15. More SQL queries : SQL joins

Principles of Data Science with R

Dr. Uma Ravat

PSTAT 10

Announcement

1. Questions? UAW strike sections, TA office hours, due dates
2. Quiz this week as usual
3. No new HW this week.
4. Next week: Tues/Wed
5. Week after Thanksgiving: Redo Quiz on Tuesday

See announcement sent yesterday.

We saw

- Integrity constraints (Primary and Foreign Keys)
- All parts of a SQL query

```
SELECT columns or computations
  FROM table
  WHERE condition
  GROUP BY columns
  HAVING condition
  ORDER BY column [ASC | DESC]
  LIMIT offset, count;
```

- Database tools for R
  - the R packages `RSQLite`, `DBI`
  - the database `Chinook_Sqlite.sqlite`

Connecting to a DB

```
library(DBI)
library(RSQLite)
drv = dbDriver("SQLite") # the driver for the db you want to connect to
chinook_db = dbConnect(drv, # the driver to use
              dbname="./data/Chinook_Sqlite.sqlite") #path to the db file
```

SELECT

```
SELECT columns or computations
  FROM table
  WHERE condition # specify a subset of the rows to use (*pre-aggregation/pre-calculation*)
  GROUP BY columns # defines aggregation groups
  HAVING condition # specify a subset of the rows to display (*post-aggregation/post-calculation*)
  ORDER BY column [ASC | DESC]
  LIMIT offset ;
```

"simple" queries on a single table

SELECT, expanded

In the second line of SELECT, we can specify more than one data table using JOIN using primary keys to join on.

```
SELECT columns or computations
  FROM tableA JOIN tableB "USING(key)"
  WHERE condition
  GROUP BY columns
  HAVING condition
  ORDER BY column [ASC | DESC]
  LIMIT offset;
```

queries over several tables.

Querying multiple tables

In a relational DB, the data are split among multiple tables and related by their keys.

```
dbGetQuery(chinook_db, "select TrackId, Name, AlbumId from Track
                        where AlbumId = 250 limit 5")
```

```
##   TrackId              Name AlbumId
## 1    3178       The Dundies     250
## 2    3179 Sexual Harassment     250
## 3    3180   Office Olympics     250
## 4    3181          The Fire     250
## 5    3182         Halloween     250
```

What if we wanted to know the name of the album? The album name is in the Album table.

```
dbGetQuery(chinook_db, "select * from Album where AlbumId = 250")
```

```
##   AlbumId                Title ArtistId
## 1     250 The Office, Season 2      156
```

Remember that:

- AlbumId in Album is the primary key
- AlbumId in Track is a foreign key to the AlbumId field in Album

Joins

To combine the output , we **join on the keys**.

```
dbGetQuery(chinook_db, "select TrackId, Name, Track.AlbumId, Title from Track
                        inner join Album on Track.AlbumId = Album.AlbumId
                        where Track.AlbumId = 250
                        limit 5")
```

```
##   TrackId              Name AlbumId                Title
## 1    3178       The Dundies     250 The Office, Season 2
## 2    3179 Sexual Harassment     250 The Office, Season 2
## 3    3180   Office Olympics     250 The Office, Season 2
## 4    3181          The Fire     250 The Office, Season 2
## 5    3182         Halloween     250 The Office, Season 2
```

`<Table 1> inner join <Table 2> on <key1> = <key2>`

Why specify `Track.AlbumId`? Since both Track and Album tables have an AlbumId field, we need to clarify to SQL which table we mean.

YT with joins

In a single query, return the track id, track name, album id, mediatype id, and mediatype name of all tracks with AlbumId = 10.

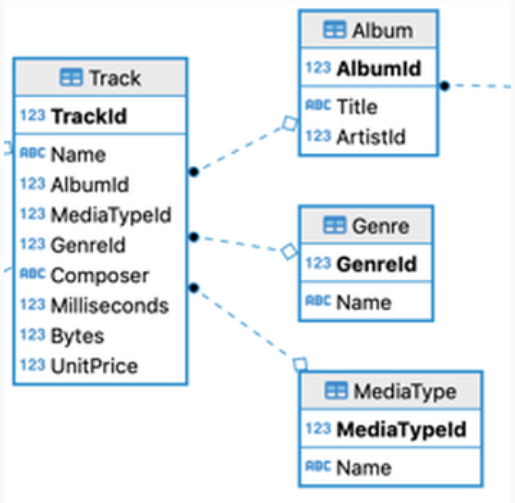
Buildling the query

Return the track id, track name, album id, mediatype id, and mediatype name of all tracks with AlbumId $= 10$.

Want : track id, track name, album id, mediatype id, and mediatype name

Which tables: ?

Any grouping? Any pre/post-aggregration?



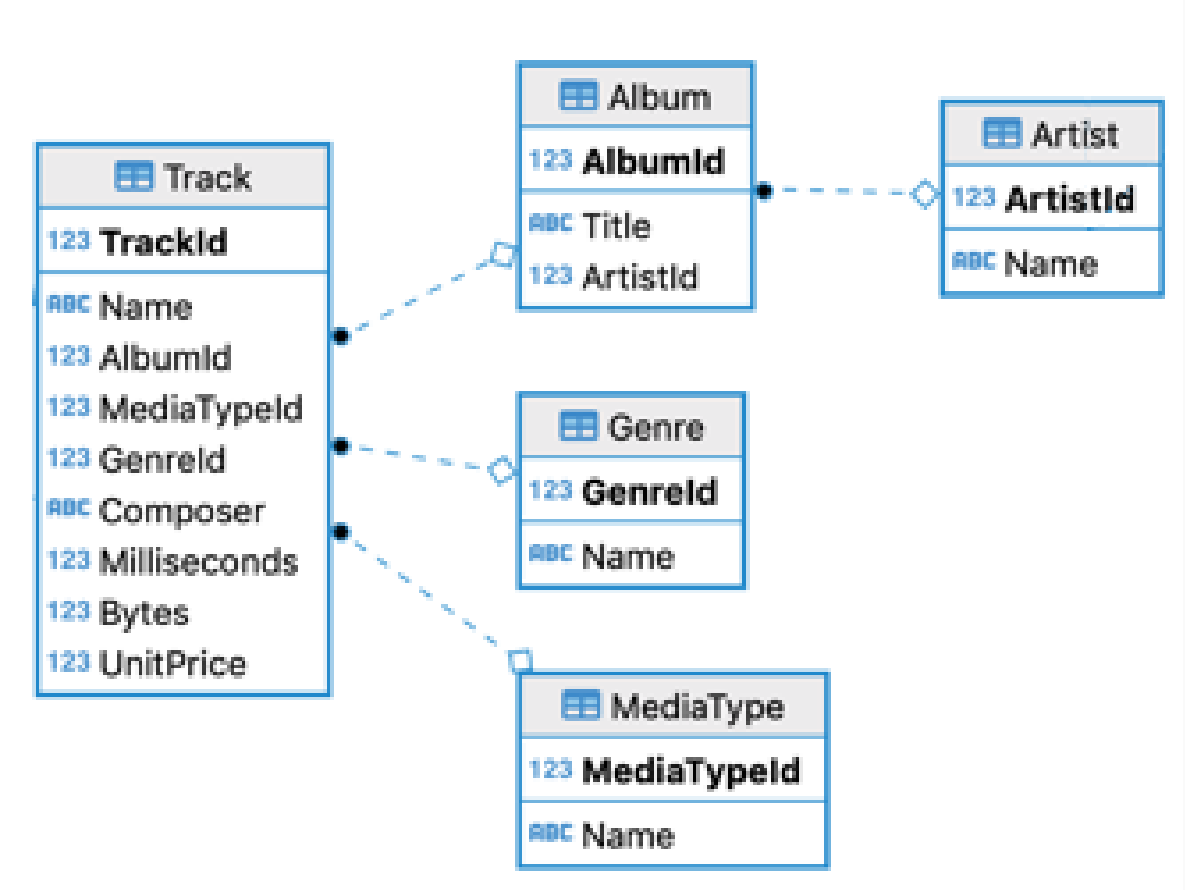
```
dbGetQuery(chinook_db,
"select Track.TrackId, Track.Name, AlbumId, MediaType.MediaTypeId, MediaType.Name
from Track inner join MediaType
    on Track.MediaTypeId = MediaType.MediaTypeId
where AlbumId = 10")
```

```
##    TrackId                    Name AlbumId MediaTypeId            Name
## 1       85                 Cochise      10           1 MPEG audio file
## 2       86      Show Me How to Live      10           1 MPEG audio file
## 3       87                 Gasoline      10           1 MPEG audio file
## 4       88             What You Are      10           1 MPEG audio file
## 5       89             Like a Stone      10           1 MPEG audio file
## 6       90               Set It Off      10           1 MPEG audio file
## 7       91        Shadow on the Sun      10           1 MPEG audio file
## 8       92         I am the Highway      10           1 MPEG audio file
## 9       93                 Exploder      10           1 MPEG audio file
## 10      94                Hypnotize      10           1 MPEG audio file
## 11      95      Bring'em Back Alive      10           1 MPEG audio file
## 12      96             Light My Way      10           1 MPEG audio file
## 13      97              Getaway Car      10           1 MPEG audio file
## 14      98 The Last Remaining Light      10           1 MPEG audio file
```

A Three-way Join

Return the track information and artist information of all tracks with AlbumId $= 10$.

- Note that Album.ArtistId is a foreign key to Artist.ArtistId.



A Three-way Join

```
dbGetQuery(chinook_db,
"select TrackId, Track.Name, Track.AlbumId, Title, Artist.Name
 from Track
inner join Album on Track.AlbumId = Album.AlbumId
inner join Artist on Album.ArtistId = Artist.ArtistId
where Track.AlbumId = 250
                         limit 5")
```

```
##   TrackId              Name AlbumId                Title       Name
## 1    3178       The Dundies     250 The Office, Season 2 The Office
## 2    3179 Sexual Harassment     250 The Office, Season 2 The Office
## 3    3180   Office Olympics     250 The Office, Season 2 The Office
## 4    3181          The Fire     250 The Office, Season 2 The Office
## 5    3182         Halloween     250 The Office, Season 2 The Office
```

Table Alias

```
dbGetQuery(chinook_db,
              "select TrackId, t.Name, t.AlbumId, Title, ar.Name
               from Track t
              inner join Album al on t.AlbumId = al.AlbumId
              inner join Artist ar on al.ArtistId = ar.ArtistId
              where t.AlbumId = 250
                          limit 5")
```

```
##   TrackId              Name AlbumId                Title       Name
## 1    3178       The Dundies     250 The Office, Season 2 The Office
## 2    3179 Sexual Harassment     250 The Office, Season 2 The Office
## 3    3180   Office Olympics     250 The Office, Season 2 The Office
## 4    3181          The Fire     250 The Office, Season 2 The Office
## 5    3182         Halloween     250 The Office, Season 2 The Office
```

`TableName alias`

- Track: t
- Album: al
- Artist: ar

Table Alias

The two Name fields are confusing. Let's alias those field names.

```
dbGetQuery(chinook_db,
"select TrackId, t.Name as TrackName, t.AlbumId,
Title as AlbumTitle, ar.Name as AristName
from Track t
inner join Album al on t.AlbumId = al.AlbumId
inner join Artist ar on al.ArtistId = ar.ArtistId
where t.AlbumId = 250
limit 5")
```

```
##   TrackId         TrackName AlbumId           AlbumTitle  AristName
## 1    3178       The Dundies     250 The Office, Season 2 The Office
## 2    3179 Sexual Harassment     250 The Office, Season 2 The Office
## 3    3180   Office Olympics     250 The Office, Season 2 The Office
## 4    3181          The Fire     250 The Office, Season 2 The Office
## 5    3182         Halloween     250 The Office, Season 2 The Office
```

YT: A Three-way Join

Produce the following relation with a three-way join.

```
##   TrackId         TrackName AlbumId           AlbumTitle GenreName
## 1    3178       The Dundies     250 The Office, Season 2  TV Shows
## 2    3179 Sexual Harassment     250 The Office, Season 2  TV Shows
## 3    3180   Office Olympics     250 The Office, Season 2  TV Shows
## 4    3181          The Fire     250 The Office, Season 2  TV Shows
## 5    3182         Halloween     250 The Office, Season 2  TV Shows
```

```
dbGetQuery(chinook_db,
"select TrackId, t.Name as TrackName, t.AlbumId,
Title as AlbumTitle, g.Name as GenreName
from Track t
    inner join Album al on t.AlbumId = al.AlbumId
    inner join Genre g on g.GenreId = t.GenreId
where t.AlbumId = 250
limit 5")
```

```
##   TrackId         TrackName AlbumId           AlbumTitle GenreName
## 1    3178       The Dundies     250 The Office, Season 2  TV Shows
## 2    3179 Sexual Harassment     250 The Office, Season 2  TV Shows
## 3    3180   Office Olympics     250 The Office, Season 2  TV Shows
## 4    3181          The Fire     250 The Office, Season 2  TV Shows
## 5    3182         Halloween     250 The Office, Season 2  TV Shows
```

YT: Aggregation with Join

Retrieve the average track size for all tracks in each genre, along with the name of the genre.

```
##     AvgBytes GenreId          GenreName
## 1    9007374       1               Rock
## 2    9488137       2               Jazz
## 3    9234573       3              Metal
## 4    7691003       4 Alternative & Punk
## 5    2123262       5      Rock And Roll
## 6    8625576       6              Blues
## 7    7710589       7              Latin
## 8    8237493       8             Reggae
## 9    4745668       9                Pop
## 10   8090772      10         Soundtrack
## 11   7239057      11         Bossa Nova
## 12   6160518      12     Easy Listening
## 13   9474752      13        Heavy Metal
## 14   6575926      14           R&B/Soul
## 15  10691926      15  Electronica/Dance
## 16   7129267      16              World
## 17   6534717      17        Hip Hop/Rap
## 18 507078984      18    Science Fiction
## 19 340261678      19           TV Shows
## 20 532930426      20   Sci Fi & Fantasy
## 21 506946967      21              Drama
## 22 316904466      22             Comedy
## 23   5883474      23        Alternative
## 24   5220907      24          Classical
## 25   2861468      25              Opera
```

Aggregation with Join

```
dbGetQuery(chinook_db,
"select avg(Bytes)  as AvgBytes,
       g.GenreId, g.Name as GenreName
from Track t
   inner join Genre g on g.GenreId = t.GenreId
  group by t.GenreId")
```

```
##     AvgBytes GenreId          GenreName
## 1    9007374       1               Rock
## 2    9488137       2               Jazz
## 3    9234573       3              Metal
## 4    7691003       4 Alternative & Punk
## 5    2123262       5      Rock And Roll
## 6    8625576       6              Blues
## 7    7710589       7              Latin
## 8    8237493       8             Reggae
## 9    4745668       9                Pop
## 10   8090772      10         Soundtrack
## 11   7239057      11         Bossa Nova
## 12   6160518      12     Easy Listening
## 13   9474752      13        Heavy Metal
## 14   6575926      14           R&B/Soul
## 15  10691926      15  Electronica/Dance
## 16   7129267      16              World
## 17   6534717      17        Hip Hop/Rap
## 18 507078984      18    Science Fiction
## 19 340261678      19           TV Shows
## 20 532930426      20   Sci Fi & Fantasy
## 21 506946967      21              Drama
## 22 316904466      22             Comedy
```

implicit joins

Some people prefer to do an *implicit join*, selecting the Cartesian product and then using a `where` clause to identify matching keys. They can do this without the `join` keyword by specifying a comma-separated list of tables.

The query below has no `join` keyword!

```
dbGetQuery(chinook_db,
"select al.title, al.artistid, ar.artistid, ar.name
 from Album al, Artist ar
 where al.artistid = ar.artistid
limit 3")
```

```
##                                   Title ArtistId ArtistId   Name
## 1 For Those About To Rock We Salute You        1        1  AC/DC
## 2                      Balls to the Wall        2        2 Accept
## 3                      Restless and Wild        2        2 Accept
```

natural join

Often times, the foreign key matches the primary key of the target table.

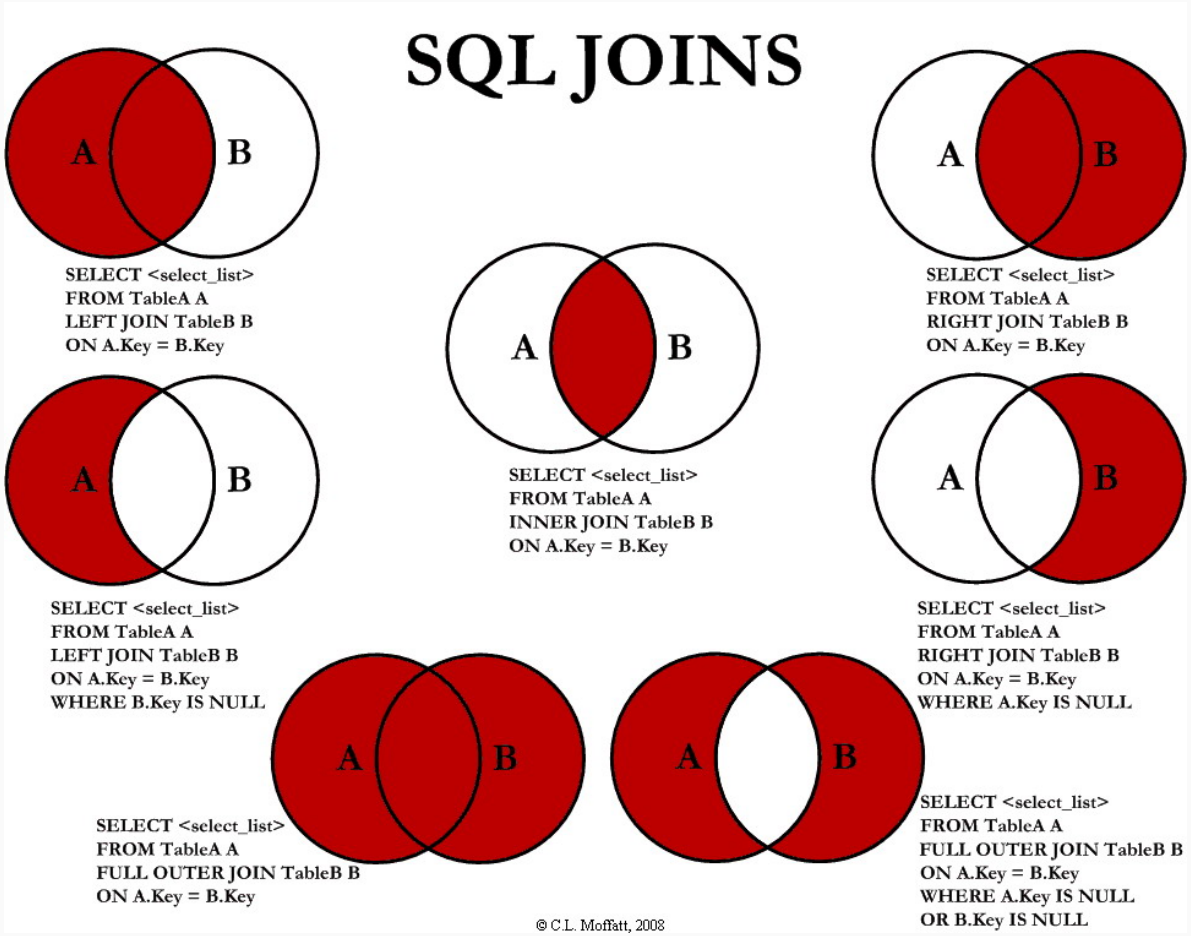
In this case, SQL can infer the key based on the field name via a `natural join`. The query below specifies no keys, but is joined on the matching field name, ArtistId.

```
dbGetQuery(chinook_db,
"select * from album natural join artist limit 3")
```

```
##   AlbumId                                 Title ArtistId   Name
## 1       1 For Those About To Rock We Salute You        1  AC/DC
## 2       2                      Balls to the Wall        2 Accept
## 3       3                      Restless and Wild        2 Accept
```

Avoid natural joins.specify your keys explicitly.

other joins


SQL JOINS
A
B
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
© C.L. Moffatt, 2008

GROUP BY

We can use the `GROUP BY` option in `SELECT` to define aggregation groups

```
dbGetQuery(chinook_db, paste("SELECT AlbumId, AVG(Bytes) AS AvgBytes",
                      "FROM Track",
                      "GROUP BY AlbumId",
                      "ORDER BY AVG(Bytes) DESC",
                      "LIMIT 10"))
```

```
##    AlbumId  AvgBytes
## 1      253 536359244
## 2      229 535292434
## 3      227 529469291
## 4      231 514373372
## 5      228 512231374
## 6      254 492670102
## 7      226 490750393
## 8      261 453454450
## 9      251 306109250
## 10     249 268393262
```

(Note: the order of commands here matters; try switching the order of `GROUP BY` and `ORDER BY`, you'll get an error)

WHERE

```
dbGetQuery(chinook_db,
           paste("SELECT  AlbumId, Avg(Bytes)",
                      "FROM Track",
                      "WHERE AlbumId = 50"
                      ))
```

```
##   AlbumId Avg(Bytes)
## 1      50   30444082
```

We can use the WHERE option in SELECT to specify a subset of the rows to use (*pre-aggregation/pre-calculation*)

```
dbGetQuery(chinook_db,
           paste("SELECT AlbumId, MediaTypeId,AVG(Bytes) as AvgBytes",
                      "FROM Track",
                      "WHERE AlbumId <= 160",
                      "GROUP BY AlbumId",
                      "ORDER BY AvgBytes DESC",
                      "LIMIT 10"))
```

```
##    AlbumId MediaTypeId AvgBytes
## 1       50           1 30444082
## 2      138           1 24822832
## 3      137           1 19120969
## 4       43           1 16221538
## 5       97           1 16089011
## 6      114           1 15975057
## 7      109           1 15934275
## 8      113           1 15521017
## 9      127           1 15194926
```

HAVING

We can use the `HAVING` option in `SELECT` to specify a subset of the rows to display (*post-aggregation/post-calculation*)

```
dbGetQuery(chinook_db,
           paste("SELECT AlbumId, MediaTypeId,AVG(Bytes) as AvgBytes",
                      "FROM Track",
                      "WHERE AlbumId >= 160",
                      "GROUP BY AlbumId",
                      "HAVING AvgBytes >= 25000000",
                      "ORDER BY AVG(Bytes) DESC",
                      "LIMIT 10"))
```

```
##    AlbumId MediaTypeId  AvgBytes
## 1      253           3 536359244
## 2      229           3 535292434
## 3      227           3 529469291
## 4      231           3 514373372
## 5      228           3 512231374
## 6      254           3 492670102
## 7      226           3 490750393
## 8      261           3 453454450
## 9      251           3 306109250
## 10     249           3 268393262
```

What happened?

```
dbGetQuery(chinook_db,
           paste("SELECT AlbumId, MediaTypeId,AVG(Bytes) as AvgBytes",
                      "FROM Track",
                      "WHERE AlbumId >= 160"
           ))
```

```
##   AlbumId MediaTypeId AvgBytes
## 1     160           1 65784686
```

```
dbGetQuery(chinook_db,
           paste("SELECT AlbumId, MediaTypeId,AVG(Bytes) as AvgBytes",
                      "FROM Track",
                      "WHERE AlbumId >= 160",
                      "GROUP BY AlbumId"
                      ))
```

```
##     AlbumId MediaTypeId  AvgBytes
## 1       160           1   5948116
## 2       161           1   7287240
## 3       162           1   8528704
## 4       163           1   6228242
## 5       164           1   6937701
## 6       165           1   7830875
## 7       166           1   7153672
## 8       167           1   7127553
## 9       168           1   6939521
## 10      169           1   7485054
## 11      170           2   4601224
## 12      171           2   5390211
## 13      172           2   5179599
## 14      173           2   5832677
```

Disconnecting from the database

After the end of a session, it is good practice to explicitly close your connection.

```
dbDisconnect(chinook_db)
```

```
# Try selecting data
dbGetQuery(chinook_db,
           "select CustomerId, FirstName, LastName from Customer")
```

```
## Error: Invalid or closed connection
```

We did

- SQl queries involving multiple tables : joins