


List of API Called at Home Screen and Canine Screen


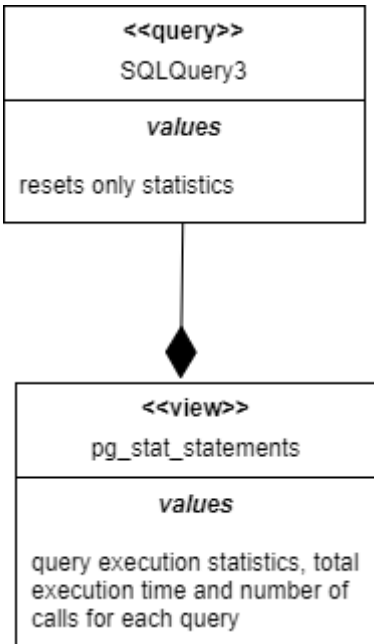



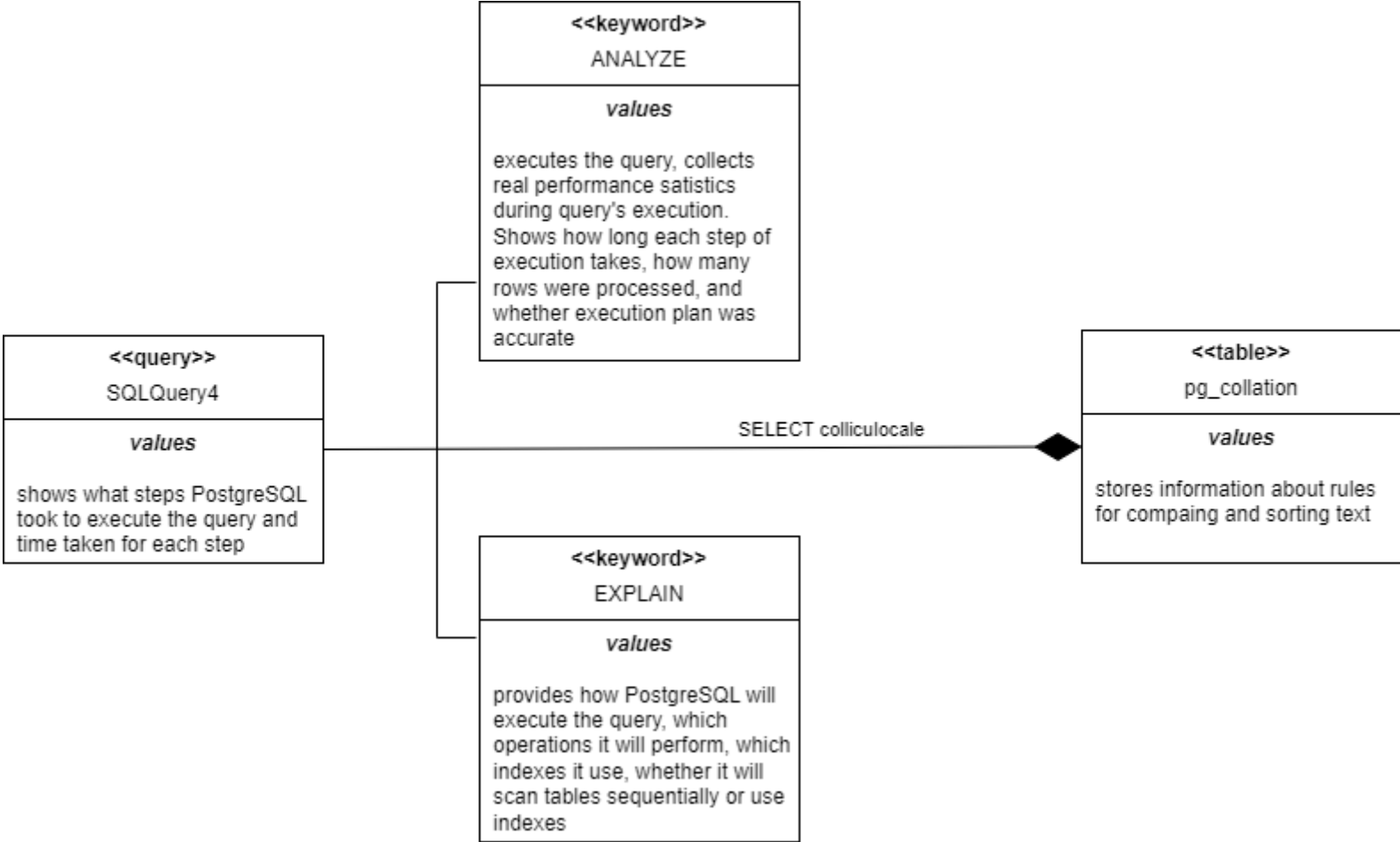
API_Home_Canine-s
creen.txt

Analysis of Queries Shared


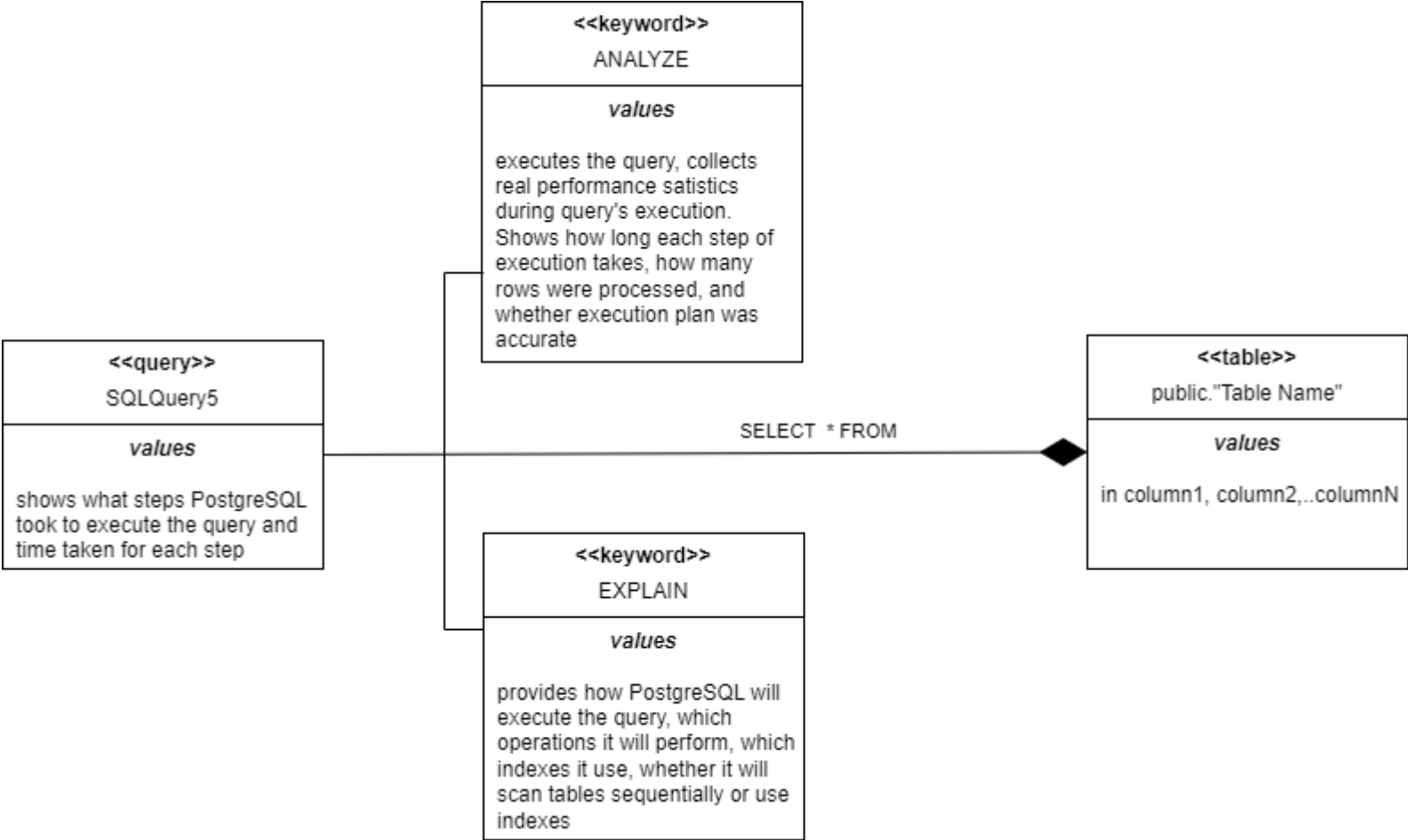
Query	 SQLQuery1.sql				
Purpose	This query retrieves performance statistics for 1000 most time-consuming queries stored in the pg_stat_statements view in postgresSQL.				
Flow	<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <<query>> SQLQuery1 <i>values</i> retrieves performance statistics </div> LIMIT 1000 queries ORDER BY total_exec_time DESC <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<statement>> query <i>values</i> actual SQL query text being analyzed. </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<block element>> calls <i>values</i> number of times this query was executed. </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<block element>> total_exec_time <i>values</i> /time : ms The total that PostgreSQL spent executing this query (reading, processing, writing, etc) </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<block element>> mean_exec_time <i>values</i> total execution time for this query divided by number of calls </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<block element>> percentage <i>values</i> The percentage of total execution time that this specific query contributed to all queries in the system $\frac{100 * total_exec_time}{sum(total_exec_time) OVER ()}$ </div> </div> <div style="text-align: center; margin-top: 10px;"> SELECT FROM <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <<view>> pg_stat_statements <i>values</i> query execution statistics, total execution time and number of calls for each query </div> </div> </div>				
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

Query	<div> SQLQuery2.sql</div>															
Purpose	This query helps in identifying top 1000 queries that cause most disk I/O operations															
Flow	<div><div><div><div><div><<query>></div><div>SQLQuery2</div><div>values</div><div>orders query based on the top block read and block write times</div></div><div>LIMIT 1000 queries with highest I/O times ORDER BY blk_read_time + blk_wite_time DESC</div><div><div><div><div><<block element>></div><div>blk_read_time</div><div>values</div><div>/time : ms The total time spent reading blocks from the disk. A high block read time may indicate that query is not using indexes effeciently or is accessing large amount of data</div></div><div><div><div><<block element>></div><div>total_exec_time</div><div>values</div><div>/time : ms The total that PostgreSQL spent executing this query (reading, processiing, writing, etc)</div><div>SELECT substring(query, 1, 30)</div><div><div><<view>></div><div>pg_stat_statements</div><div>values</div><div>query execution statistics, total execution time and number of calls for each query</div></div></div><div><div><div><<block element>></div><div>blk_wтите_time</div><div>values</div><div>/time : ms The total time spent writing blocks to disc. A high value may indicate that query is making a lot of changes to the database (INSERT, UPDATE, DELETE) or is not optimized for batch writes</div></div></div></div></div></div></div></div></div>															
Output Example:	<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>				query	total_time	blk_read_time	blk_wite_time	SELECT * FROM “AboutUs” where len_ourheader = 12	12000.45	8000.12	1000.55	INSERT INTO PetScheduler (len_ID, IsFav)	9500.25	5000.20	4000.80
query	total_time	blk_read_time	blk_wite_time													
SELECT * FROM “AboutUs” where len_ourheader = 12	12000.45	8000.12	1000.55													
INSERT INTO PetScheduler (len_ID, IsFav)	9500.25	5000.20	4000.80													

Query	 SQLQuery3.sql
Purpose	This query is used to reset the statistics collected by <code>pg_stat_statements</code> extension in PostgreSQL, giving “clean slate” for monitoring queries.
Flow	 <pre> classDiagram class SQLQuery3 { <<query>> values resets only statistics } class pg_stat_statements { <<view>> values query execution statistics, total execution time and number of calls for each query } SQLQuery3 --> pg_stat_statements </pre>
Output	Just a success message (or an error if something ges wrong)
When to use?	<ul style="list-style-type: none"> - 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 performamance tests and want to clear old stats

Query	 SQLQuery4.sql
Purpose	This query retrieves the <code>colliculate</code> from the system catalog table <code>pg_collation</code> . and with <code>EXPLAIN ANALYZE</code> , 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	 <pre>graph LR subgraph SQLQuery4 [SQLQuery4] direction TB QK[<<keyword>> SQLQuery4] QV[values] QV --> QV2[shows what steps PostgreSQL took to execute the query and time taken for each step] end subgraph ANALYZE [ANALYZE] direction TB AK[<<keyword>> ANALYZE] AV[values] AV --> AV2[executes the query, collects real performance statistics during query's execution. Shows how long each step of execution takes, how many rows were processed, and whether execution plan was accurate] end subgraph EXPLAIN [EXPLAIN] direction TB EK[<<keyword>> EXPLAIN] EV[values] EV --> EV2[provides how PostgreSQL will execute the query, which operations it will perform, which indexes it use, whether it will scan tables sequentially or use indexes] end subgraph pg_collation [pg_collation] direction TB PK[<<table>> pg_collation] PV[values] PV --> PV2[stores information about rules for compaing and sorting text] end SQLQuery4 -- "SELECT colliculocale" --> pg_collation ANALYZE --> SQLQuery4 EXPLAIN --> SQLQuery4</pre>
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	<p>Seq Scan on pg_collation: This means that PostgreSQL decided to perform a sequenteial scan over the <code>pg_collation</code> table. Since <code>pg_collation</code> is a system catalog table</p> <p>Cost=0.00...1.03 : This is an estimate of the cost for executing the query, where <code>0.00</code> is the startup cost (the cost of getting the first row), and <code>1.03</code> os total cost of executing the query</p> <p>row=3: This shows that PostgreSQL estiamted the query would return 3 rows.</p> <p>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</p> <p>loops=1: This indicates how many times PostgreSQL executed this part of the query.</p>


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

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	 <pre>graph LR A["<<query>> SQLQuery5 values shows what steps PostgreSQL took to execute the query and time taken for each step"] --> B["<<keyword>> ANALYZE values executes the query, collects real performance statistics during query's execution. Shows how long each step of execution takes, how many rows were processed, and whether execution plan was accurate"] B --> C["<<keyword>> EXPLAIN values provides how PostgreSQL will execute the query, which operations it will perform, which indexes it use, whether it will scan tables sequentially or use indexes"] C --> D["<<table>> public."Table Name" values in column1, column2,...columnN"] A -- "SELECT * FROM" --> D</pre>
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 retrieving all rows and column (SELECT *), PostgreSQL is doing a full table scan public."Table Name" refers to the table being scanned


	<p>(cost=0.00..35.50 rows=250 width=64) :</p> <p>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 retrieving all rows.</p> <p>rows=2550: this is the estimate of how many rows the query will return (2,550 in this case).</p> <p>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.</p> <p>(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</p> <p>rows=2550: this is the actual number of rows retrieved</p> <p>loops=1: This indicates how many times this step was executed (in this case, it was executed once).</p> <p>Output: column1, column2, column3,... This line shows which columns are being returned by the query.</p> <p>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)</p> <p>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</p>
--	---

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	<ul style="list-style-type: none"> - 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 read, rows inserted, etc
Useful for	Analyzing performance of tables used by queries and identifying bottlenecks
Example query	 SQLQuery7.sql
Columns of interest	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none">- <code>idx_scan</code>: Number of scans performed on the index- <code>idx_tup_read</code>: Number of index entries read- <code>idx_tup_fetch</code>: 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	<code>heap_blks_read</code> : Number of disk blocks read for the table. <code>heap_blks_hit</code> : Number of buffer hits for the table (when data was found in memory) <code>idx_blks_read</code> , <code>idx_blks_hit</code> : 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	<code>idx_blks_read</code> : Number of disk blocks read for the index <code>idx_blks_hit</code> : 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



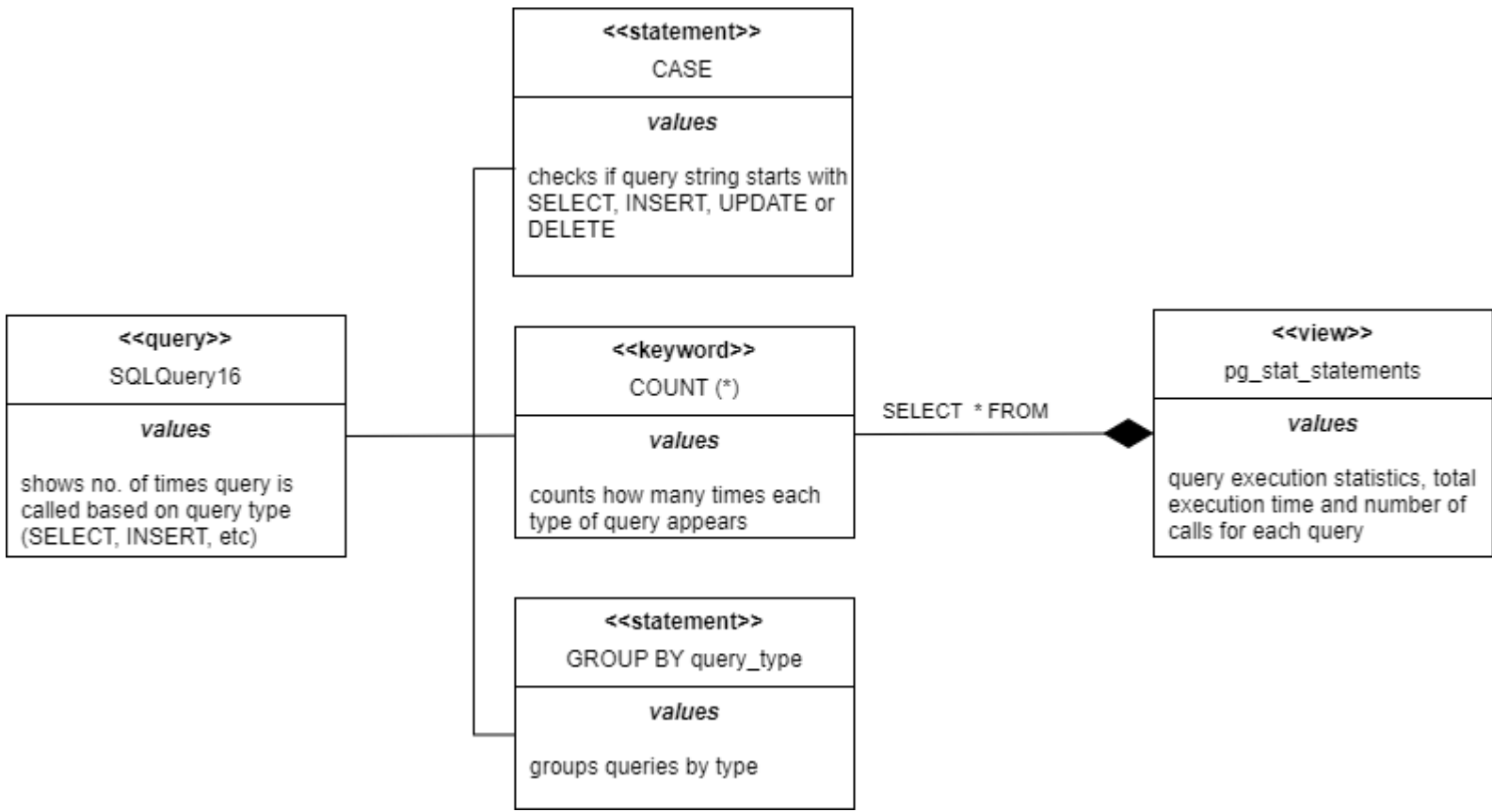
9. `pg_view` – Information About Views

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., <code>SELECT</code> , <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code>)
Useful for	To know no. of times query is called based on query type
Example query	  SQLQuery16.sql SQLQuery17.sql
Flow	 <pre> graph LR A["<<query>> SQLQuery16 values shows no. of times query is called based on query type (SELECT, INSERT, etc)"] B["<<statement>> CASE values checks if query string starts with SELECT, INSERT, UPDATE or DELETE"] C["<<keyword>> COUNT (*) values counts how many times each type of query appears"] D["<<statement>> GROUP BY query_type values groups queries by type"] E["<<view>> pg_stat_statements values query execution statistics, total execution time and number of calls for each query"] A --- B A --- D B --- C D --- C C -- "SELECT * FROM" --> E </pre>
Output	<pre> query_type count SELECT 150 INSERT 40 UPDATE 20 DELETE 10 OTHER 5 </pre>


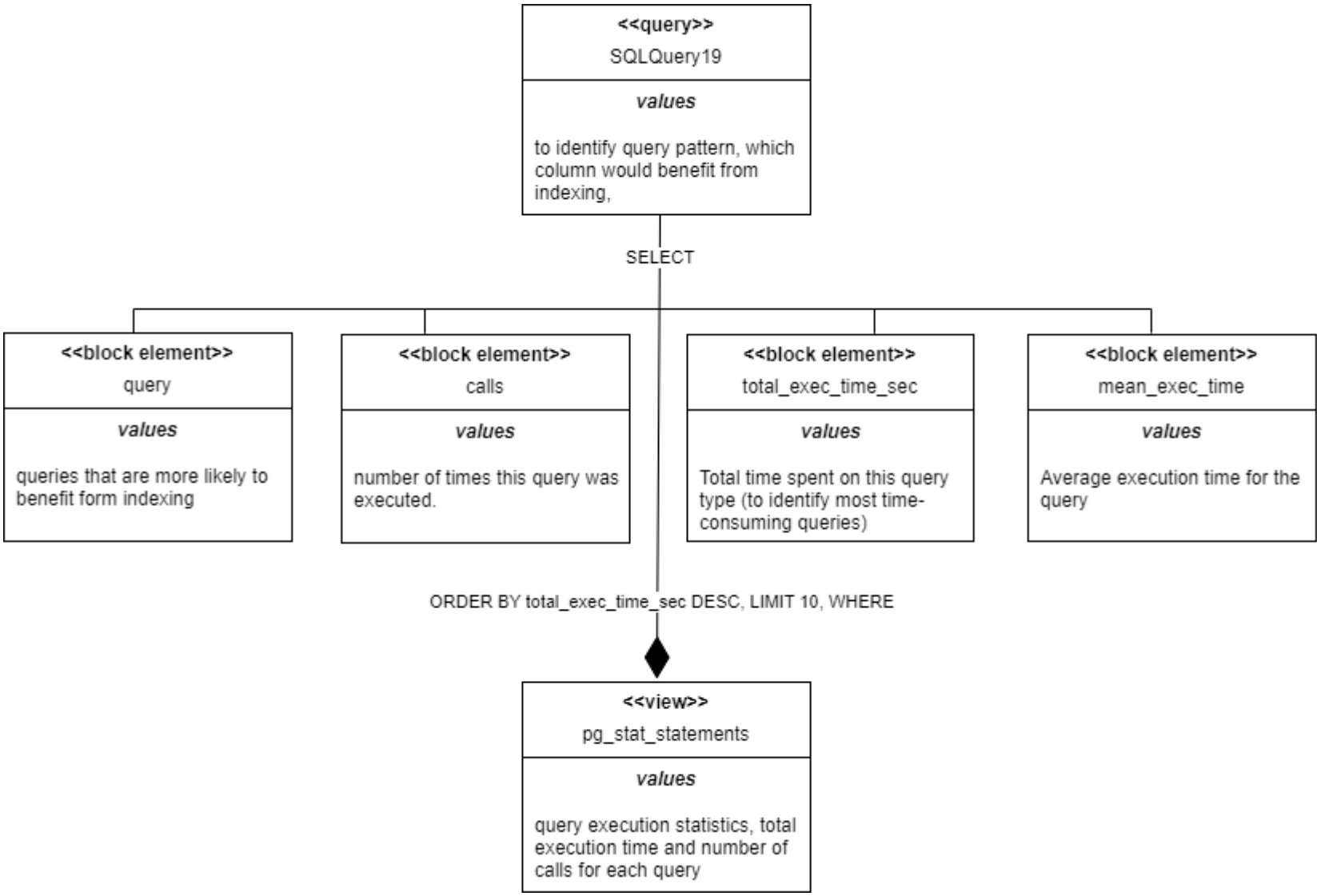
Important Factors to include to assess whether an index might improve performance

- ☐ 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.


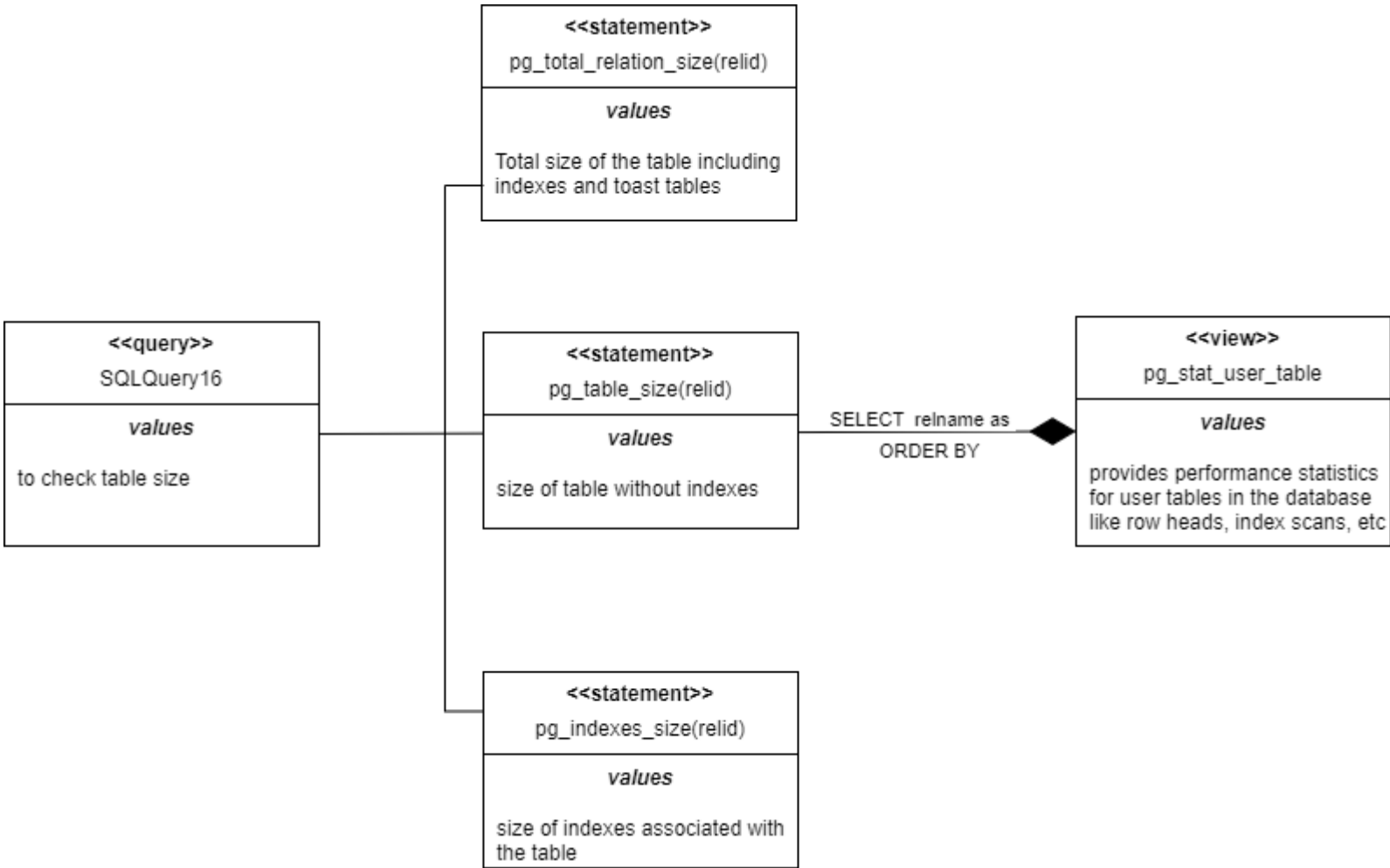
Step 1: Query to Analyze Sequential and Index Scans

Query	 SQLQuery18.sql
Purpose	Query to find out tables are scanned sequentially vs how often they are accessed using an index
Flow	<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <<query>> SQLQuery18 values shows how often tables are scanned sequentially vs how often they are accessed using index </div> <div>ORDER BY seq_scan DESC</div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> seq_scan values number of times a sequential scan was performed on the table </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> idx_scan values number of times index scan was performed </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> seq_tup_read values number of rows read by sequential scans </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> idx_tup_fetch values number of rows read by index scans </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> idx_tup_fetch values number of rows inserted (to asses write-heavy tables) </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> n_tup_upd values number of rows updated (to asses write-heavy tables) </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<keyword>> n_tup_del values number of rows deleted (to asses write-heavy tables) </div> </div> <div style="margin-top: 10px;"> <div>SELECT rename as</div> <div style="display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: black; margin-right: 5px;"></div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <<view>> table_name values in column1, column2,...columnN </div> </div> </div> </div>
Output Interpretation	<p>High seq_scan and seq_tup_read values: If sequential scans are high for large tables consider adding index on frequently queried columns</p> <p>Low idx_scan values: If index scans are low but queries often involve filtering, joining, or ordering, index could improve performance</p>

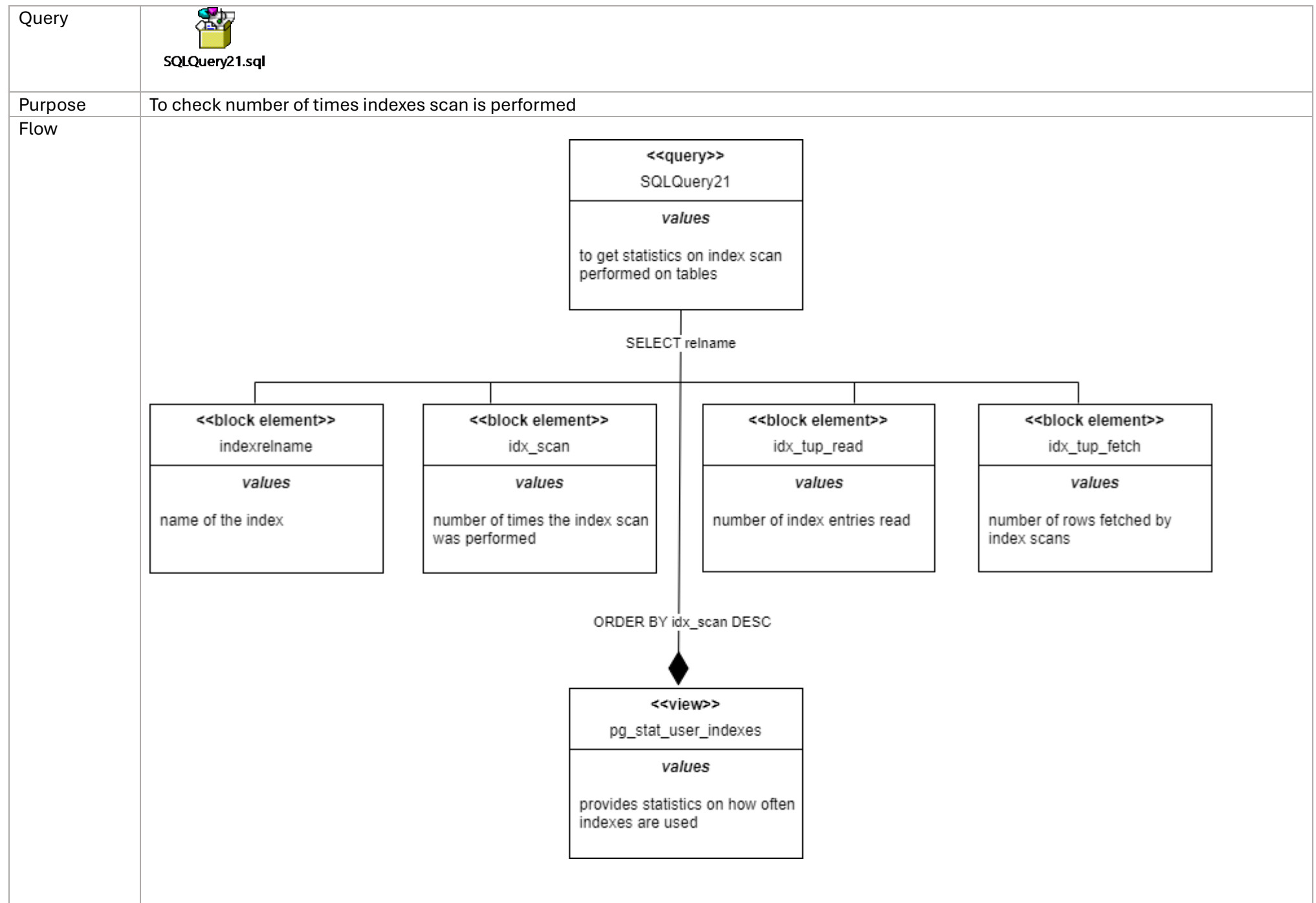
Step 2: Identify Columns to Index

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	 <pre> graph TD Q["<<query>> SQLQuery19 values to identify query pattern, which column would benefit from indexing,"] B1["<<block element>> query values queries that are more likely to benefit form indexing"] B2["<<block element>> calls values number of times this query was executed."] B3["<<block element>> total_exec_time_sec values Total time spent on this query type (to identify most time- consuming queries)"] B4["<<block element>> mean_exec_time values Average execution time for the query"] V["<<view>> pg_stat_statements values query execution statistics, total execution time and number of calls for each query"] Q -- SELECT --> B1 Q -- SELECT --> B2 Q -- SELECT --> B3 Q -- SELECT --> B4 B1 -- "ORDER BY total_exec_time_sec DESC, LIMIT 10, WHERE" --> V B2 -- "ORDER BY total_exec_time_sec DESC, LIMIT 10, WHERE" --> V B3 -- "ORDER BY total_exec_time_sec DESC, LIMIT 10, WHERE" --> V B4 -- "ORDER BY total_exec_time_sec DESC, LIMIT 10, WHERE" --> V </pre>

Step 3: Check table size to assess indexing feasibility


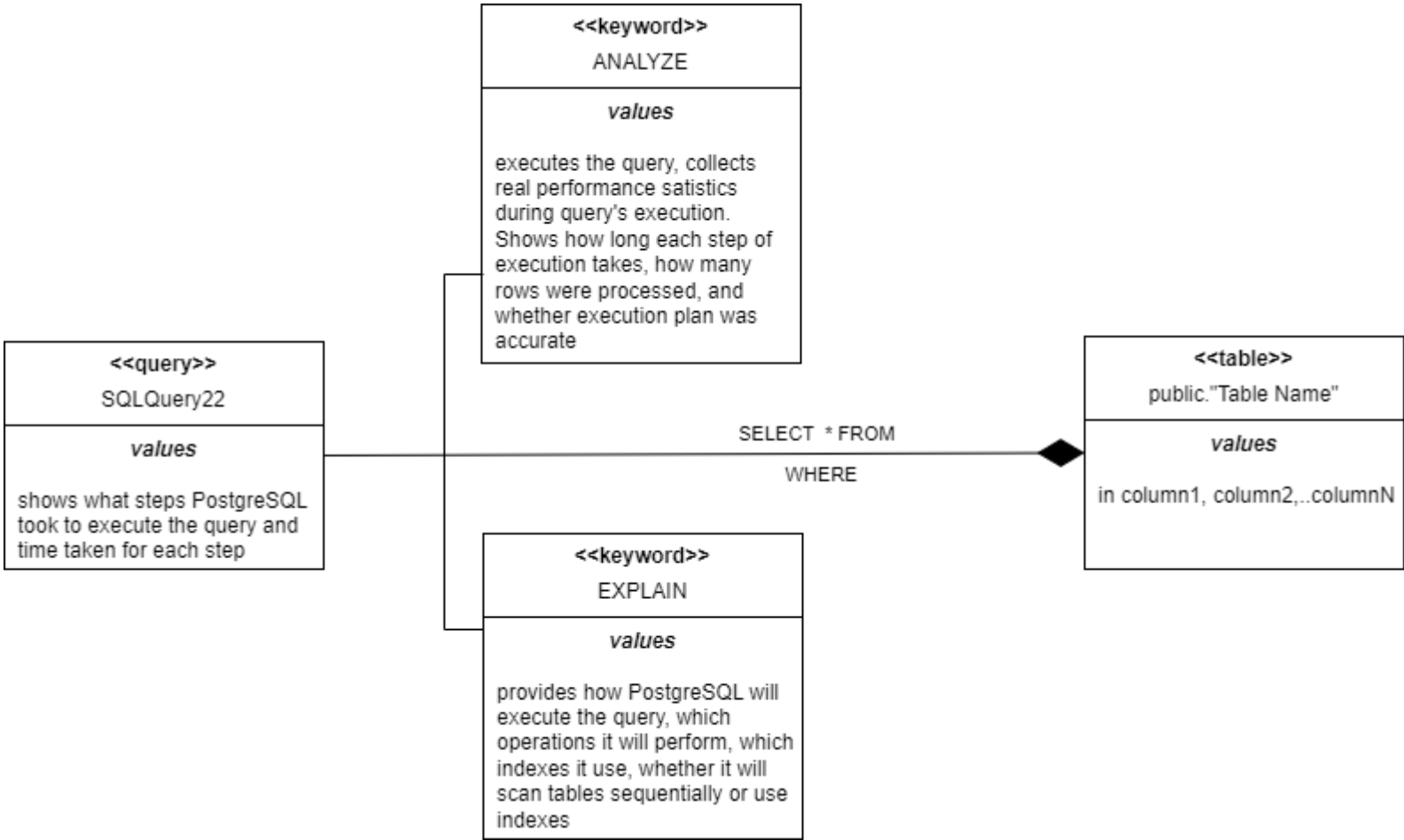
Query	 SQLQuery20.sql
Purpose	To check table sizes. The size of table should be considered before adding an index. If a table is small adding an index might not provide significant performance improvements and might just add overhead.
Flow	 <pre> graph LR Q1["<<query>> SQLQuery16 values to check table size"] S1["<<statement>> pg_total_relation_size(releid) values Total size of the table including indexes and toast tables"] S2["<<statement>> pg_table_size(releid) values size of table without indexes"] S3["<<statement>> pg_indexes_size(releid) values size of indexes associated with the table"] V1["<<view>> pg_stat_user_table values provides performance statistics for user tables in the database like row heads, index scans, etc"] Q1 --- S1 Q1 --- S2 Q1 --- S3 S2 -- "SELECT rename as ORDER BY" --> V1 </pre> <p>The flowchart illustrates the process of checking table size to assess indexing feasibility. It begins with a query, SQLQuery16, which has the purpose of checking table size. This query branches into three statements:</p> <ul style="list-style-type: none"> pg_total_relation_size(releid): Returns the total size of the table including indexes and toast tables. pg_table_size(releid): Returns the size of the table without indexes. This statement is further processed by a SELECT rename as ORDER BY operation to produce a view. pg_indexes_size(releid): Returns the size of indexes associated with the table. <p>The view, pg_stat_user_table, provides performance statistics for user tables in the database, such as row heads, index scans, etc.</p>

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



□ 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	 <pre> graph LR Q["<<query>> SQLQuery22 values shows what steps PostgreSQL took to execute the query and time taken for each step"] E["<<keyword>> EXPLAIN values provides how PostgreSQL will execute the query, which operations it will perform, which indexes it use, whether it will scan tables sequentially or use indexes"] A["<<keyword>> ANALYZE values executes the query, collects real performance statistics during query's execution. Shows how long each step of execution takes, how many rows were processed, and whether execution plan was accurate"] T["<<table>> public. 'Table Name' values in column1, column2,...columnN"] Q -- "SELECT * FROM" --> A E -- "WHERE" --> T A --> T </pre> <p>The diagram illustrates the flow of the EXPLAIN command. It starts with a query box labeled "SQLQuery22" which points to an "EXPLAIN" box. The "EXPLAIN" box points to an "ANALYZE" box, which then points to a table box labeled "public. 'Table Name'". The "EXPLAIN" box also points directly to the table box. The "ANALYZE" box contains a description of its function: "executes the query, collects real performance statistics during query's execution. Shows how long each step of execution takes, how many rows were processed, and whether execution plan was accurate". The "EXPLAIN" box contains a description: "provides how PostgreSQL will execute the query, which operations it will perform, which indexes it use, whether it will scan tables sequentially or use indexes". The table box contains a description: "in column1, column2,...columnN".</p>

- ☐ 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