`query_table` Function Documentation

MMM Arachchi

2025-09-22

Purpose

The query_table function allows users to dynamically build and execute SQL queries against a connected relational database directly from R.

It supports:

- Filtering data with multiple conditions (`WHERE`)
- Selecting specific columns (`SELECT`)
- Grouping results (`GROUP BY`)
- Sorting results (`ORDER BY`)
- Limiting the number of rows returned (`LIMIT`)

This function is useful for both exploratory data analysis and reporting workflows and programmatic data extraction from a database.

Dependencies

- A working database connection function (`create_db_connection()`)
- A helper function `run_query(query)` to execute SQL queries safely
- Ensure `connect_db.Rmd` (or the corresponding R script) has been sourced:

```r
source("connect_db.R")
```

Function Signature

```r
query_table(db_table_name,
            query_condition_1 = NULL, query_condition_2 = NULL, ...,
            select_cols = "*", group_by = NULL, order_by = NULL, limit = NULL)
```

Parameter	Type	Description
`db_table_name`	character	Name of the database table to query.
`query_condition_1, query_condition_2, ...`	character	Optional filtering conditions for the `WHERE` clause. Multiple conditions are combined with `AND`.
`select_cols`	character	Columns to select from the table. Default is `"*"`.
`group_by`	character	Optional `GROUP BY` clause to aggregate results.

Parameter	Type	Description
order_by	character	Optional ORDER BY clause to sort results.
limit	integer	Optional LIMIT to restrict the number of rows returned.

Return Value

A data.frame containing the result of the query.

Usage Examples (Needs to replace these examples later with real db examples)

1. Simple filter

```r
query_table("students", "student_id >= 5")
```

2. Ordering results

```r
query_table("students", "enrollment_year > 2022", order_by = "student_id DESC")
```

3. Limiting results

```r
query_table("students", order_by = "student_id", limit = 3)
```

4. Grouping with aggregation

```r
query_table("students",
            select_cols = "enrollment_year, COUNT(*) AS student_count",
            group_by = "enrollment_year")
```

5. Aggregation with average and ordering

```r
query_table("student_course_summary",
            select_cols = "enrollment_year, AVG(gpa) AS avg_gpa",
            group_by = "enrollment_year",
            order_by = "avg_gpa DESC")
```

6. Selecting specific columns with limit

```r
query_table("student_course_summary", select_cols = "student_id, first_name", limit = 2)
```

Notes & Recommendations

- All query_condition parameters should be valid SQL fragments. For example:
  - "student_id = 10"
  - "email LIKE '%@domain.com%'"
  - "enrollment_year >= 2020"

- When using `GROUP BY`, ensure selected columns are either in the `GROUP BY` clause or used in an aggregate function (`COUNT()`, `AVG()`, etc.), otherwise PostgreSQL will raise an error.
- This function **prints the generated SQL query** before executing it, which helps with debugging.
- The function depends on `run_query()`, so ensure `connect_db.R` is sourced before using this function:

  ```r
  source("connect_db.R")
  ```

Function Code

```r
query_table <- function(db_table_name,
                        query_condition_1 = NULL, query_condition_2 = NULL, ...,
                        select_cols = "*", group_by = NULL, order_by = NULL, limit = NULL) {
  conditions <- c(query_condition_1, query_condition_2, ...)
  conditions <- conditions[!sapply(conditions, is.null)]

  where_clause <- ""
  if (length(conditions) > 0) {
    where_clause <- paste("WHERE", paste(conditions, collapse = " AND "))
  }

  group_clause <- ""
  if (!is.null(group_by)) {
    group_clause <- paste("GROUP BY", group_by)
  }

  order_clause <- ""
  if (!is.null(order_by)) {
    order_clause <- paste("ORDER BY", order_by)
  }

  limit_clause <- ""
  if (!is.null(limit)) {
    limit_clause <- paste("LIMIT", limit)
  }

  query <- paste("SELECT", select_cols, "FROM", db_table_name,
                 where_clause, group_clause, order_clause, limit_clause)

  message("Running query: ", query)
  result <- run_query(query)
  return(result)
}
```

Recommended Workflow

1. Source the `connect_db.R` to load `create_db_connection()` and `run_query()`.
2. Use `query_table()` to fetch data.
3. Inspect and manipulate the resulting data frame in R.
4. Always disconnect the database if you opened a separate connection.

Session Info

```r
sessionInfo()
```

```
## R version 4.4.3 (2025-02-28 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 22621)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_Australia.utf8  LC_CTYPE=English_Australia.utf8
## [3] LC_MONETARY=English_Australia.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_Australia.utf8
##
## time zone: Australia/Perth
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] RPostgres_1.4.8 DBI_1.2.3
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.37     fastmap_1.2.0     xfun_0.51         bit_4.6.0
##  [5] blob_1.2.4        knitr_1.50        pkgconfig_2.0.3   htmltools_0.5.8.1
##  [9] rmarkdown_2.29    bit64_4.6.0-1     lifecycle_1.0.4   cli_3.6.4
## [13] vctrs_0.6.5       compiler_4.4.3    rstudioapi_0.17.1 tools_4.4.3
## [17] hms_1.1.3         evaluate_1.0.3    yaml_2.3.10       rlang_1.1.5
```