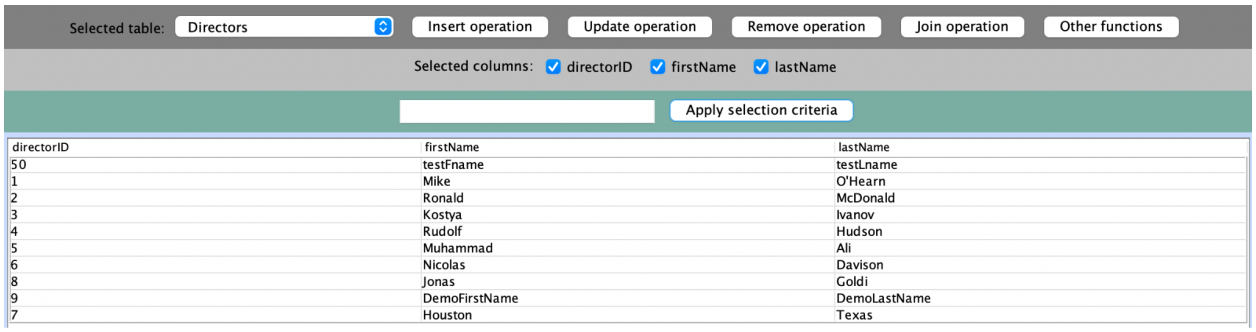
Queries we accomplished:

Queries: INSERT Operation 0.5 Points

Example 1) Inserting into Director

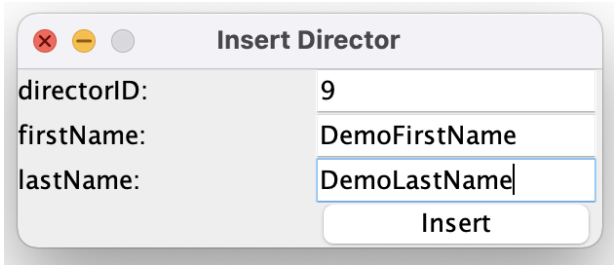
Before insertion

```
SELECT *
FROM Director;
```



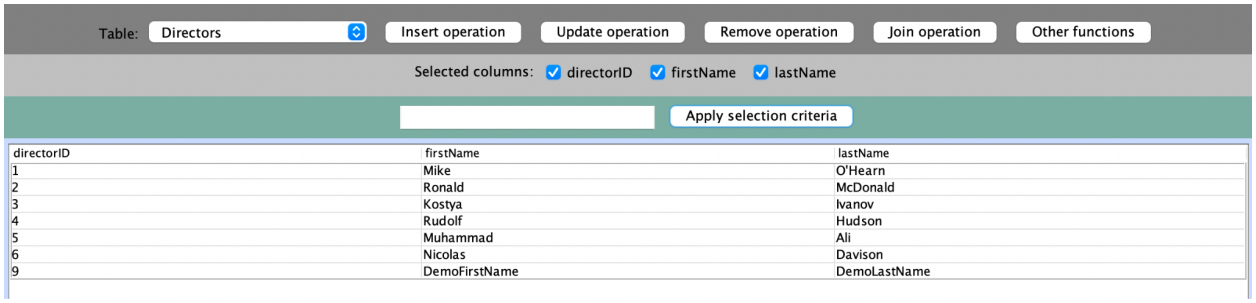
directorID	firstName	lastName
50	testFname	testLname
1	Mike	O'Hearn
2	Ronald	McDonald
3	Kostya	Ivanov
4	Rudolf	Hudson
5	Muhammad	Ali
6	Nicolas	Davison
8	Jonas	Goldi
9	DemoFirstName	DemoLastName
7	Houston	Texas

```
INSERT INTO Director(directorID, firstName, lastName)
VALUES (9, 'DemoFirstName', 'DemoLastName');
```



After insertion

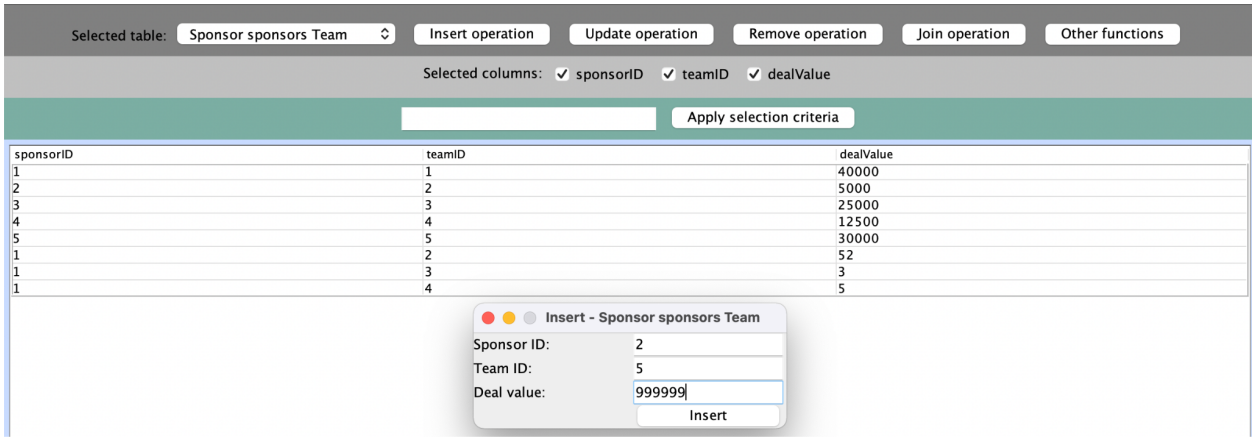
```
SELECT *
FROM Director;
```



directorID	firstName	lastName
1	Mike	O'Hearn
2	Ronald	McDonald
3	Kostya	Ivanov
4	Rudolf	Hudson
5	Muhammad	Ali
6	Nicolas	Davison
9	DemoFirstName	DemoLastName

We also have Insert for the table “Sponsor sponsors Team”

```
SELECT *
FROM SponsorsTeam;
```



sponsorID	teamID	dealValue
1	1	40000
2	2	5000
3	3	25000
4	4	12500
5	5	30000
1	2	52
1	3	3
1	4	5

Clicking on the Insert button executes this query:

```
INSERT INTO SponsorsTeam(sponsorID, teamID, dealValue)
VALUES (2, 5, 999999);
```

sponsorID	teamID	dealValue
1	1	40000
2	2	5000
3	3	25000
4	4	12500
5	5	30000
2	5	999999
1	2	52
1	3	3
1	4	5

The sql table contains 1 more row with the input values.

<u>Where to find the functionality for these 2 inserts?</u>

Add director window made possible by:

src/database/directorHandler

-> public void **insertDirector**(Director dir)

UI backend call inside of **src/ui/InsertFrame**

In the **actionPerformed**(ActionEvent e) **method**

Add SponsorsTeam window made possible by:

src/database/SponsorsTeamHandler

-> public void **insertSponsorsTeam**(SponsorsTeam st)

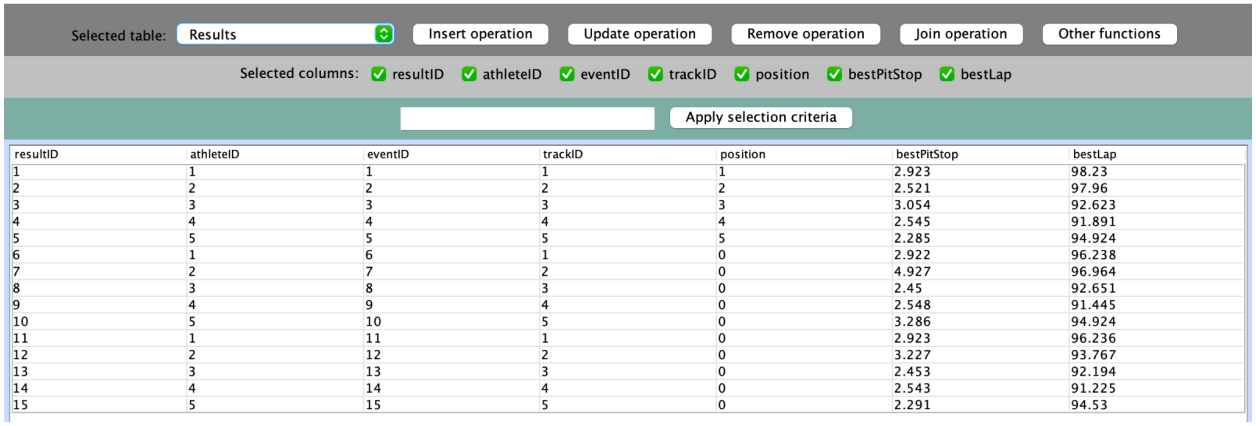
UI backend call inside of src/ui/InsertFrameSponsorsTeam

In the **actionPerformed**(ActionEvent e) **method**

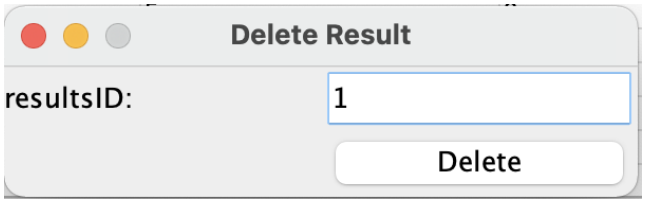
Queries: DELETE Operation

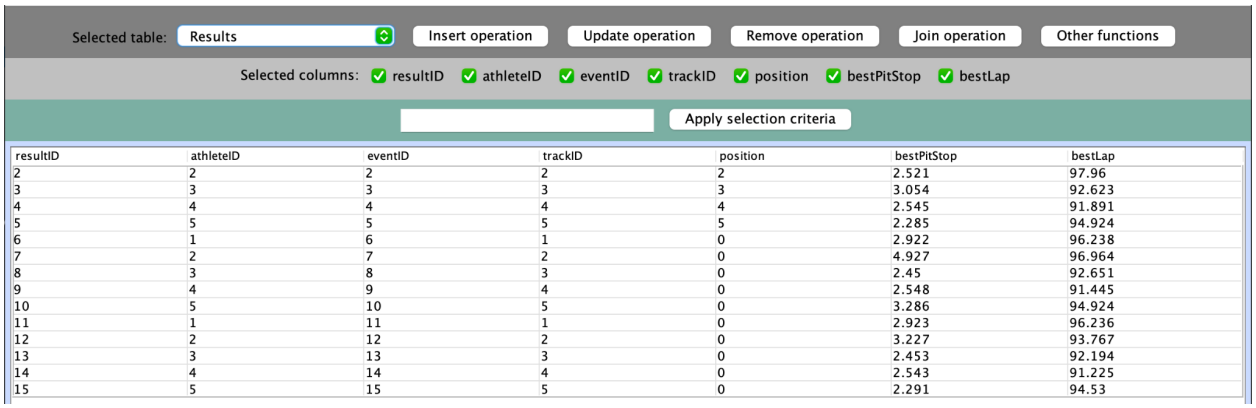
- src.database.ResultsHandler.deleteResults() contains this functionality

```
DELETE FROM Results
WHERE resultID='1';
```



resultID	athleteID	eventID	trackID	position	bestPitStop	bestLap
1	1	1	1	1	2.923	98.23
2	2	2	2	2	2.521	97.96
3	3	3	3	3	3.054	92.623
4	4	4	4	4	2.545	91.891
5	5	5	5	5	2.285	94.924
6	1	6	1	0	2.922	96.238
7	2	7	2	0	4.927	96.964
8	3	8	3	0	2.45	92.651
9	4	9	4	0	2.548	91.445
10	5	10	5	0	3.286	94.924
11	1	11	1	0	2.923	96.236
12	2	12	2	0	3.227	93.767
13	3	13	3	0	2.453	92.194
14	4	14	4	0	2.543	91.225
15	5	15	5	0	2.291	94.53





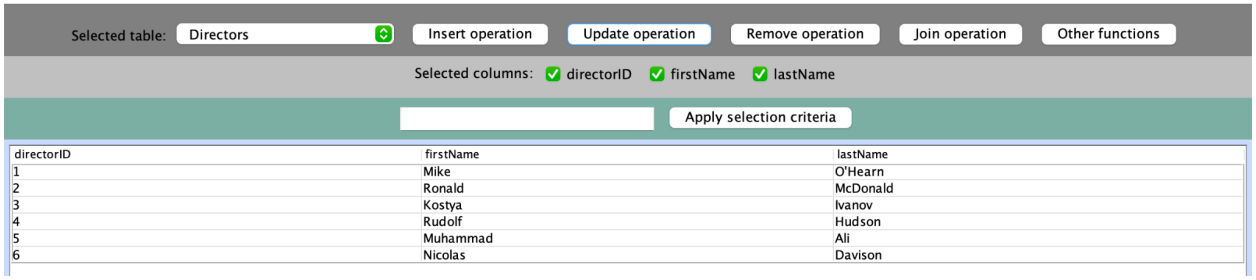
resultID	athleteID	eventID	trackID	position	bestPitStop	bestLap
2	2	2	2	2	2.521	97.96
3	3	3	3	3	3.054	92.623
4	4	4	4	4	2.545	91.891
5	5	5	5	5	2.285	94.924
6	1	6	1	0	2.922	96.238
7	2	7	2	0	4.927	96.964
8	3	8	3	0	2.45	92.651
9	4	9	4	0	2.548	91.445
10	5	10	5	0	3.286	94.924
11	1	11	1	0	2.923	96.236
12	2	12	2	0	3.227	93.767
13	3	13	3	0	2.453	92.194
14	4	14	4	0	2.543	91.225
15	5	15	5	0	2.291	94.53

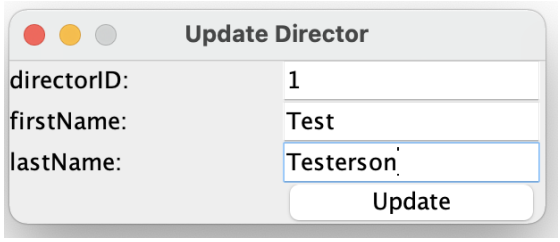
Queries: UPDATE Operation

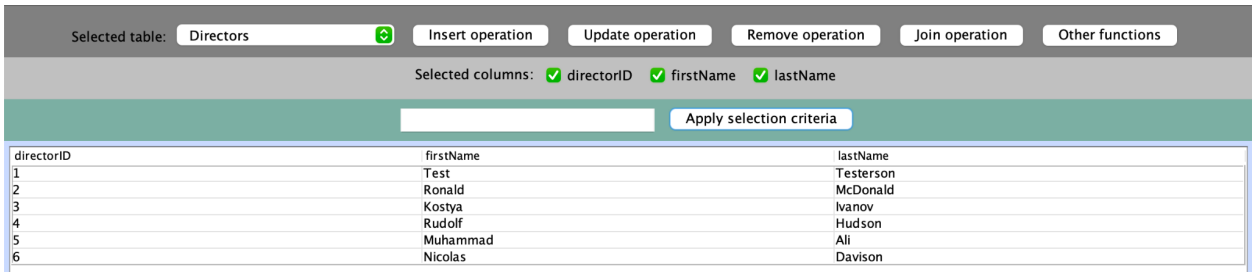
- src.database.directorHandler.updateDirector() contains this functionality

```
UPDATE Directors
SET firstName='Test', lastName='Testerson'
WHERE directorID='1';
```

Eduardo Freitas







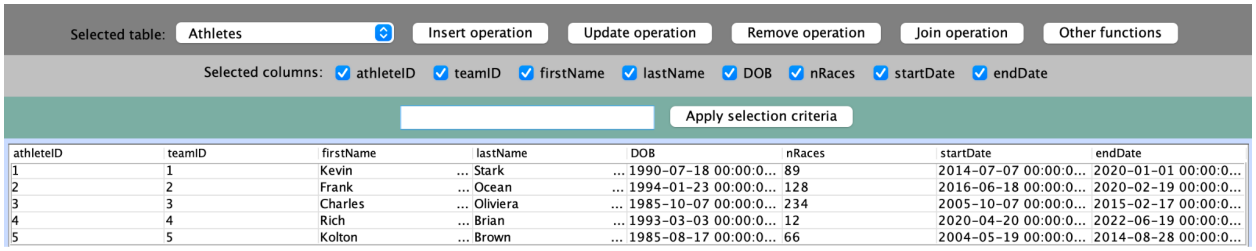
Queries: Selection

Our application supports Selection using criteria to filter the out table.

Example 1) Selecting all athletes who participated in more than 70 races

Here, we are doing

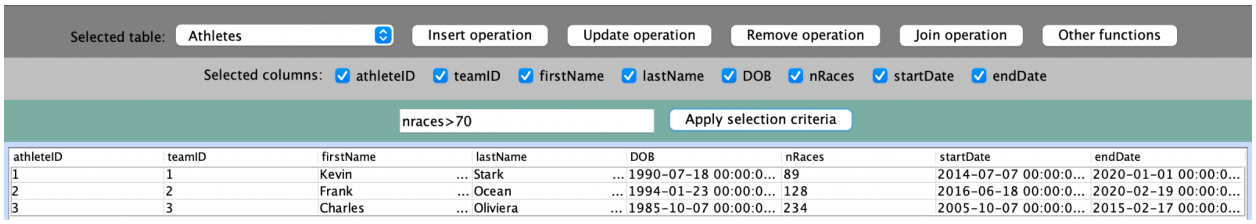
```
SELECT *
FROM Athlete;
```



athleteID	teamID	firstName	...	lastName	...	DOB	nRaces	startDate	endDate
1	1	Kevin	...	Stark	...	1990-07-18 00:00:0...	89	2014-07-07 00:00:0...	2020-01-01 00:00:0...
2	2	Frank	...	Ocean	...	1994-01-23 00:00:0...	128	2016-06-18 00:00:0...	2020-02-19 00:00:0...
3	3	Charles	...	Oliviera	...	1985-10-07 00:00:0...	234	2005-10-07 00:00:0...	2015-02-17 00:00:0...
4	4	Rich	...	Brian	...	1993-03-03 00:00:0...	12	2020-04-20 00:00:0...	2022-06-19 00:00:0...
5	5	Kolton	...	Brown	...	1985-08-17 00:00:0...	66	2004-05-19 00:00:0...	2014-08-28 00:00:0...

If we specify this selection criteria, we are doing

```
SELECT * FROM Athlete
WHERE nraces>70;
```

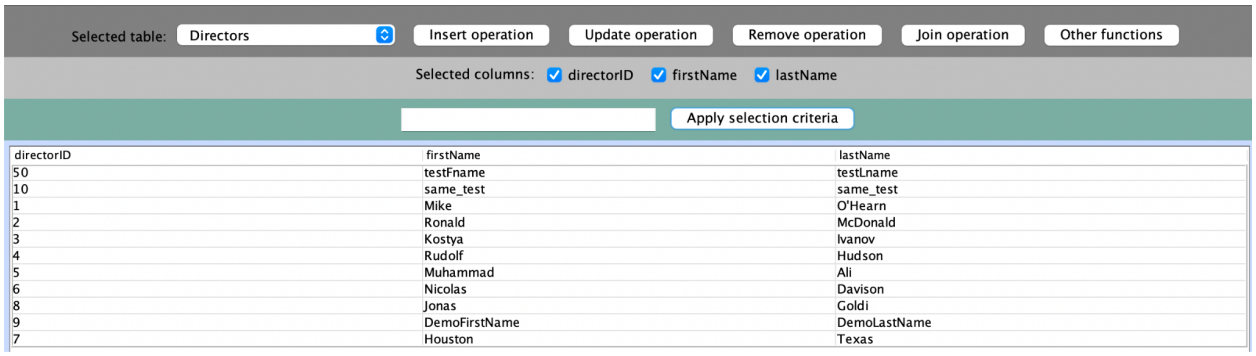


athleteID	teamID	firstName	...	lastName	...	DOB	nRaces	startDate	endDate
1	1	Kevin	...	Stark	...	1990-07-18 00:00:0...	89	2014-07-07 00:00:0...	2020-01-01 00:00:0...
2	2	Frank	...	Ocean	...	1994-01-23 00:00:0...	128	2016-06-18 00:00:0...	2020-02-19 00:00:0...
3	3	Charles	...	Oliviera	...	1985-10-07 00:00:0...	234	2005-10-07 00:00:0...	2015-02-17 00:00:0...

We can see that the only Athletes left are the ones that satisfy the selection criteria.

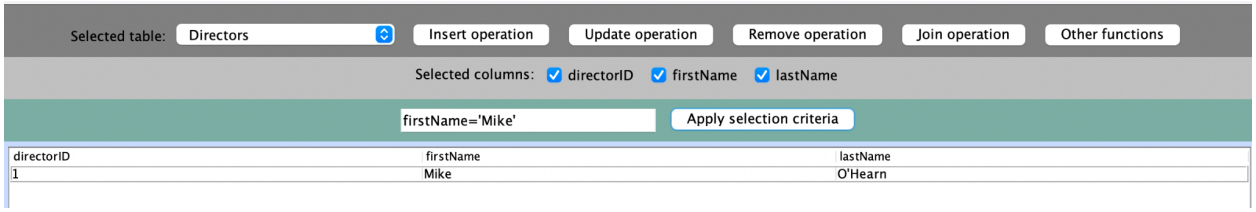
Example 2) Selecting all directors whose first name is “Mike”

```
SELECT *
FROM Director;
```



directorID	firstName	lastName
50	testFname	testLname
10	same_test	same_test
1	Mike	O'Hearn
2	Ronald	McDonald
3	Kostya	Ivanov
4	Rudolf	Hudson
5	Muhammad	Ali
6	Nicolas	Davison
8	Jonas	Goldi
9	DemoFirstName	DemoLastName
7	Houston	Texas

```
SELECT *
FROM Director
WHERE firstName = 'Mike';
```



directorID	firstName	lastName
1	Mike	O'Hearn

Where to find the functionality for projection?

The “Apply selection criteria” is created inside of the **src/ui/HomeWindow** class which implements the ActionListener. When the button is pressed, the actionPerformed function calls the **handleTable** function inside **src/ui/TableComboBox**.

The **TableComboBox** has the “selectedColumns” and “AllColumns” array list for storing the current selected columns, which are useful when calling the handleTable function from the **HomeWindow** class, as the program is able to query the correct table in the dropdown menu, with the correct columns together with the selection criteria taken from the text field.

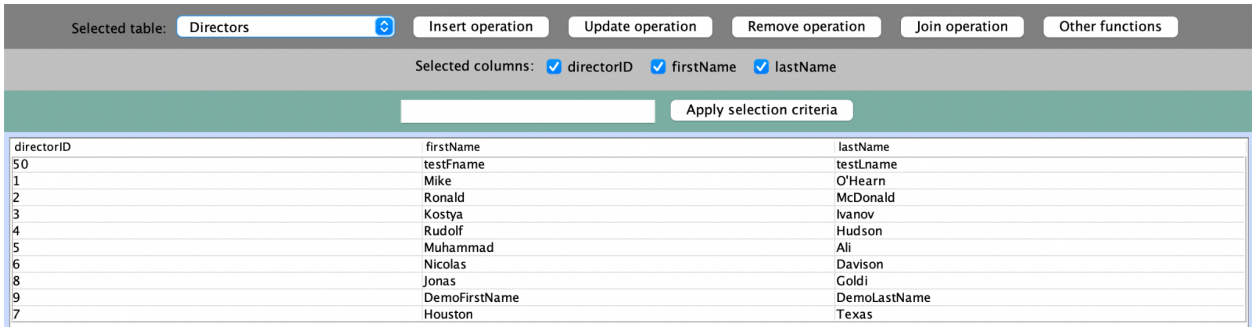
The handleTable function is using the DatabaseConnectionHandler object to ensure connection is established with the Oracle database, the query is formatted correctly with the **buildSelect** function written by Taylor inside the Query builder class, and then executes the prepared statement.

Queries: Projection

Our application supports projection of any combination of columns for any table.

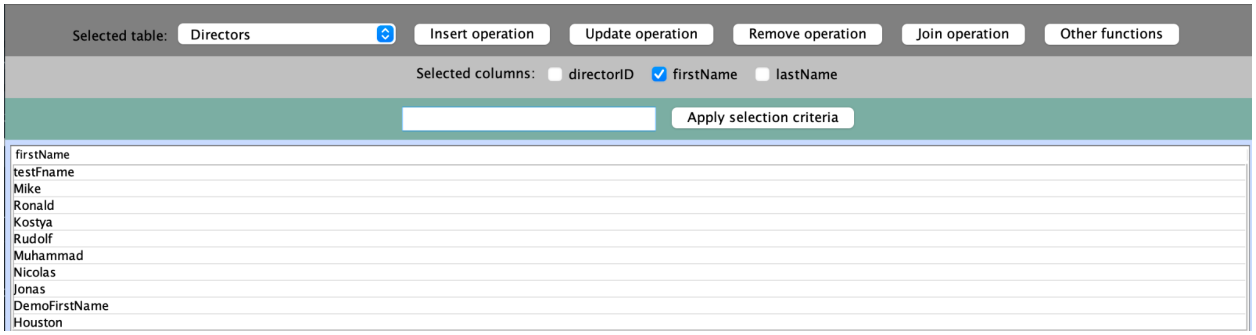
Example 1: Demonstration on Directors table

```
SELECT directorID, firstName, lastName
FROM Director;
```



directorID	firstName	lastName
50	testFname	testLname
1	Mike	O'Hearn
2	Ronald	McDonald
3	Kostya	Ivanov
4	Rudolf	Hudson
5	Muhammad	Ali
6	Nicolas	Davison
8	Jonas	Goldi
9	DemoFirstName	DemoLastName
7	Houston	Texas

```
SELECT firstName
FROM Director;
```

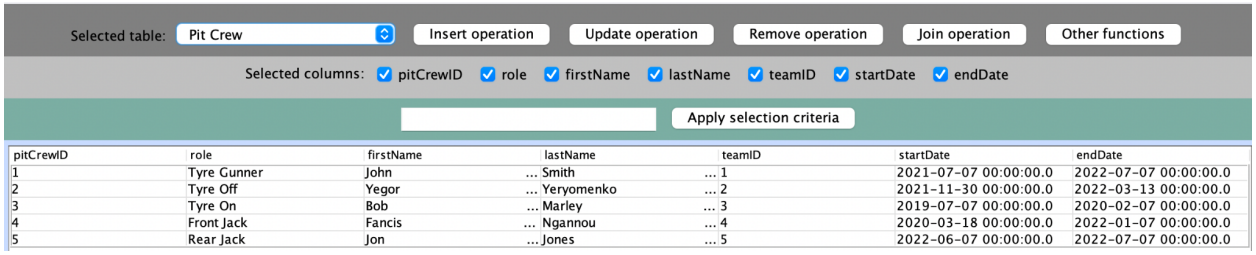


firstName
testFname
Mike
Ronald
Kostya
Rudolf
Muhammad
Nicolas
Jonas
DemoFirstName
Houston

Only the selected attributes are kept.

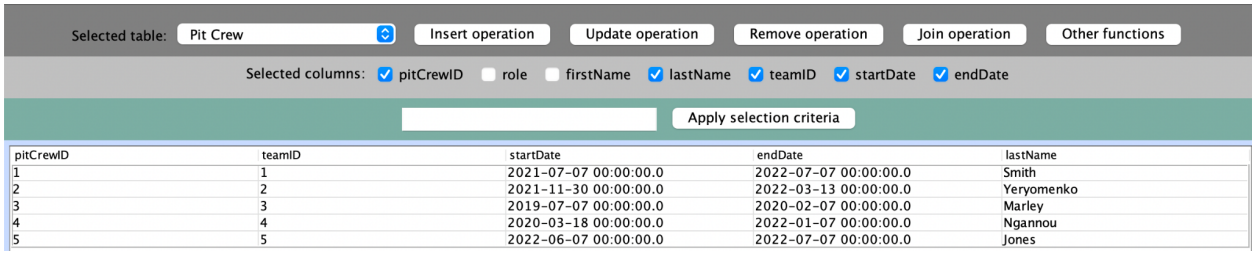
Example 2: Demonstrating on PitCrew table

```
SELECT *
FROM PitCrew;
```



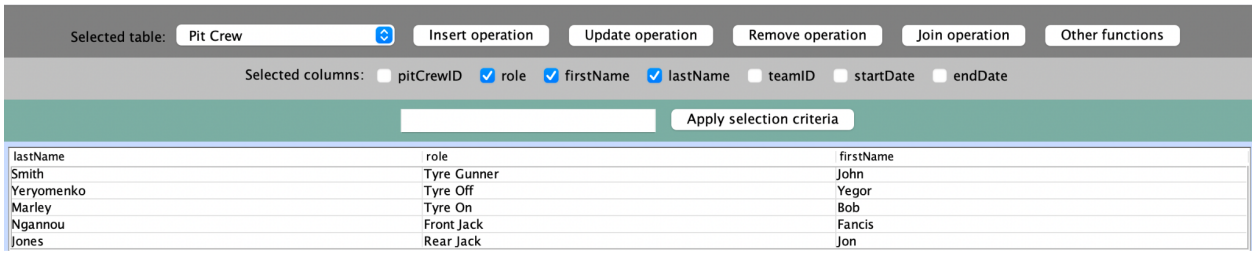
pitCrewID	role	firstName	lastName	teamID	startDate	endDate
1	Tyre Gunner	John	... Smith	... 1	2021-07-07 00:00:00.0	2022-07-07 00:00:00.0
2	Tyre Off	Yegor	... Yeryomenko	... 2	2021-11-30 00:00:00.0	2022-03-13 00:00:00.0
3	Tyre On	Bob	... Marley	... 3	2019-07-07 00:00:00.0	2020-02-07 00:00:00.0
4	Front Jack	Fancis	... Ngannou	... 4	2020-03-18 00:00:00.0	2022-01-07 00:00:00.0
5	Rear Jack	Jon	... Jones	... 5	2022-06-07 00:00:00.0	2022-07-07 00:00:00.0

```
SELECT pitCrewID, lastName, teamID, startDate, endDate
FROM PitCrew;
```



pitCrewID	teamID	startDate	endDate	lastName
1	1	2021-07-07 00:00:00.0	2022-07-07 00:00:00.0	Smith
2	2	2021-11-30 00:00:00.0	2022-03-13 00:00:00.0	Yeryomenko
3	3	2019-07-07 00:00:00.0	2020-02-07 00:00:00.0	Marley
4	4	2020-03-18 00:00:00.0	2022-01-07 00:00:00.0	Ngannou
5	5	2022-06-07 00:00:00.0	2022-07-07 00:00:00.0	Jones

```
SELECT role, firstName, lastName
FROM PitCrew;
```



lastName	role	firstName
Smith	Tyre Gunner	John
Yeryomenko	Tyre Off	Yegor
Marley	Tyre On	Bob
Ngannou	Front Jack	Fancis
Jones	Rear Jack	Jon

Where to find the functionality for projection?

Just like the selection implementation for the HomeWindow table is implemented inside the **src/ui/util/TableComboBox,** by the same token, the projection code is located in the same java file (**src/ui/util/TableComboBox).**

Inside the TableComboBox class, we made a private class called **AttributeCheckbox** that extends JCheckBox and implements ActionListener.
It is responsible for ensuring that the List of “selected checkbox columns” stays updated, and it also has helpers other functions inside **TableComboBox.java** that utilize them for clearing and building new attribute columns when the user switches table.

Since the **AttributeCheckbox** implements an Action listener, every time any checkbox is clicked, the correct columns are either removed or added to the “selectedColumns” and then the

Method skeleton: handleTable(String table_name, ArrayList<String> columns, String criteria)
handleTable(DetermineTable(comboBox.getSelectedItem().toString()), selectedColumns, "") function is called.

The table_name is the current table, the columns are the updated list of selected attribute columns to query, and the criteria is an empty string because we are doing a “naked selection” i.e.
Projection operator is simply a “SELECT * FROM table” sql query but without the “WHERE” criteria.

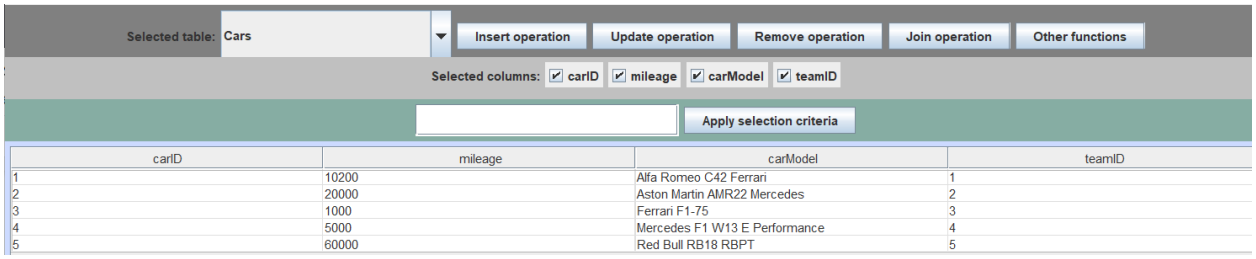
Inside the handleTable, the line of code that does the querying call is
Object[][] result = **DatabaseConnectionHandler**.getHandler().select(table_name, columns, criteria);

The select(table_name, columns, criteria) function uses QueryBuilder.buildSelect to format the query call into a string, followed by the standard procedure of using PreparedStatement and ResultSet to execute the actual query.

Queries: Join

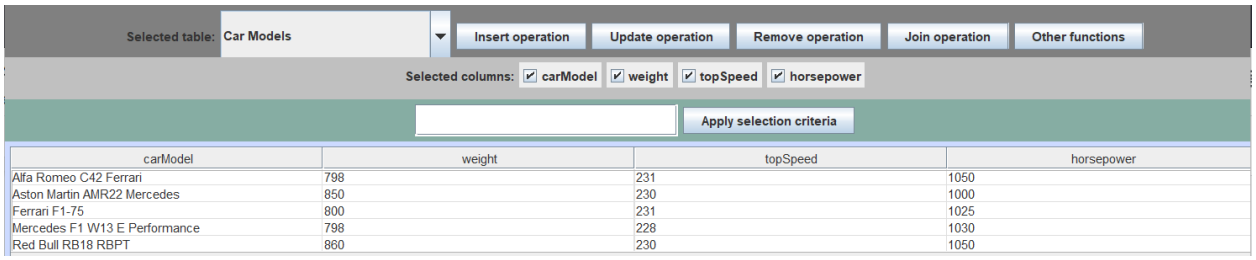
Natural join of Car and CarModel where horsepower of the car model is more than 1025

```
SELECT *
FROM Car
NATURAL JOIN CarModel
WHERE horsepower>1025
```



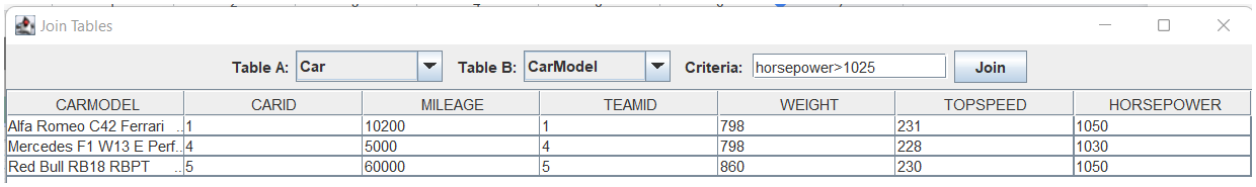
carID	mileage	carModel	teamID
1	10200	Alfa Romeo C42 Ferrari	1
2	20000	Aston Martin AMR22 Mercedes	2
3	1000	Ferrari F1-75	3
4	5000	Mercedes F1 W13 E Performance	4
5	60000	Red Bull RB18 RBPT	5

Showing cars



carModel	weight	topSpeed	horsepower
Alfa Romeo C42 Ferrari	798	231	1050
Aston Martin AMR22 Mercedes	850	230	1000
Ferrari F1-75	800	231	1025
Mercedes F1 W13 E Performance	798	228	1030
Red Bull RB18 RBPT	860	230	1050

Showing car models



CARMODEL	CARID	MILEAGE	TEAMID	WEIGHT	TOPSPEED	HORSEPOWER
Alfa Romeo C42 Ferrari	1	10200	1	798	231	1050
Mercedes F1 W13 E Perf..	4	5000	4	798	228	1030
Red Bull RB18 RBPT	5	60000	5	860	230	1050

Showing results of join on Car and CarModel with criteria of horsepower greater than 1025

<u>Where to find?</u>

The Join Tables frame is located in **src/ui/JoinFrame**, it is launched from **src/ui/HomeWindow** in **`actionPerformed`** via **`new JoinFrame`**

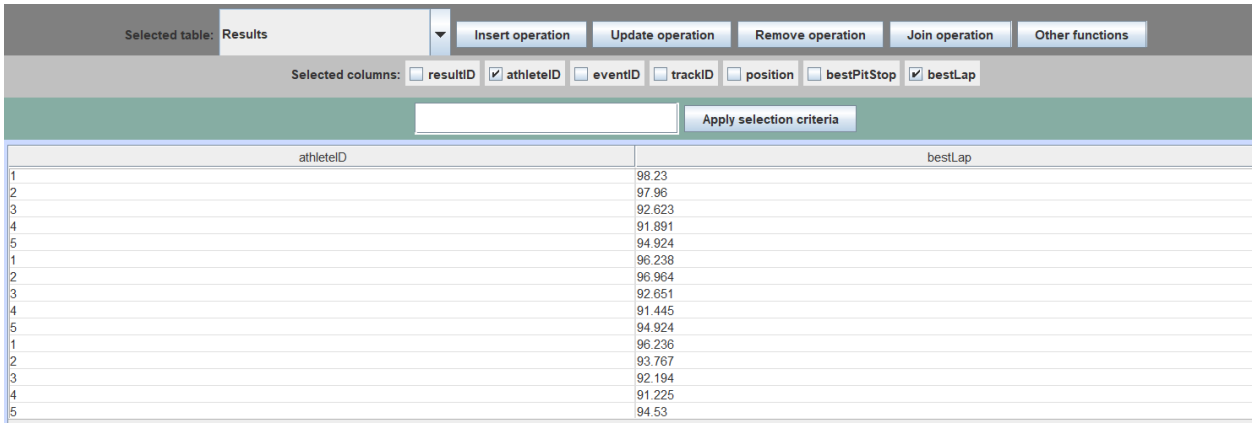
When the join button is pressed from “Join Tables” Frame, the **`joinButtonPress()`** is called which calls the **`join()`** method in **src/database/DatabaseConnectionHandler**

The **`join()`** delegates the query string creation to the **`buildJoin()`** method located in **`src/database/QueryBuilder`**, and then executes the prepared statement. The oracle database data that is returned back is then displayed as a table by **`processTable`** method in **src/ui/JoinFrame**

Queries: Aggregation with Group By

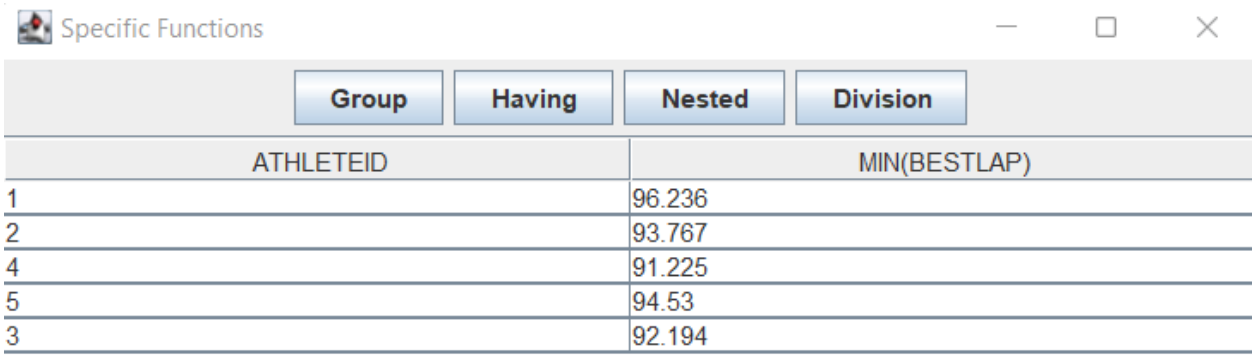
Find each athlete's best lap time

```
SELECT athleteID, MIN(bestLap)
FROM results
GROUP BY athleteID
```



athleteID	bestLap
1	98.23
2	97.96
3	92.623
4	91.891
5	94.924
1	96.238
2	96.964
3	92.651
4	91.445
5	94.924
1	96.236
2	93.767
3	92.194
4	91.225
5	94.53

Showing all athlete best laps.



ATHLETEID	MIN(BESTLAP)
1	96.236
2	93.767
4	91.225
5	94.53
3	92.194

Showing minimum best lap time when grouped by athleteID.

<u>Where to find?</u>

The sql query string is located inside **src/database/QueryBuilder** `AGG_BY_GROUP_QUERY`

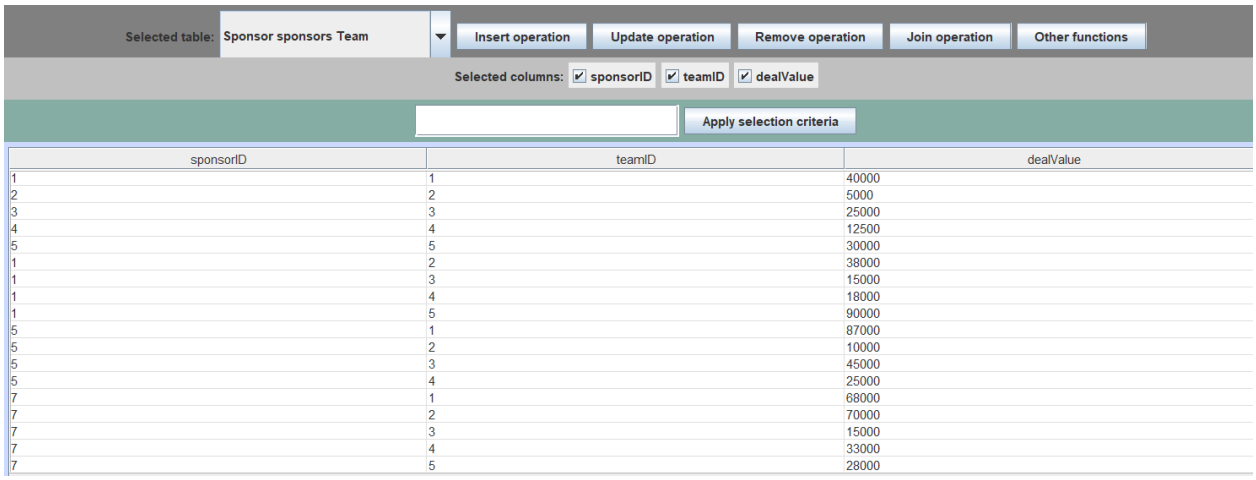
The query is executed by **`aggByGroup()`** inside **src/database/DatabaseConnection/Handler**

The **`aggByGroup()`** function is called by **`groupButtonPress()`** inside **sr/ui/FunctionsFrame**

Queries: Aggregation with Having

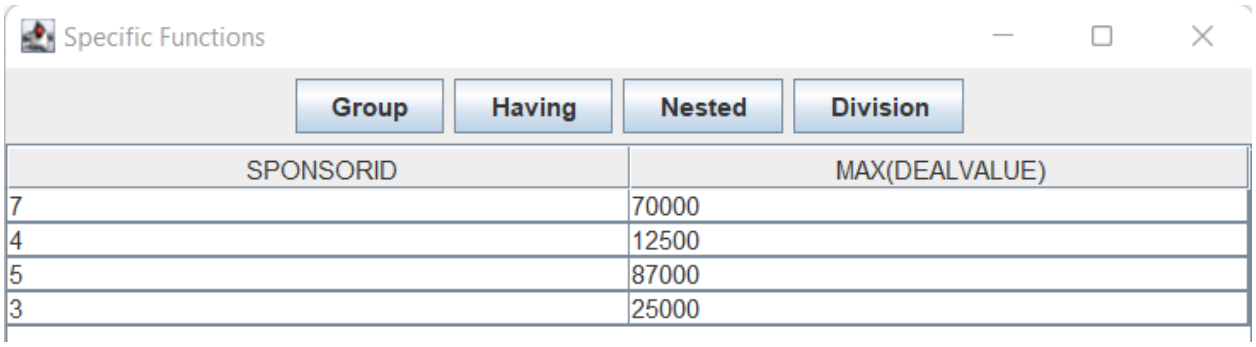
Find the max team sponsorship value for each sponsorID where the sponsorID is greater than or equal to 3.

```
SELECT sponsorID, max(dealvalue)
FROM sponsorsteam
GROUP BY sponsorID
HAVING sponsorID >= 3
```



sponsorID	teamID	dealValue
1	1	40000
2	2	5000
3	3	25000
4	4	12500
5	5	30000
1	2	38000
1	3	15000
1	4	18000
1	5	90000
5	1	87000
5	2	10000
5	3	45000
5	4	25000
7	1	68000
7	2	70000
7	3	15000
7	4	33000
7	5	28000

Showing all team sponsorship values.



SPONSORID	MAX(DEALVALUE)
7	70000
4	12500
5	87000
3	25000

Showing max deal values for sponsors who's sponsorIR is greater than or equal to 3.

<u>Where to find?</u>

The sql query string is located inside **src/database/QueryBuilder** `AGG_WITH_HAVING_QUERY`

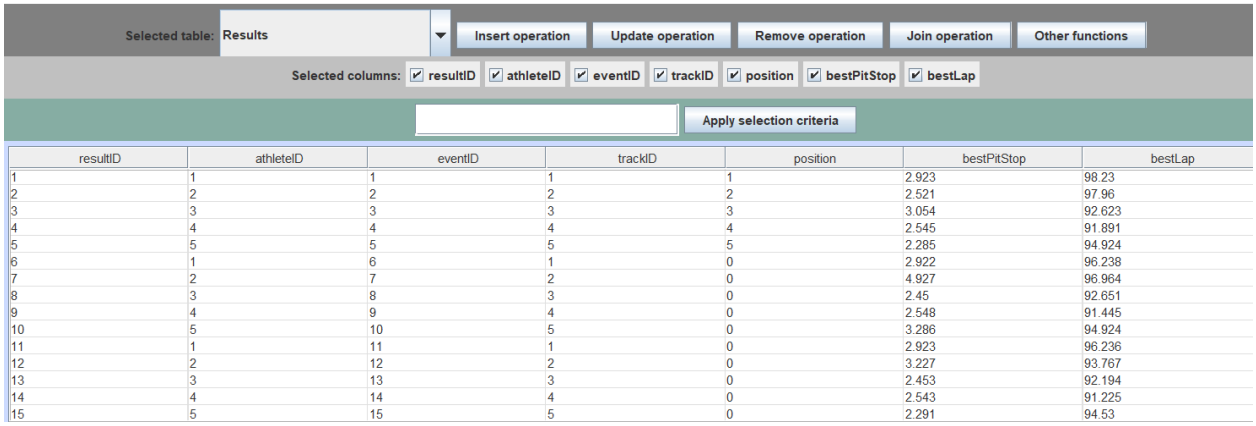
The query is executed by **`aggWithHaving()`** inside **src/database/DatabaseConnection/Handler**

The **`aggWithHaving()`** function is called by **`havingButtonPress()`** inside **sr/ui/FunctionsFrame**

Queries: Nested Aggregation with Group By

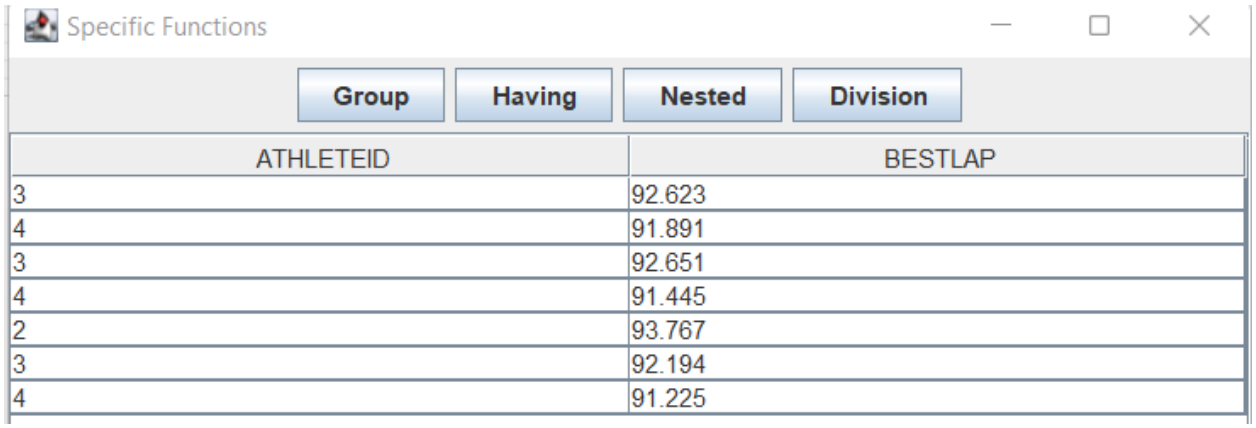
Find the best laps that are faster than the average best lap time.

```
SELECT r1.athleteID, r1.bestLap
FROM results r1
WHERE r1.bestLap < (SELECT AVG(r2.bestLap)
                    FROM results r2)
```



resultID	athleteID	eventID	trackID	position	bestPitStop	bestLap
1	1	1	1	1	2.923	98.23
2	2	2	2	2	2.521	97.96
3	3	3	3	3	3.054	92.623
4	4	4	4	4	2.545	91.891
5	5	5	5	5	2.285	94.924
6	1	6	1	0	2.922	96.238
7	2	7	2	0	4.927	96.964
8	3	8	3	0	2.45	92.651
9	4	9	4	0	2.548	91.445
10	5	10	5	0	3.286	94.924
11	1	11	1	0	2.923	96.236
12	2	12	2	0	3.227	93.767
13	3	13	3	0	2.453	92.194
14	4	14	4	0	2.543	91.225
15	5	15	5	0	2.291	94.53

Showing all results.



ATHLETEID	BESTLAP
3	92.623
4	91.891
3	92.651
4	91.445
2	93.767
3	92.194
4	91.225

Showing the best laps that are faster than the average best lap.

<u>Where to find?</u>

The sql query string is located inside **src/database/QueryBuilder** `NESTED_AGG_QUERY`

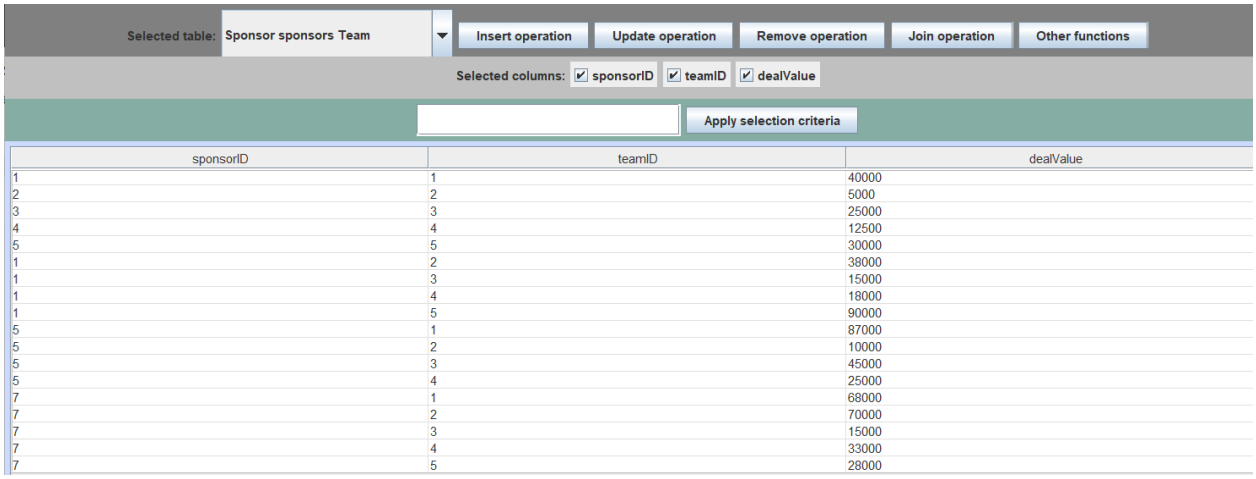
The query is executed by **`nestedAgg()`** inside **src/database/DatabaseConnection/Handler**

The **`nestedAgg()`** function is called by **`nestedButtonPress()`** inside **sr/ui/FunctionsFrame**

Queries: Division

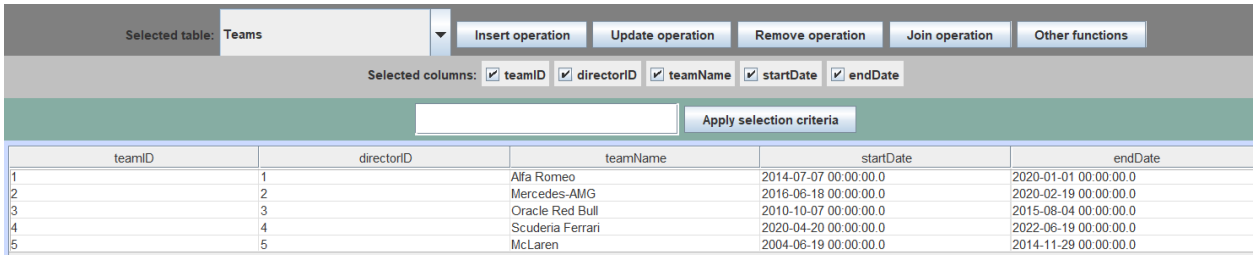
Find all sponsors who have sponsored every team.

```
SELECT *
FROM sponsor s
WHERE NOT EXISTS ((SELECT t.teamID
                   FROM team t)
                   MINUS
                  (SELECT st.teamID
                   FROM sponsorsteam st
                   WHERE st.sponsorID = s.sponsorID))
```



sponsorID	teamID	dealValue
1	1	40000
2	2	5000
3	3	25000
4	4	12500
5	5	30000
1	2	38000
1	3	15000
1	4	18000
1	5	90000
5	1	87000
5	2	10000
5	3	45000
5	4	25000
7	1	68000
7	2	70000
7	3	15000
7	4	33000
7	5	28000

Showing all team sponsorships.



teamID	directorID	teamName	startDate	endDate
1	1	Alfa Romeo	2014-07-07 00:00:00.0	2020-01-01 00:00:00.0
2	2	Mercedes-AMG	2016-06-18 00:00:00.0	2020-02-19 00:00:00.0
3	3	Oracle Red Bull	2010-10-07 00:00:00.0	2015-08-04 00:00:00.0
4	4	Scuderia Ferrari	2020-04-20 00:00:00.0	2022-06-19 00:00:00.0
5	5	McLaren	2004-06-19 00:00:00.0	2014-11-29 00:00:00.0

Showing all teams

Showing all sponsors that have sponsored every team.

Where to find?

The sql query string is located inside **src/database/QueryBuilder** `DIVISION_QUERY`

The query is executed by **`division()`** inside **src/database/DatabaseConnection/Handler**

The **`division()`** function is called by **`divisionButtonPress()`** inside **sr/ui/FunctionsFrame**