pg_proc` – Information About Functions

Purpose	Contains meta about all stored functions, including their name, return type, and argument types
Useful for	Analyzing functions and procedures in your database, especially when optimizing stored functions
Example query	 SQLQuery13.sql
Columns of interest	proname: The name of the functions. prorettype: The return type of the function. proargtypes: The argument types of the function

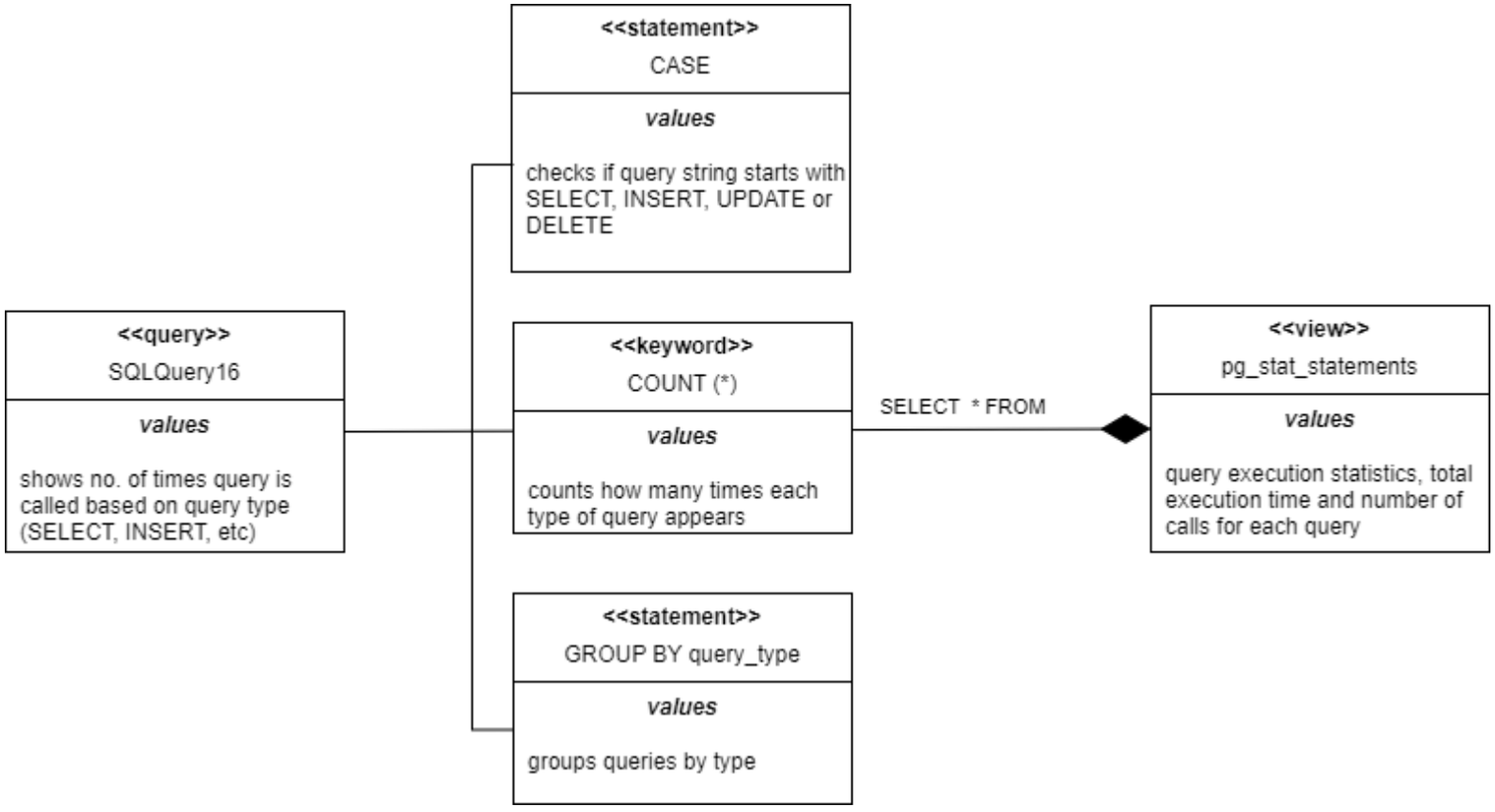
9. `pg_view` – Information About Views

Purpose	Provides metadata about view in the database
Useful for	Understanding the structure of views when analysing performance
Example query	 SQLQuery14.sql
Columns of interest	-

10. pg_stat_user_functions – Function Execution Statistics

Purpose	Provides performance statistics for user-defined functions including how often they are called and the total execution time
Useful for	Monitoring performance of stored procedures and functions, especially where logic may be encapsulated in functions
Example query	 SQLQuery15.sql
Columns of interest	funcname: The name of the function calls: The number of times function is called total_time: Total execution time of the function self_time: The time spent in the function itself (excluding the time spent in other function it calls)

11. regexp_matches(): No. of times Query is called based on query time (SELECT, INSERT, UPDATE, DELETE) _ Ex: in pg_stat_statements table

Purpose	Query to categorize and count query types (e.g., SELECT, INSERT, UPDATE, DELETE)
Useful for	To know no. of times query is called based on query type
Example query	SQLQuery16.sql SQLQuery17.sql
Flow	
Output	query_type | count SELECT | 150 INSERT | 40 UPDATE | 20 DELETE | 10 OTHER | 5

Important Factors to include to assess whether an index might improve performance

☐ High Number of sequential scans: If a table is frequently read using sequential scans (scanning all rows) and it often results in high I/O, This could indicate that index might improve performance

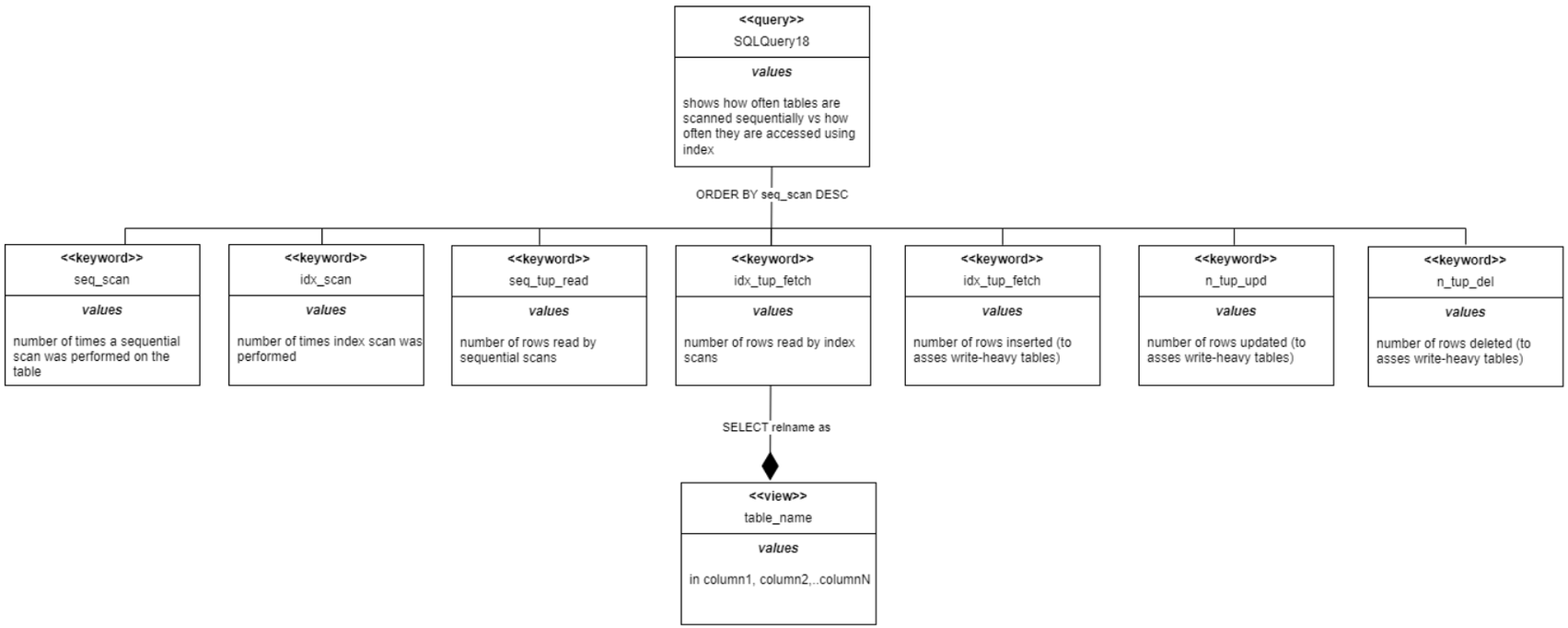
☐ Low Number of index scans: If there are few or no index scans happening for certain queries, and those queries frequently filter data or join adding index on those columns may help

☐ High selectivity: indexes are beneficial when the column has high selectivity (many distinct values, so each index entry points to a small set of rows)

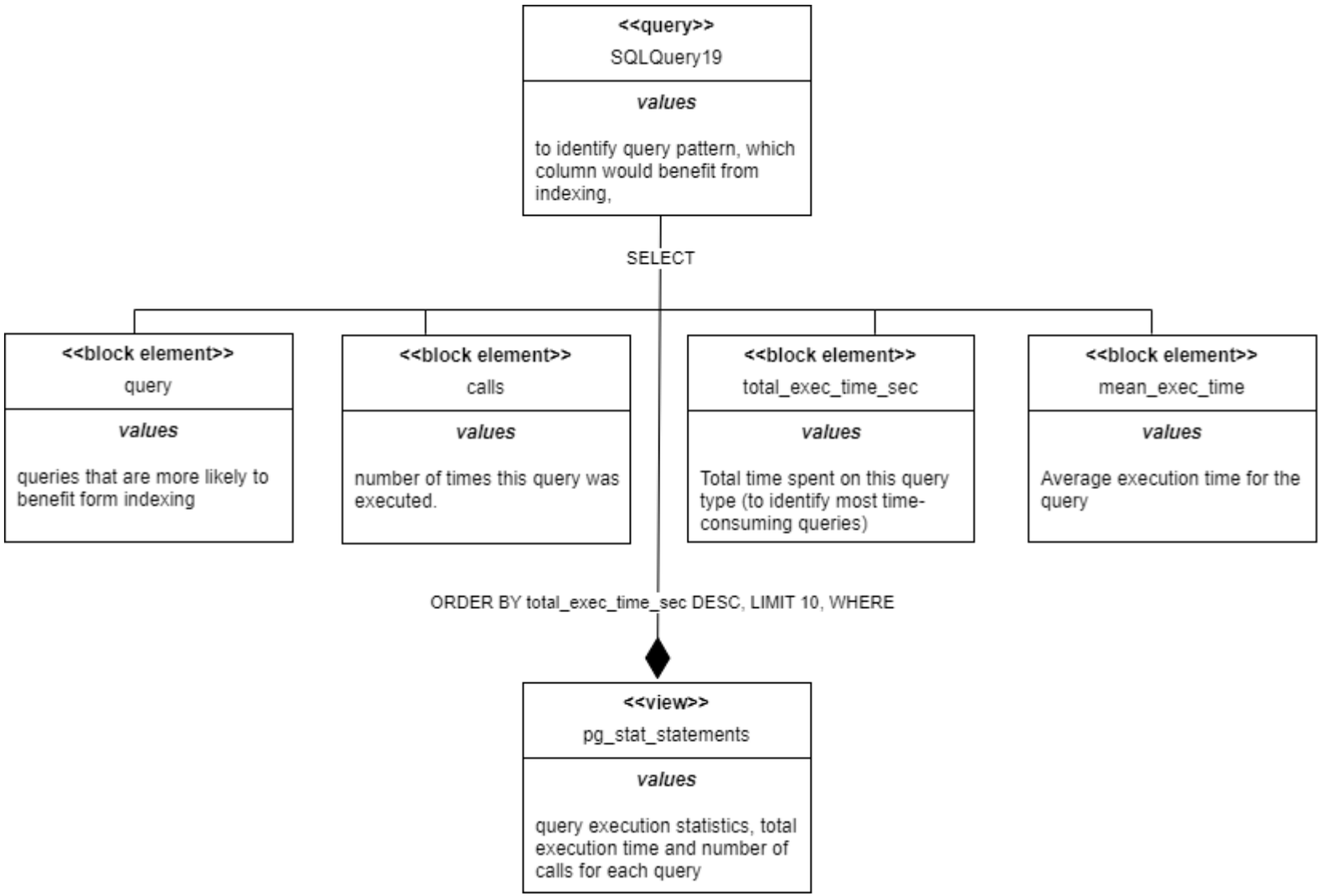
☐ Frequent filtering: Columns used frequently in WHERE, JOIN, ORDER_BY and GROUP_BY clauses are good candidates for indexing

☐ Read-heavy tables: Tables that are read often and are large in size will benefit the most from indexing.

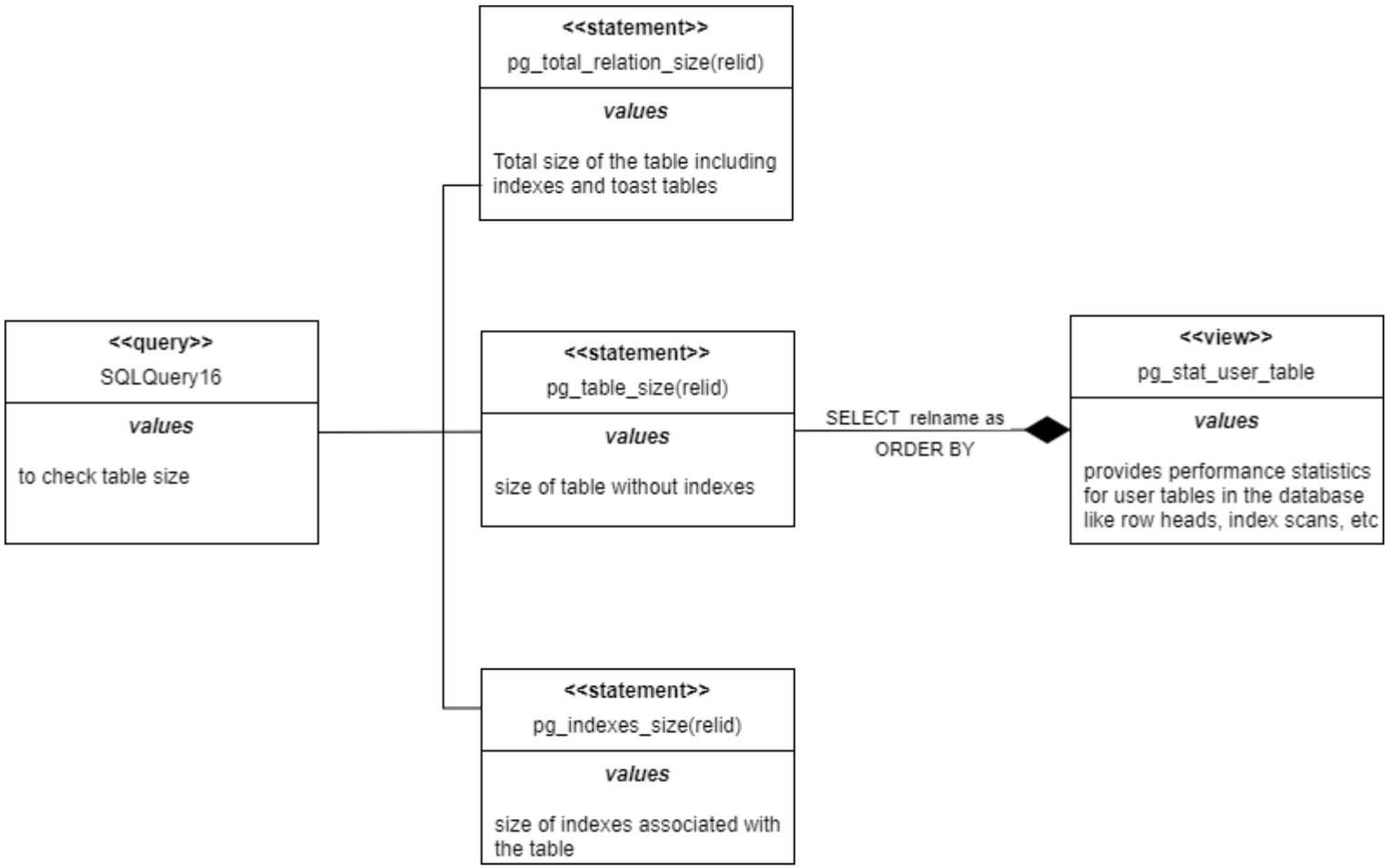
Step 1: Query to Analyze Sequential and Index Scans

Query	SQLQuery18.sql
Purpose	Query to find out tables are scanned sequentially vs how often they are accessed using an index
Flow	
Output Interpretation	High seq_scan and seq_tup_read values: If sequential scans are high for large tables consider adding index on frequently queried columns Low idx_scan values: If index scans are low but queries often involve filtering, joining, or ordering, index could improve performance

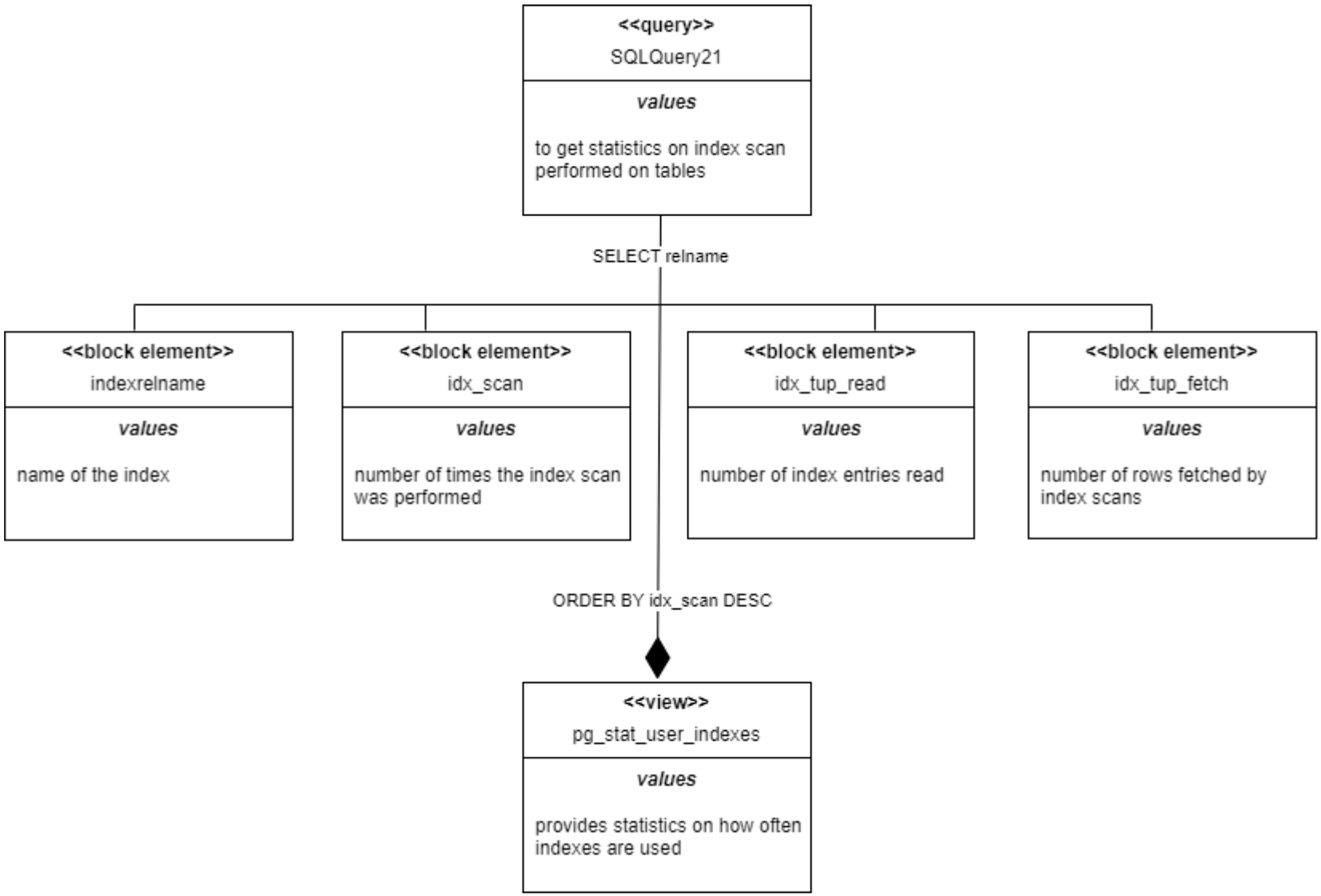
Step 2: Identify Columns to Index

Query	SQLQuery19.sql
Purpose	Query to find out which column would benefit from indexing, Queries that use `WHERE`, `JOIN`, `ORDER BY`, clauses may benefit from indexing on relevant columns.
Flow	

Step 3: Check table size to assess indexing feasibility

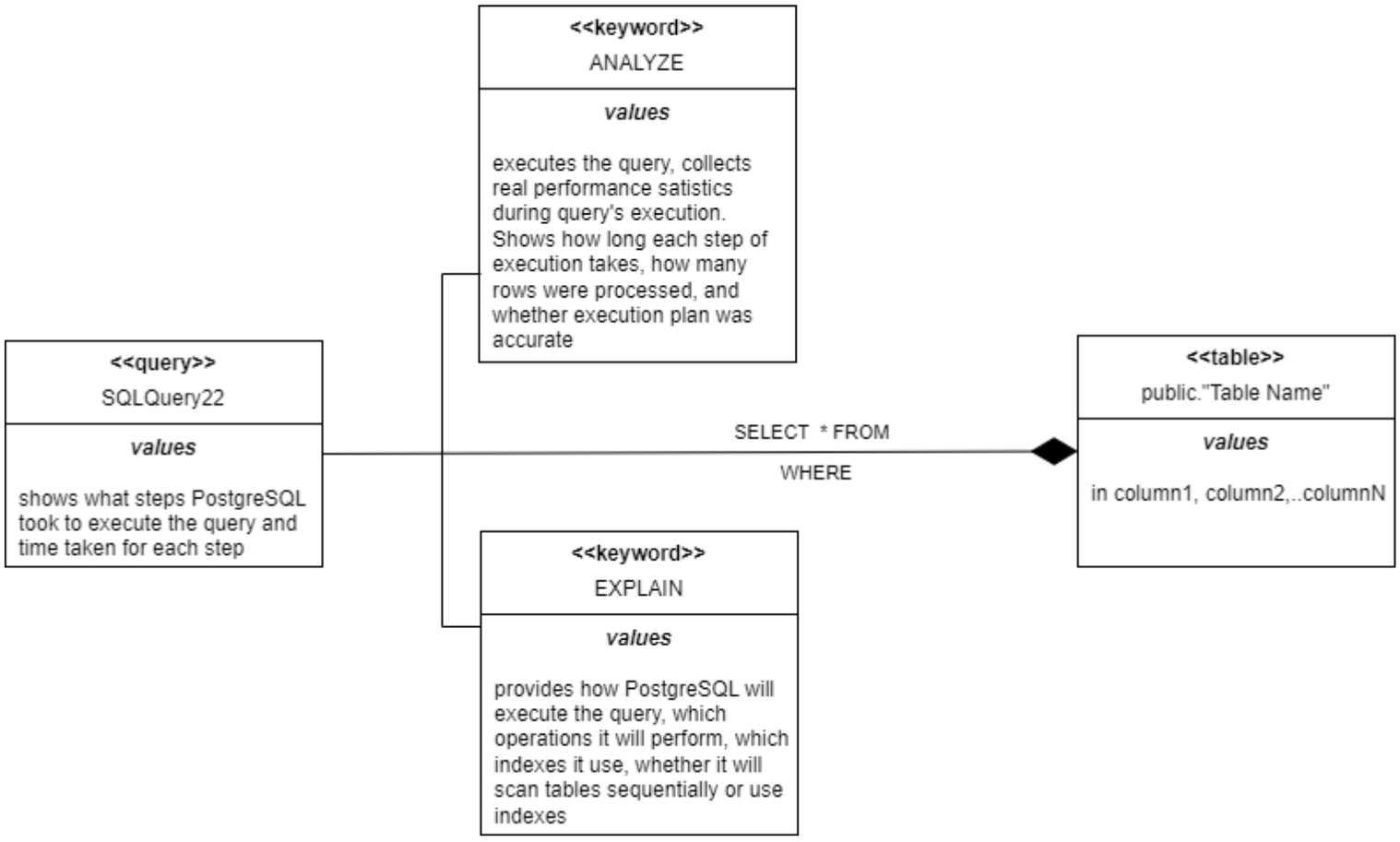
Query	SQLQuery20.sql
Purpose	To check table sizes. The size of table should be considered before adding an index. If a table is small adding an index might not provide significant performance improvements and might just add overhead.
Flow	

Step 4: Use `pg_stat_user_indexes` to see how often existing indexes are being used:

Query	SQLQuery21.sql
Purpose	To check number of times indexes scan is performed
Flow	

☐ If an index is not being used (`idx_scan` is low or 0) then you may need to: rewrite the queries to take advantage of existing indexes, consider if indexes is on appropriate column

Step 5: Use EXPLAIN to check Query execution plans

Query	SQLQuery22.sql
Purpose	To check how queries are executed and whether an index would help
Flow	

☐ If the output shows a sequential scan (i.e Seq Scan), and the table is large an index on column_name could improve performance
☐ If the output already shows an index scan (i.e Index Scan), the index is being utilized, and no additional indexing is required