14. SQL queries

Principles of Data Science with R

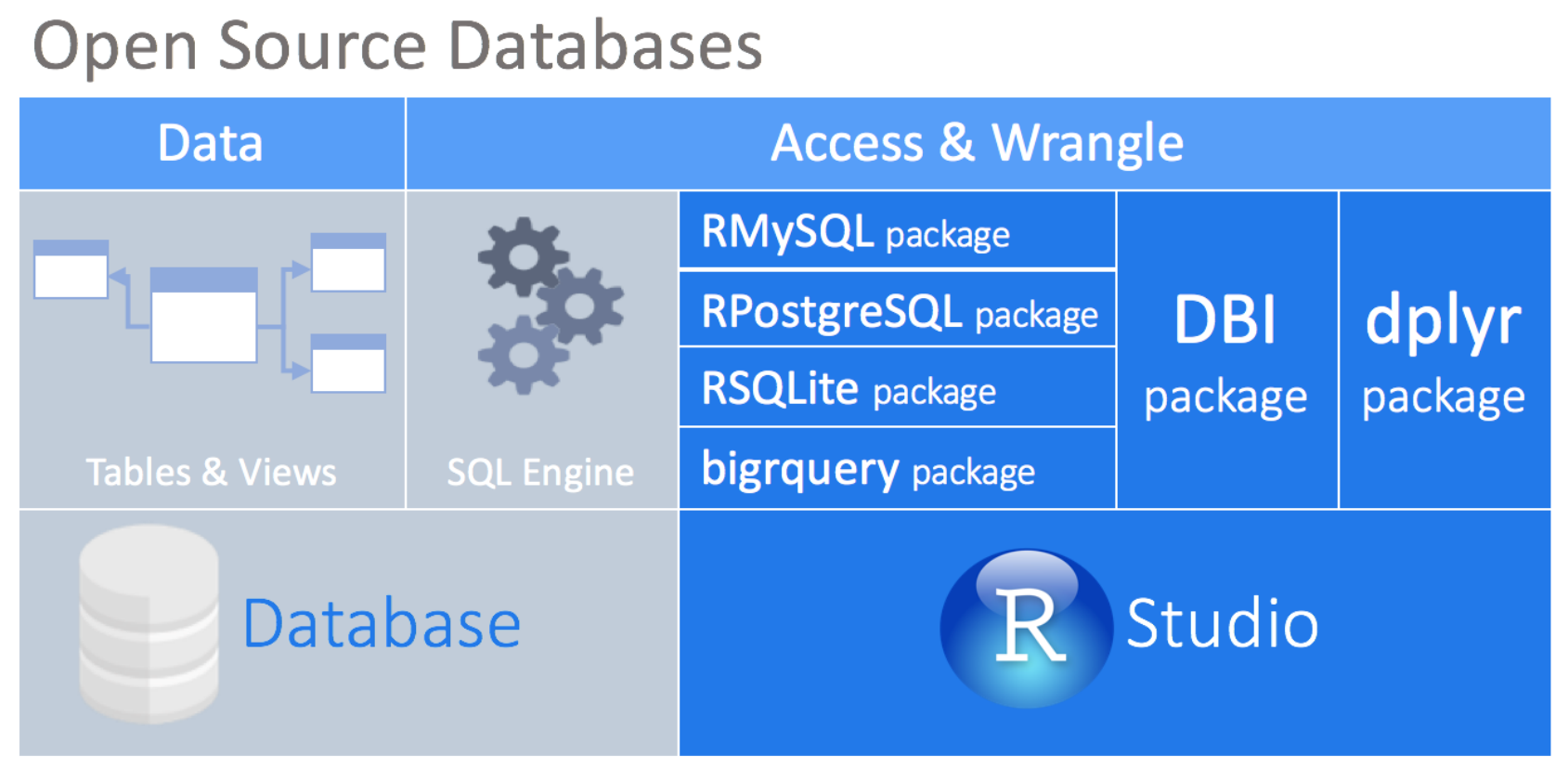
Dr. Uma Ravat

PSTAT 10

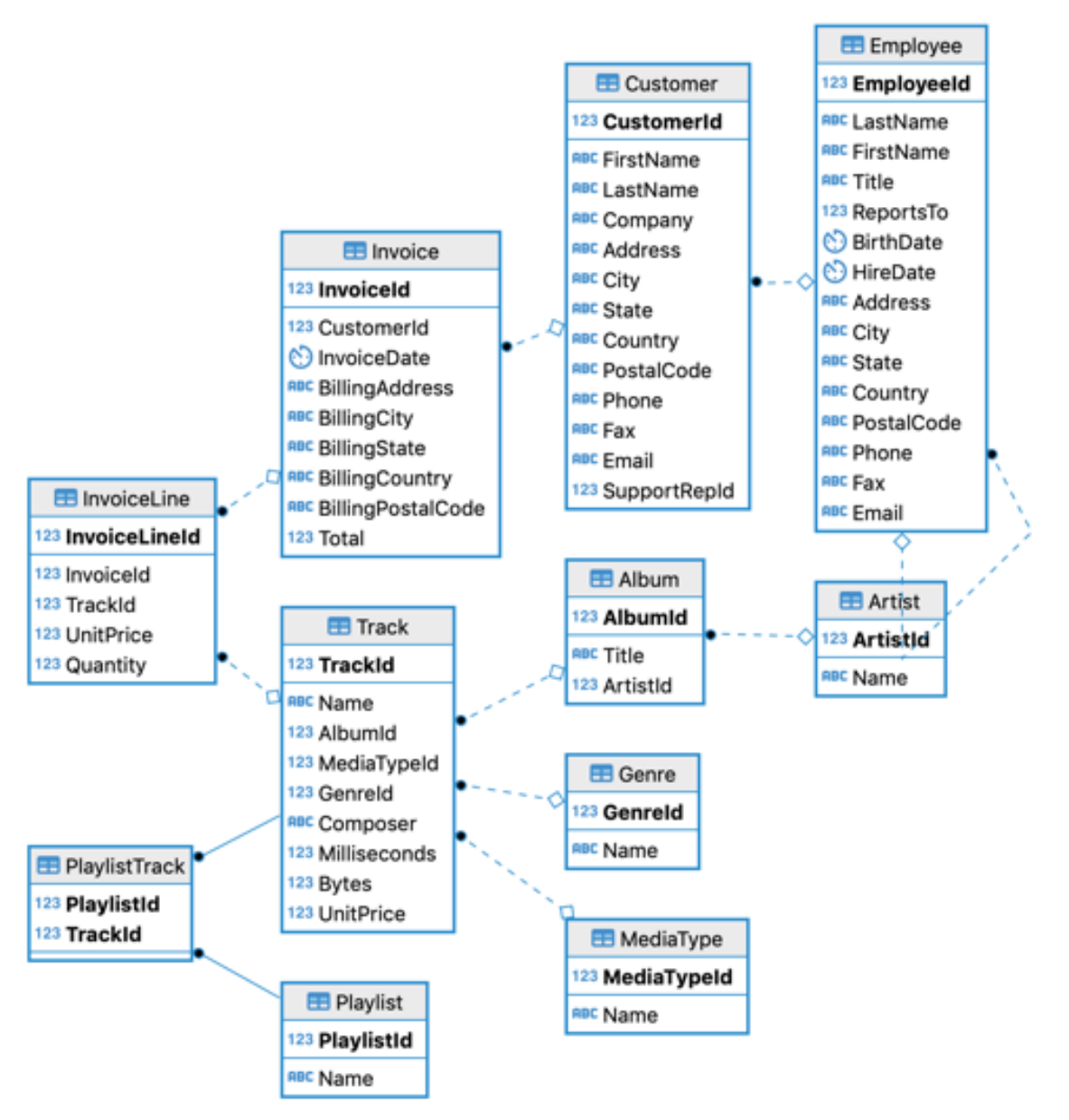
Next we will see...

- More SQL queries

SQLite RDBMS and Rstudio



ChinookDB Entity-Relationship Diagram (ER Diagram)



Connecting to a DB

- (Install) and load the `DBI` (Database interface) package
- (Install) and load DBI complaint DataBase Connectivity driver package for the database you will be using
  - For SQLite RDBMS, we will use the `SQLite()` driver from the `RSQLite` R package
  - For Postgres RDBMS, use the `Postgres()` driver from the `RPostgres` package
  - ...

```
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
```

`chinook_db` is an `R` object that represents a connection to the database file `Chinook_Sqlite.sqlite`

SQL queries

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

`WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT` are all optional

We saw `SELECT`, `FROM`, `ORDER BY`, `LIMIT`

More SQL queries

SELECT

```
dbGetQuery(chinook_db,
                        "SELECT count(*) FROM track")
```

```
##   count(*)
## 1     3503
```

What are all the fields for every track?

```
dbListFields(chinook_db, "track")
```

```
## [1] "TrackId" "Name" "AlbumId" "MediaTypeId" "GenreId"
## [6] "Composer" "Milliseconds" "Bytes" "UnitPrice"
```

```
track_sel <- dbGetQuery(chinook_db,
                        "SELECT * FROM track")
str(track_sel)
```

```
## 'data.frame': 3503 obs. of 9 variables:
## $ TrackId : int 1 2 3 4 5 6 7 8 9 10 ...
## $ Name : chr "For Those About To Rock (We Salute You)" "Balls to the Wall"
"Fast As a Shark" "Restless and Wild" ...
## $ AlbumId : int 1 2 3 3 3 1 1 1 1 1 ...
## $ MediaTypeId : int 1 2 2 2 2 1 1 1 1 1 ...
## $ GenreId : int 1 1 1 1 1 1 1 1 1 1 ...
## $ Composer : chr "Angus Young, Malcolm Young, Brian Johnson" NA "F. Baltes,
S. Kaufman, U. Dirkscneider & W. Hoffman" "F. Baltes, R.A. Smith-Diesel, S.
Kaufman, U. Dirkscneider & W. Hoffman" ...
## $ Milliseconds: int 343719 342562 230619 252051 375418 205662 233926 210834
```

Suppose we only want the first five records for TrackId, Name, AlbumId, Milliseconds, Bytes, UnitPrice from Track table

```
dbGetQuery(chinook_db,
           "SELECT TrackId, Name, AlbumId, Milliseconds, Bytes, UnitPrice
           FROM  track
           limit 5")
```

```
## TrackId Name AlbumId Milliseconds Bytes
## 1 1 For Those About To Rock (We Salute You) 1 343719 11170334
## 2 2 Balls to the Wall 2 342562 5510424
## 3 3 Fast As a Shark 3 230619 3990994
## 4 4 Restless and Wild 3 252051 4331779
## 5 5 Princess of the Dawn 3 375418 6290521
## UnitPrice
## 1 0.99
## 2 0.99
## 3 0.99
## 4 0.99
## 5 0.99
```

SELECT, expanded

In the first line of SELECT, we can directly specify **computations** that we want performed

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

Main tools for computations:

MIN, MAX, COUNT, SUM, AVG or any math formula

Example

To calculate the average Milliseconds, Bytes and Max UnitPrice

```
dbGetQuery(chinook_db,
     "SELECT AVG(Milliseconds), AVG(Bytes), MAX(UnitPrice)
      FROM Track")
```

```
##   AVG(Milliseconds) AVG(Bytes) MAX(UnitPrice)
## 1          393599.2   33510207           1.99
```

To replicate this simple command on an imported data frame:

```
mean(track_sel$Milliseconds, na.rm=TRUE)
```

```
## [1] 393599.2
```

```
mean(track_sel$Bytes, na.rm=TRUE)
```

```
## [1] 33510207
```

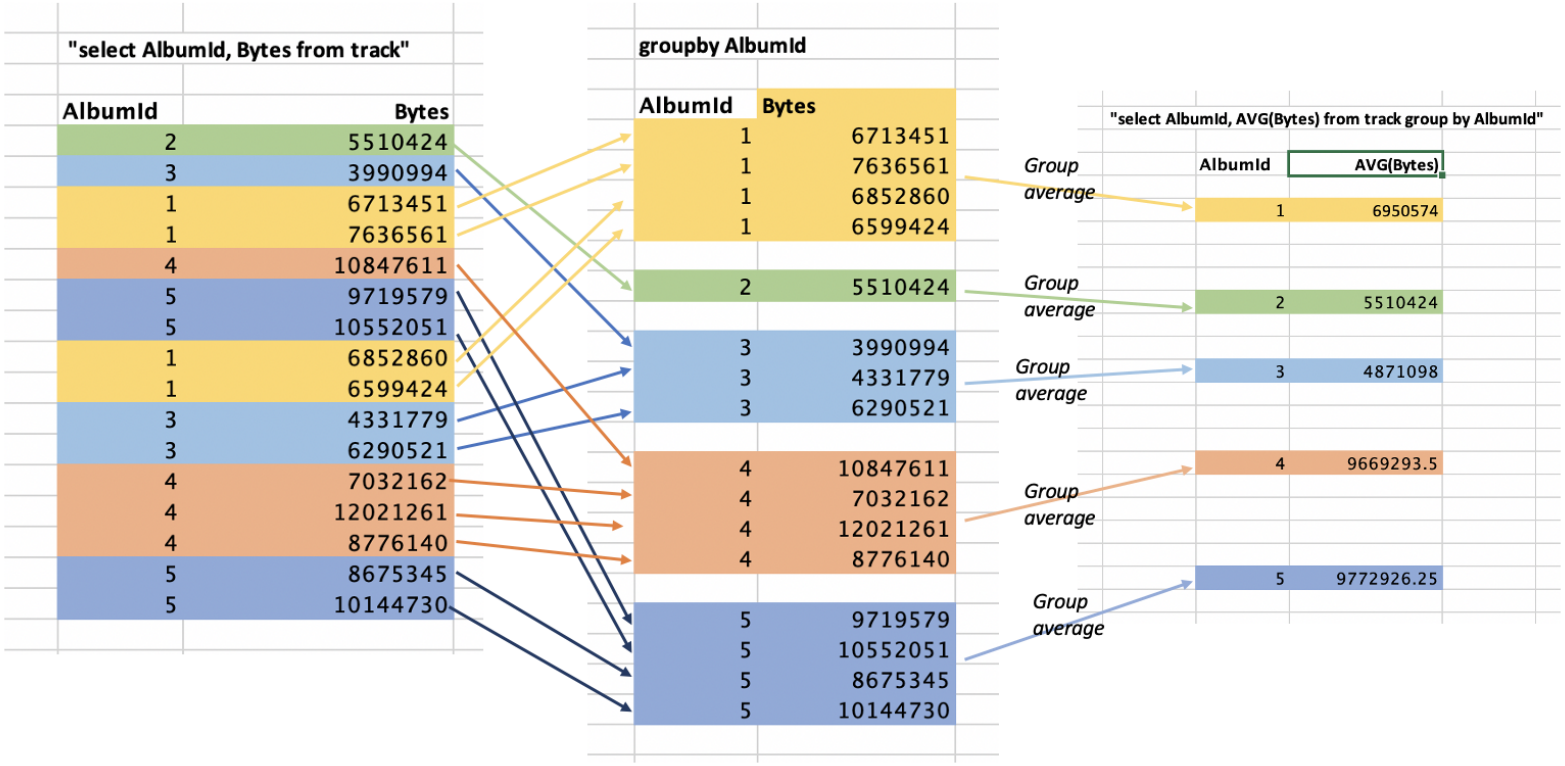
```
max(track_sel$UnitPrice, na.rm=TRUE)
```

```
## [1] 1.99
```

GROUP BY

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

The `GROUP BY` statement groups rows that have the same values and performs aggregation (COUNT(), MAX(), MIN(), SUM(), AVG()) on those groups.



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

```
##    AlbumId AVG(Bytes)
## 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)

AS

We can use `AS` in the first line of `SELECT` to rename computed columns

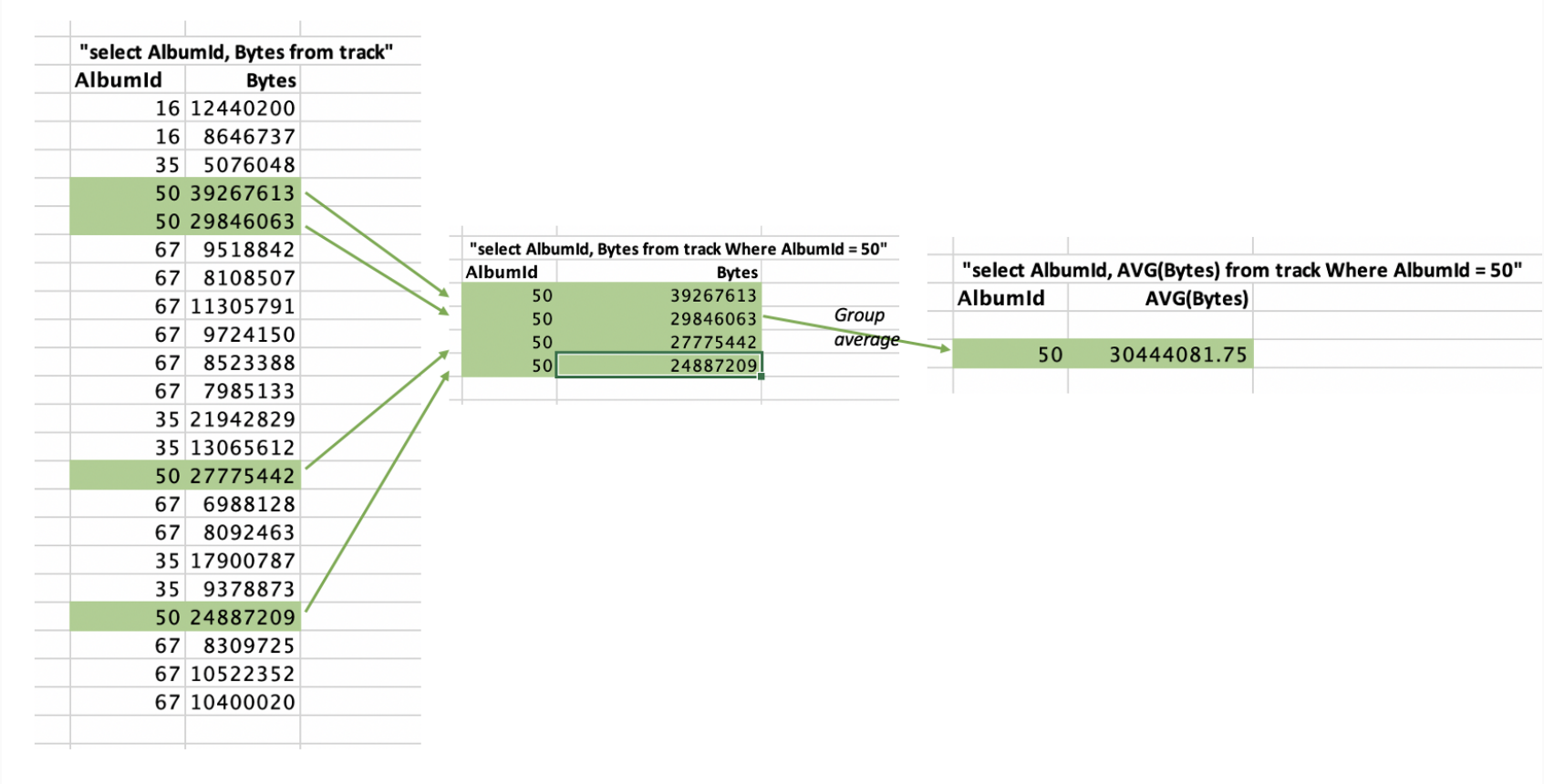
```
dbGetQuery(chinook_db,
           "SELECT AlbumId, AVG(Bytes) AS AvgBytes
            FROM Track
            GROUP BY AlbumId
            ORDER BY AVG(Bytes) DESC
            LIMIT 10")
```

```
##    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
```

WHERE

```
# case insensitive
dbGetQuery(chinook_db, "Select count(DISTINCT(ALBUMID)) FROM track")
```

```
##   count(DISTINCT(ALBUMID))
## 1                      347
```



WHERE

```
dbGetQuery(chinook_db,
           "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,
           "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
## 10      98           1 14851676
```

Note we used the alias AvgBytes for AVG(BYTES) in the ORDER BY.

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,
           "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
```

WHERE condition

Pattern Matching on String Operators:

- **LIKE operator:** Allows wildcards to be used
  - % : matches any sequence of zero or more characters
  - _ : matches any single character
- the p% pattern will match any strings that begin with p
  - e.g.: pstat, pineapple, pop
- the %al pattern matches any string that ends with al
  - e.g.: pal, bridal, opal
- the %ul% pattern matches any string that contains ul
  - e.g.: ultimate and forceful
- the pattern r_n will match the strings run, ran, ron
- the pattern ___rd matches the strings yard, ward, herd, etc.
  - then what is the difference between ___rd and %rd ?
  - %rd would also match longer words ending in rd, like bernard, heard, etc

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
```

Does this remove the database connection `chinook_db` in the R session?

We saw

- Integrity constraints (Entity integrity, Referential integrity)
- Keys: 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`