Lab 10 – Structured Query Langauge (SQL)

Contents


```
# FOR MAC USERS:
# Sys.setenv(MARIADB_TLS_DISABLE_PEER_VERIFICATION = 1)
```

```
install.packages(c("DBI", "odbc", "RMySQL", "dbplyr"))
```

Lab

```
library(DBI)
library(odbc)
library(RMySQL)
library(dbplyr)
```

We start by establishing a connection to an active RDBMS server hosted here on Grinnell's campus (note: you must be on campus wi-fi to do this). We can name this connection whatever we want (in this case, we call it `con`), and all of our SQL operations will be performed in the context of this connection.

```
con <- dbConnect(
  MySQL(),
  dbname   = "STA230",
  host     = "aiken.cs.grinnell.edu",
  port     = 3306,
  user     = "student",
  password = "password!"
)
```

Once we have finished working with our connection, it's good practice to disconnect using

```
## Don't run this until you're finished
dbDisconnect(con)
```

```
dbListTables(con)
```

```
##  [1] "Batting"            "Iowa_homes_1"       "Iowa_homes_2"
##  [4] "MLBPlayers"         "MLBTeams"           "Police_shootings_1"
##  [7] "Police_shootings_2" "batting2"           "course"
## [10] "course_offering"    "courses"            "customers"
## [13] "enrollments"        "office_hour"        "orders"
## [16] "players2"           "professor_info"     "students"
## [19] "syllabus_info"      "teams2"
```

Problem 1: First, display the variables that are available in the `customers` table. Return the `id` and `name` columns in all instances where the row names are less than 4

```
dbGetQuery(con, "select id, name from customers where row_names<4")
```

```
##   id    name
## 1  4   Tukey
## 2  8 Wickham
## 3 15   Mason
```

Problem 2: Write a SQL query that performs the following:

- Selects only order, id, and date from orders table, but change column names to be all caps
- Left join on customers by id
- Filter to only include ID values of 4, 8, and 50
- (Hint: If something is ambiguous, it helps to specify which table it comes from)

Your output should look like this

```
dbGetQuery(con, "select o.id as ID, o.date as DATE, o.order AS ORDERS from orders AS o
           LEFT JOIN customers ON o.id = customers.id
           WHERE o.ID in (4, 8, 50)")
```

```
##   ID   DATE ORDERS
## 1  4 Jan-01      1
## 2  4 Apr-18     11
## 3  8 Feb-01      2
## 4  8 Feb-11     12
## 5 50 Apr-17      4
## 6 50 Jun-17     14
```

```
dbListTables(con)
```

```
##  [1] "Batting"            "Iowa_homes_1"       "Iowa_homes_2"
##  [4] "MLBPlayers"         "MLBTeams"           "Police_shootings_1"
##  [7] "Police_shootings_2" "batting2"           "course"
## [10] "course_offering"    "courses"            "customers"
## [13] "enrollments"        "office_hour"        "orders"
## [16] "players2"           "professor_info"     "students"
## [19] "syllabus_info"      "teams2"
```

Problem 3: Identify the all unique student IDs for students who are enrolled in at least one course

```
dbGetQuery(con, "SELECT DISTINCT students.student_id FROM students
           INNER JOIN enrollments as e ON e.student_id = students.student_id")
```

```
##   student_id
## 1       S001
## 2       S002
## 3       S003
## 4       S007
## 5       S008
```

Problem 4: Identify all of the courses (by course name) that have at least one student enrolled

```
dbGetQuery(con, "SELECT DISTINCT course_name from courses
           INNER JOIN enrollments as e ON e.course_id = courses.course_id")
```

```
##                 course_name
## 1       Intro to Statistics
## 2            Linear Algebra
## 3              Econometrics
```

```
## 4 Intro to Political Science
```

Problem 5: How do you think you join on multiple tables? Try to produce the following output showing each student and the name of the courses in which they're enrolled. Note also the change in variable names

```
dbGetQuery(con, "select s.name as Student_Name, c.course_name as Course_Name from students as s
           inner join enrollments as e ON s.student_id = e.student_id
           inner join courses as c on e.course_id=c.course_id")
```

```
##   Student_Name               Course_Name
## 1        Alice        Intro to Statistics
## 2        Alice             Linear Algebra
## 3          Bob        Intro to Statistics
## 4          Bob               Econometrics
## 5      Charlie               Econometrics
## 6        Fiona Intro to Political Science
## 7       George             Linear Algebra
```

Problem 6: Return the names and majors of all students not enrolled in any classes.

```
dbGetQuery(con, "SELECT s.name, s.major FROM students AS s
           LEFT JOIN enrollments as e ON e.student_id = s.student_id
           WHERE e.student_id IS NULL")
```

```
##    name      major
## 1 Diana Statistics
## 2  Evan    Physics
```

Problem 7: Suppose we wanted to retain the IDs with the largest two orders (like we did just above) without also returning the `total_orders` column (that is, we *just* want to return the ID). How might we do this?

```
dbGetQuery(con, "SELECT id FROM orders
           WHERE row_names > 2
           GROUP BY id
           ORDER BY COUNT(orders.order) DESC
           LIMIT 2")
```

```
##   id
## 1 42
## 2 50
```

Problem 8: Revisit your code from **Problem 3**, and expand upon this to return the number of students who are enrolled in at least one course.

Problem 9: The tables `batting2` and `players2` include MLB batting and player information from the 2010 season or later, respectively. Only considering the 2020 season and later, return the *given name* of the top 10 MLB players with the most homeruns. It should look something like this: (your query may take a few seconds to run. Also, you may ignore warning about importing as numeric)

```
dbGetQuery(con, "select sum(HR) as total_hr, nameGiven as name from batting2
           LEFT JOIN players2 as p ON p.playerID = batting2.playerID
           WHERE yearID > 2020
           GROUP BY batting2.playerID
           ORDER BY total_hr DESC LIMIT 10")
```

```
##    total_hr           name
## 1       196    Aaron James
## 2       178         Shohei
## 3       163    Kyle Joseph
## 4       157   Peter Morgan
```

```
## 5       156   Matthew Kent
## 6       136   Yordan Ruben
## 7       136       Vladimir
## 8       132      Juan Jose
## 9       128   Jose Enrique
## 10      127 Michael Austin
```

Problem 10: Why does the following query fail? What can you do to fix it?

```
# dbGetQuery(con, "SELECT COUNT(orders.order), id from orders
#            GROUP BY id
#            HAVING row_names BETWEEN 2 AND 6 OR id = 50")
```

```
dbGetQuery(con, "SELECT COUNT(orders.order), id from orders
           WHERE row_names BETWEEN 2 AND 6 OR id = 50
           GROUP BY id")
```

```
##   COUNT(orders.order) id
## 1                   1  4
## 2                   2  8
## 3                   1 42
## 4                   2 50
```

Extracises

Problem 11: In Homework 2, Question 3, you were asked to find the slugging percentage for each team from 1969 and return the top 10. Rewrite that query in SQL syntax, pulling instead from the `MLBTeams` table on the database. You should end up with the following query

```
dbGetQuery(con, "SELECT
           yearID, teamID,
           ((H - X2B - X3B - HR) + 2*X2B + 3*X3B + 4*HR) / AB AS SLG
           FROM MLBTeams
           WHERE yearID > 1969
           ORDER BY SLG desc
           LIMIT 10")
```

```
##    yearID teamID    SLG
## 1    2023    ATL 0.5008
## 2    2019    HOU 0.4955
## 3    2019    MIN 0.4941
## 4    2003    BOS 0.4909
## 5    2019    NYA 0.4899
## 6    1997    SEA 0.4845
## 7    1994    CLE 0.4838
## 8    1996    SEA 0.4836
## 9    2001    COL 0.4830
## 10   2020    LAN 0.4829
```

Problem 12: Run a query to find the ten players with at least 5 career at-bats (AB) (that is, not 5 AB per season) with the highest home run percentage (home runs divided by at bats) from the `Batting` table

```
dbGetQuery(con, "SELECT playerID,
           sum(H) / sum(AB) AS avg_hr FROM Batting
           GROUP BY playerID
           HAVING avg_hr IS NOT NULL AND SUM(AB) > 5
           ORDER BY avg_hr DESC
```

```
          LIMIT 10")
```

```
##     playerID avg_hr
## 1  kerwida01 0.6667
## 2  wiedeto01 0.6667
## 3  brittza01 0.6250
## 4  riverca02 0.5714
## 5  leflewa01 0.5556
## 6  silvelu01 0.5455
## 7  baconed01 0.5000
## 8  bakerda02 0.5000
## 9  bierbni01 0.5000
## 10 blauvhe01 0.5000
```

Problem 13: Run a query to determine how many players with at least 5 at-bats (AB) have a career home run percentage (home runs divided by at bats) of at least 0.3. Confirm that it is 647

```
dbGetQuery(con, "SELECT COUNT(*) AS n_players
          FROM (SELECT sum(H) / sum(AB) AS avg_hr FROM Batting
          GROUP BY playerID
          HAVING avg_hr > 0.3 AND sum(AB) > 5) as new_table")
```

```
##   n_players
## 1       647
```

Problem 14: Run a query to find all players whose total career home runs exceeds the average total career home runs across all players since 1980

```
dbGetQuery(con, "SELECT playerID, SUM(H) as career_hr FROM Batting
          WHERE yearID > 1980
          GROUP BY playerID
          HAVING career_hr > (
            SELECT AVG(total_h) FROM (
              SELECT SUM(H) AS total_h from Batting
              GROUP BY playerID
            ) AS new_table
          )")
```

Problem 15: Determine the number of players who have at least 5 seasons in which they hit 20 or more home runs since 1980

```
dbGetQuery(con, "SELECT COUNT(*) AS n_players
          FROM (
            SELECT playerID
            FROM Batting
            WHERE yearID >= 1980 AND HR >= 20
            GROUP BY playerID
            HAVING COUNT(yearID) >= 5
          ) AS new_table")
```

```
##   n_players
## 1       253
```