

#### Source code for spec/spec\_helper.rb

```
1 require "bundler/setup"
2
3 unless ENV["CODECOV_TOKEN"].nil?
4   require 'simplecov'
5   SimpleCov.start
6
7   require 'codecov'
8   SimpleCov.formatter = SimpleCov::Formatter::Codecov
9 end
10
11 require "sql_assess"
12 require "timecop"
13 require "pry"
14
15 module SharedConnection
16   def connection
17     @shared_connection
18   end
19 end
20
21 RSpec.configure do |config|
22   # Enable flags like --only-failures and --next-failure
23   config.example_status_persistence_file_path = ".rspec_status"
24
25   # Disable RSpec exposing methods globally on `Module` and `main`
26   config.disable_monkey_patching!
27
28   config.expect_with :rspec do |c|
29     c.syntax = :expect
30   end
31
32   config.include SharedConnection
33
34   config.before(:suite) do
35     SqlAssess::DatabaseConnection.new(database: "local_db")
36   end
37
38   config.before(:all) do
39     @shared_connection = SqlAssess::DatabaseConnection.new(database: "local_db")
40   end
41
42   config.before(:each) do
43     @shared_connection.delete_database
44   end
45 end
```

#### Source code for spec/sql\_assess\_spec.rb

```
1 RSpec.describe SqlAssess do
2   it "has a version number" do
3     expect(SqlAssess::VERSION).not_to be nil
4   end
5 end
```

#### Source code for spec/fixtures/transformer\_integration\_tests.yml

```
1 -
2 schema: |
3   CREATE TABLE table1 (id1 integer, id2 integer)
4 query: SELECT * from table1
5 expected_result: SELECT `table1`.`id1`, `table1`.`id2` FROM `table1`
6 -
7 schema: |
```

```

8 CREATE TABLE table1 (name integer, id2 integer)
9 query: SELECT table1.name from table1
10 expected_result: SELECT `table1`.`name` FROM `table1`
11 -
12 schema: |
13 CREATE TABLE table1 (id1 integer, id2 integer)
14 query: SELECT * from table1 ORDER BY id1 DESC
15 expected_result: SELECT `table1`.`id1`, `table1`.`id2` FROM `table1` ORDER BY `table1`.`id1` DESC
16 -
17 schema: |
18 CREATE TABLE table1 (id1 integer, id2 integer);
19 CREATE TABLE table2 (id3 integer, id4 integer)
20 query: SELECT * from table1, table2
21 expected_result: SELECT `table1`.`id1`, `table1`.`id2`, `table2`.`id3`, `table2`.`id4` FROM `table1` CROSS
↪ JOIN `table2`
22 -
23 schema: |
24 CREATE TABLE table1 (id1 integer, id2 integer);
25 CREATE TABLE table2 (id3 integer, id4 integer)
26 query: SELECT * from table1 LEFT JOIN table2 on table1.id1 = table2.id3
27 expected_result: SELECT `table1`.`id1`, `table1`.`id2`, `table1`.`id1`, `table2`.`id4` FROM `table1` LEFT
↪ JOIN `table2` ON `table1`.`id1` = `table2`.`id3`
28 -
29 schema: |
30 CREATE TABLE table1 (id1 integer, id2 integer);
31 CREATE TABLE table2 (id3 integer, id4 integer)
32 query: SELECT * from table1, table2 WHERE id1 > 3
33 expected_result: SELECT `table1`.`id1`, `table1`.`id2`, `table2`.`id3`, `table2`.`id4` FROM `table1` CROSS
↪ JOIN `table2` WHERE 3 < `table1`.`id1`
34 -
35 schema: |
36 CREATE TABLE table1 (id1 integer, id2 integer);
37 CREATE TABLE table2 (id3 integer, id4 integer)
38 query: SELECT * from table1, table2 WHERE id1 BETWEEN 1 and 3
39 expected_result: SELECT `table1`.`id1`, `table1`.`id2`, `table2`.`id3`, `table2`.`id4` FROM `table1` CROSS
↪ JOIN `table2` WHERE (1 <= `table1`.`id1` AND `table1`.`id1` <= 3)
40 -
41 schema: |
42 CREATE TABLE table1 (id1 integer, id2 integer);
43 CREATE TABLE table2 (id3 integer, id4 integer)
44 query: SELECT * from table1, table2 WHERE id1 BETWEEN 1 and 3 AND id2 > 3 ORDER BY 1
45 expected_result: SELECT `table1`.`id1`, `table1`.`id2`, `table2`.`id3`, `table2`.`id4` FROM `table1` CROSS
↪ JOIN `table2` WHERE ((1 <= `table1`.`id1` AND `table1`.`id1` <= 3) AND 3 < `table1`.`id2`) ORDER BY
↪ `table1`.`id1` ASC

```

Source code for spec/fixtures/transformer\_hacker\_rank\_integration\_tests.yml

1 # Basic Select

```

2 -
3 name: Revising the Select Query I
4 schema: |
5 CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪ population integer)
6 query: SELECT * from CITY WHERE `POPULATION` > 100000 and `COUNTRYCODE` = "USA"
7 expected_result: SELECT `CITY`.`id`, `CITY`.`name`, `CITY`.`countrycode`, `CITY`.`district`,
↪ `CITY`.`population` FROM `CITY` WHERE (100000 < `CITY`.`POPULATION` AND `CITY`.`COUNTRYCODE` = 'USA')
8 -
9 name: Revising the Select Query II
10 schema: |
11 CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪ population integer)
12 query: select name from CITY where POPULATION > 120000 and `COUNTRYCODE` = 'USA'
13 expected_result: SELECT `CITY`.`name` FROM `CITY` WHERE (120000 < `CITY`.`POPULATION` AND
↪ `CITY`.`COUNTRYCODE` = 'USA')

```

```

14 -
15     name: Select All
16     schema: |
17         CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
18         ↪ population integer)
19     query: SELECT * from CITY
20     expected_result: SELECT `CITY`.`id`, `CITY`.`name`, `CITY`.`countrycode`, `CITY`.`district`,
21     ↪ `CITY`.`population` FROM `CITY`
22 -
23     name: Select By Id
24     schema: |
25         CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
26         ↪ population integer)
27     query: select * from CITY WHERE ID = 1661
28     expected_result: SELECT `CITY`.`id`, `CITY`.`name`, `CITY`.`countrycode`, `CITY`.`district`,
29     ↪ `CITY`.`population` FROM `CITY` WHERE `CITY`.`ID` = 1661
30 -
31     name: Japanese Cities' Attributes
32     schema: |
33         CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
34         ↪ population integer)
35     query: select * from CITY WHERE `COUNTRYCODE` = 'JPN'
36     expected_result: SELECT `CITY`.`id`, `CITY`.`name`, `CITY`.`countrycode`, `CITY`.`district`,
37     ↪ `CITY`.`population` FROM `CITY` WHERE `CITY`.`COUNTRYCODE` = 'JPN'
38 -
39     name: Japanese Cities' Names
40     schema: |
41         CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
42         ↪ population integer)
43     query: select name from CITY WHERE `COUNTRYCODE` = 'JPN'
44     expected_result: SELECT `CITY`.`name` FROM `CITY` WHERE `CITY`.`COUNTRYCODE` = 'JPN'
45 -
46     name: Weather Observation Station 1
47     schema: |
48         CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
49     query: SELECT city, state from STATION
50     expected_result: SELECT `STATION`.`city`, `STATION`.`state` FROM `STATION`
51 -
52     name: Weather Observation Station 3
53     schema: |
54         CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
55     query: select DISTINCT(CITY) from STATION where ID % 2 = 0 ORDER by CITY DESC
56     expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE (`STATION`.`ID` % 2) = 0 ORDER BY
57     ↪ `STATION`.`CITY` DESC
58 -
59     name: Weather Observation Station 4
60     schema: |
61         CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
62     query: select count(CITY) - count(DISTINCT CITY) FROM STATION;
63     support: false
64 -
65     name: Weather Observation Station 5 - part 1
66     schema: |
67         CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
68     query: select city, length(city) from STATION order by length(city) ASC, city ASC LIMIT 2
69     support: false
70 -
71     name: Weather Observation Station 5 - part 2
72     schema: |
73         CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
74     query: select city, length(city) from STATION order by length(city) desc, city ASC limit 1
75     support: false
76 -

```

```

69 name: Weather Observation Station 6
70 schema: |
71 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
72 query: select DISTINCT CITY from STATION WHERE CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY
↵ LIKE 'O%' OR CITY LIKE 'U%'
73 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE ((((`STATION`.`CITY` LIKE 'A%' OR
↵ `STATION`.`CITY` LIKE 'E%') OR `STATION`.`CITY` LIKE 'I%') OR `STATION`.`CITY` LIKE 'O%') OR
↵ `STATION`.`CITY` LIKE 'U%')
74 -
75 name: Weather Observation Station 7
76 schema: |
77 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
78 query: select DISTINCT CITY from STATION WHERE CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY
↵ LIKE 'O%' OR CITY LIKE 'U%'
79 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE ((((`STATION`.`CITY` LIKE 'A%' OR
↵ `STATION`.`CITY` LIKE 'E%') OR `STATION`.`CITY` LIKE 'I%') OR `STATION`.`CITY` LIKE 'O%') OR
↵ `STATION`.`CITY` LIKE 'U%')
80 -
81 name: Weather Observation Station 8
82 schema: |
83 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
84 query: select DISTINCT CITY from STATION WHERE CITY LIKE '[AEIOU][AEIOU]'
85 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE `STATION`.`CITY` LIKE
↵ '[AEIOU][AEIOU]'
86 -
87 name: Weather Observation Station 9
88 schema: |
89 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
90 query: select DISTINCT CITY from STATION WHERE CITY NOT LIKE '[AEIOUaeiou]%'
91 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE `CITY` NOT LIKE '[AEIOUaeiou]%'
92 -
93 name: Weather Observation Station 10
94 schema: |
95 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
96 query: select DISTINCT CITY from STATION WHERE CITY LIKE '%[^AEIOU]'
97 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE `STATION`.`CITY` LIKE '%[^AEIOU]'
98 -
99 name: Weather Observation Station 11
100 schema: |
101 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
102 query: select DISTINCT CITY from STATION WHERE CITY LIKE '[^AEIOU]%' OR CITY LIKE '%[^AEIOU]'
103 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE (`STATION`.`CITY` LIKE '[^AEIOU]%' OR
↵ `STATION`.`CITY` LIKE '%[^AEIOU]')
104 -
105 name: Weather Observation Station 12
106 schema: |
107 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
108 query: select DISTINCT CITY from STATION WHERE CITY LIKE '[^AEIOU][^AEIOU]'
109 expected_result: SELECT DISTINCT `STATION`.`CITY` FROM `STATION` WHERE `STATION`.`CITY` LIKE
↵ '[^AEIOU][^AEIOU]'
110 -
111 name: Higher Than 75 Marks
112 schema: |
113 CREATE TABLE STUDENTS(id integer, name varchar(255), marks integer)
114 query: select Name from STUDENTS where Marks > 75 order by substr(Name, -3) ASC, ID ASC
115 support: false
116 -
117 name: Employee names
118 schema: |
119 CREATE TABLE EMPLOYEE(employee_id integer, name varchar(255), months integer, salary integer)
120 query: SELECT name FROM `EMPLOYEE` ORDER BY name ASC
121 expected_result: SELECT `EMPLOYEE`.`name` FROM `EMPLOYEE` ORDER BY `EMPLOYEE`.`name` ASC
122 -

```

```

23 name: Employee salaries
24 schema: |
25     CREATE TABLE EMPLOYEE(employee_id integer, name varchar(255), months integer, salary integer)
26 query: SELECT name FROM `EMPLOYEE` WHERE salary > 2000 and months < 10
27 expected_result: SELECT `EMPLOYEE`.`name` FROM `EMPLOYEE` WHERE (2000 < `EMPLOYEE`.`salary` AND
↳ `EMPLOYEE`.`months` < 10)
28 # Basic join
29 -
30 name: Asian Population
31 schema: |
32     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↳ population integer);
33     CREATE TABLE COUNTRY(
34         code varchar(255), name varchar(255), continent varchar(255),
35         region varchar(255), surfacearea integer, indepyear varchar(255),
36         population integer, lifeexpectancy varchar(255), gnp integer,
37         gnpold varchar(255), localname varchar(255), governmentform varchar(255),
38         headofstate varchar(255), capital varchar(255), code2 varchar(255)
39     );
40 query: SELECT SUM(CITY.POPULATION) FROM CITY LEFT JOIN `COUNTRY` ON `COUNTRY`.CODE = CITY.`COUNTRYCODE`
↳ WHERE `COUNTRY`.CONTINENT = "ASIA"
41 expected_result: SELECT SUM(`CITY`.`POPULATION`) FROM `CITY` LEFT JOIN `COUNTRY` ON `COUNTRY`.`CODE` =
↳ `CITY`.`COUNTRYCODE` WHERE `COUNTRY`.`CONTINENT` = 'ASIA'
42 -
43 name: African cities
44 schema: |
45     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↳ population integer);
46     CREATE TABLE COUNTRY(
47         code varchar(255), name varchar(255), continent varchar(255),
48         region varchar(255), surfacearea integer, indepyear varchar(255),
49         population integer, lifeexpectancy varchar(255), gnp integer,
50         gnpold varchar(255), localname varchar(255), governmentform varchar(255),
51         headofstate varchar(255), capital varchar(255), code2 varchar(255)
52     );
53 query: SELECT CITY.NAME FROM CITY LEFT JOIN `COUNTRY` ON `COUNTRY`.CODE = CITY.`COUNTRYCODE` WHERE
↳ `COUNTRY`.CONTINENT = 'AFRICA'
54 expected_result: SELECT `CITY`.`NAME` FROM `CITY` LEFT JOIN `COUNTRY` ON `COUNTRY`.`CODE` =
↳ `CITY`.`COUNTRYCODE` WHERE `COUNTRY`.`CONTINENT` = 'AFRICA'
55 -
56 name: Average Population of Each Continent
57 schema: |
58     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↳ population integer);
59     CREATE TABLE COUNTRY(
60         code varchar(255), name varchar(255), continent varchar(255),
61         region varchar(255), surfacearea integer, indepyear varchar(255),
62         population integer, lifeexpectancy varchar(255), gnp integer,
63         gnpold varchar(255), localname varchar(255), governmentform varchar(255),
64         headofstate varchar(255), capital varchar(255), code2 varchar(255)
65     );
66 query: SELECT `COUNTRY`.CONTINENT, AVG(CITY.POPULATION) FROM CITY LEFT JOIN `COUNTRY` ON `COUNTRY`.CODE =
↳ CITY.`COUNTRYCODE` GROUP BY `COUNTRY`.CONTINENT
67 expected_result: SELECT `COUNTRY`.`CONTINENT`, AVG(`CITY`.`POPULATION`) FROM `CITY` LEFT JOIN `COUNTRY` ON
↳ `COUNTRY`.`CODE` = `CITY`.`COUNTRYCODE` GROUP BY `COUNTRY`.CONTINENT
68 -
69 name: The report
70 schema: |
71     CREATE TABLE Students(id integer, name varchar(255), marks integer);
72     CREATE TABLE Grades(grade integer, min_mark integer, max_mark integer);
73 query: SELECT CASE WHEN `Grades`.`grade` < 8 THEN NULL ELSE Students.Name END, Grades.grade, Students.Marks
↳ FROM Students LEFT JOIN Grades ON Students.Marks >= Grades.`Min_Mark` AND Students.Marks <=
↳ Grades.`Max_Mark` ORDER BY Grades.grade DESC, Students.Name ASC

```

```

74 support: false
75 -
76 name: Top competitors
77 schema: |
78     CREATE TABLE hackers(hacker_id integer, name varchar(255));
79     CREATE TABLE difficulty(difficulty_level integer, score integer);
80     CREATE TABLE challenges(difficulty_level integer, hacker_id integer, challenge_id integer);
81     CREATE TABLE submissions(submission_id integer, hacker_id integer, challenge_id integer, score integer);
82 query: |
83     select hackers.hacker_id, hackers.name
84     from
85         submissions
86         inner join challenges on submissions.challenge_id = challenges.challenge_id
87         inner join difficulty on challenges.difficulty_level = difficulty.difficulty_level
88         inner join hackers on submissions.hacker_id = hackers.hacker_id
89     where submissions.score = difficulty.score and challenges.difficulty_level = difficulty.difficulty_level
90     group by hackers.hacker_id, hackers.name
91     having count(submissions.hacker_id) > 1
92     order by count(submissions.hacker_id) desc, submissions.hacker_id asc
93 expected_result: |
94     SELECT `hackers`.`hacker_id`, `hackers`.`name`
95     FROM
96         `submissions`
97         INNER JOIN `challenges` ON `submissions`.`challenge_id` = `challenges`.`challenge_id`
98         INNER JOIN `difficulty` ON `challenges`.`difficulty_level` = `difficulty`.`difficulty_level`
99         INNER JOIN `hackers` ON `submissions`.`hacker_id` = `hackers`.`hacker_id`
100     WHERE (`submissions`.`score` = `difficulty`.`score` AND `challenges`.`difficulty_level` =
↪ `challenges`.`difficulty_level`)
101     GROUP BY `hackers`.`hacker_id`, `hackers`.`name`
102     HAVING 1 < COUNT(`hackers`.`hacker_id`)
103     ORDER BY COUNT(`hackers`.`hacker_id`) DESC, `hackers`.`hacker_id` ASC
104 -
105 name: Challenges
106 schema: |
107     CREATE TABLE hackers(hacker_id integer, name varchar(255));
108     CREATE TABLE difficulty(difficulty_level integer, score integer);
109     CREATE TABLE challenges(difficulty_level integer, hacker_id integer, challenge_id integer);
110     CREATE TABLE submissions(submission_id integer, hacker_id integer, challenge_id integer, score integer);
111 query: |
112     select * from hackers
113 support: false
114 -
115 name: Contest leaderboard
116 schema: |
117     CREATE TABLE hackers(hacker_id integer, name varchar(255));
118     CREATE TABLE difficulty(difficulty_level integer, score integer);
119     CREATE TABLE challenges(difficulty_level integer, hacker_id integer, challenge_id integer);
120     CREATE TABLE submissions(submission_id integer, hacker_id integer, challenge_id integer, score integer);
121 query: |
122     select * from hackers
123 support: false
124 # Advance select
125 -
126 name: Type of triangle
127 support: false
128 -
129 name: The pads
130 support: false
131 -
132 name: Occupations
133 support: false
134 -
135 name: Binary Tree nodes

```



```

36 support: false
37 -
38 name: New companies
39 schema: |
40 CREATE TABLE Company(company_code varchar(255), founder varchar(255));
41 CREATE TABLE Lead_Manager(company_code varchar(255), lead_manager_code varchar(255));
42 CREATE TABLE Senior_Manager(company_code varchar(255), lead_manager_code varchar(255), senior_manager_code
↪ varchar(255));
43 CREATE TABLE Manager(company_code varchar(255), lead_manager_code varchar(255), senior_manager_code
↪ varchar(255), manager_code varchar(255));
44 CREATE TABLE Employee(company_code varchar(255), lead_manager_code varchar(255), senior_manager_code
↪ varchar(255), manager_code varchar(255), employee_code varchar(255));
45 query: |
46 select Company.company_code, Company.founder,
47 count(Lead_Manager.lead_manager_code), count(Senior_Manager.senior_manager_code),
48 count(Manager.manager_code), count(`Employee`.`employee_code`)
49 from Company, Lead_Manager, Senior_Manager, Manager, `Employee`
50 where Company.company_code = Lead_Manager.company_code
51 and Lead_Manager.lead_manager_code = Senior_Manager.lead_manager_code
52 and Senior_Manager.senior_manager_code = Manager.senior_manager_code
53 and Manager.manager_code = `Employee`.manager_code
54 group by Company.company_code, Company.founder
55 order by Company.company_code
56 expected_result: |
57 SELECT
58 `Company`.`company_code`,
59 `Company`.`founder`,
60 COUNT(`Lead_Manager`.`lead_manager_code`),
61 COUNT(`Senior_Manager`.`senior_manager_code`),
62 COUNT(`Manager`.`manager_code`),
63 COUNT(`Employee`.`employee_code`)
64 FROM
65 `Company`
66 CROSS JOIN `Lead_Manager`
67 CROSS JOIN `Senior_Manager`
68 CROSS JOIN `Manager`
69 CROSS JOIN `Employee`
70 WHERE
71 (((`Company`.`company_code` = `Lead_Manager`.`company_code` AND `Lead_Manager`.`lead_manager_code` =
↪ `Senior_Manager`.`lead_manager_code`) AND `Senior_Manager`.`senior_manager_code` =
↪ `Manager`.`senior_manager_code`) AND `Manager`.`manager_code` = `Employee`.`manager_code`)
72 GROUP BY
73 `Company`.`company_code`,
74 `Company`.`founder`
75 ORDER BY `Company`.`company_code` ASC
76 # Aggregate
77 -
78 name: Revising Aggregations - The Count Function
79 schema: |
80 CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪ population integer)
81 query: SELECT COUNT(id) FROM CITY WHERE population > 100000
82 expected_result: SELECT COUNT(`CITY`.`id`) FROM `CITY` WHERE 100000 < `CITY`.`population`
83 -
84 name: Revising Aggregations - The Sum Function
85 schema: |
86 CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪ population integer)
87 query: SELECT SUM(POPULATION) FROM CITY WHERE DISTRICT="CALIFORNIA" GROUP BY DISTRICT
88 expected_result: SELECT SUM(`CITY`.`POPULATION`) FROM `CITY` WHERE `CITY`.`DISTRICT` = 'CALIFORNIA' GROUP BY
↪ `CITY`.`DISTRICT`
89 -
90 name: Revising Aggregations - Averages

```

```

91 schema: |
92     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪     population integer)
93 query: SELECT AVG(population) FROM CITY WHERE district = 'California'
94 expected_result: SELECT AVG(`CITY`.`population`) FROM `CITY` WHERE `CITY`.`district` = 'California'
95 -
96 name: Average Population
97 schema: |
98     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪     population integer)
99 query: SELECT AVG(POPULATION) FROM CITY
100 expected_result: SELECT AVG(`CITY`.`POPULATION`) FROM `CITY`
101 -
102 name: Japan Population
103 schema: |
104     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪     population integer)
105 query: SELECT SUM(POPULATION) FROM CITY WHERE `COUNTRYCODE` = 'JPN'
106 expected_result: SELECT SUM(`CITY`.`POPULATION`) FROM `CITY` WHERE `CITY`.`COUNTRYCODE` = 'JPN'
107 -
108 name: Population Density Difference
109 schema: |
110     CREATE TABLE CITY(id integer, name varchar(255), countrycode varchar(255), district varchar(255),
↪     population integer)
111 query: SELECT MAX(Population) - MIN(Population) FROM CITY
112 expected_result: SELECT (MAX(`CITY`.`Population`) - MIN(`CITY`.`Population`)) FROM `CITY`
113 -
114 name: The Blunder
115 schema: |
116     CREATE TABLE employees(id integer, name varchar(255), salary integer)
117 query: SELECT AVG(salary - REPLACE(salary, '0', '')) FROM employees;
118 support: false
119 -
120 name: Top earners
121 schema: |
122     CREATE TABLE EMPLOYEE(employee_id integer, name varchar(255), months integer, salary integer)
123 query: select salary * months FROM `EMPLOYEE` group by 1 order by 1 desc
124 expected_result: SELECT (`EMPLOYEE`.`salary` * `EMPLOYEE`.`months`) FROM `EMPLOYEE` GROUP BY
↪ (`EMPLOYEE`.`salary` * `EMPLOYEE`.`months`) ORDER BY (`EMPLOYEE`.`salary` * `EMPLOYEE`.`months`) DESC
125 -
126 name: Weather Observation Station 2
127 schema: |
128     CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
129 query: SELECT ROUND(SUM(LAT_N), 2), ROUND(SUM(LONG_W), 2) FROM STATION;
130 support: false
131 -
132 name: Weather Observation Station 13
133 schema: |
134     CREATE TABLE station(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
135 query: select sum(lat_n) from station where lat_n>38.7880 and lat_n<137.2345
136 expected_result: SELECT SUM(`station`.`lat_n`) FROM `station` WHERE (38.788 < `station`.`lat_n` AND
↪ `station`.`lat_n` < 137.2345)
137 -
138 name: Weather Observation Station 14
139 schema: |
140     CREATE TABLE station(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
141 support: false
142 -
143 name: Weather Observation Station 16
144 schema: |
145     CREATE TABLE station(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
146 support: false
147 -

```



```

48 name: Weather Observation Station 17
49 schema: |
50 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
51 query: select LONG_W from STATION where LAT_N>38.7780 order by LAT_N
52 expected_result: SELECT `STATION`.`LONG_W` FROM `STATION` WHERE 38.778 < `STATION`.`LAT_N` ORDER BY
↵ `STATION`.`LAT_N` ASC
53 -
54 name: Weather Observation Station 18
55 schema: |
56 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
57 support: false
58 -
59 name: Weather Observation Station 19
60 schema: |
61 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
62 support: false
63 -
64 name: Weather Observation Station 20
65 schema: |
66 CREATE TABLE STATION(id integer, CITY varchar(255), STATE varchar(255), LAT_N DOUBLE, LONG_W DOUBLE)
67 support: false

```

Source code for spec/fixtures/assessor\_integration\_tests.yml

```

1 -
2 schema: CREATE TABLE t1(id integer);
3 seed: INSERT INTO t1(id) VALUES (122);
4 instructor_query: SELECT * from t1;
5 student_query: SELECT 2 from t1;
6 message: Your query is not correct. Check what columns you are selecting.

```

Source code for spec/sql\_assess/runner\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Runner do
4   subject { described_class.new(connection) }
5
6   describe "#create_schema" do
7     context "with a correct command" do
8       context "with a single command" do
9         it "runs the command" do
10           subject.create_schema('CREATE TABLE table1 (id integer);')
11
12           tables = connection.query("SHOW tables");
13           expect(tables.first["Tables_in_local_db"]).to eq("table1")
14         end
15       end
16
17       context "with multiple commands" do
18         it "runs all commands" do
19           subject.create_schema('CREATE TABLE table1 (id integer); CREATE TABLE table2 (id integer);')
20
21           tables = connection.query("SHOW tables");
22           expect(tables.size).to eq(2)
23           expect(tables.map{ |line| line["Tables_in_local_db"] }).to eq(["table1", "table2"])
24         end
25       end
26     end
27
28     context "with an incorrect command" do
29       it "raises an exception" do
30         expect do
31           subject.create_schema('CREATE TABLES table1 (id integer);')
32         end.to raise_error(

```

```

33         SqlAssess::DatabaseSchemaError,
34         /near .+ at line 1/
35     )
36 end
37 end
38 end
39
40 describe "#seed_initial_data" do
41     before do
42         subject.create_schema('CREATE TABLE table1 (id integer);')
43     end
44
45     context "with a correct command" do
46         context "with a single command" do
47             it "runs the command" do
48                 subject.seed_initial_data('INSERT INTO table1 (id) values(1);')
49
50                 rows = connection.query('SELECT * FROM table1');
51                 expect(rows.count).to eq(1)
52                 expect(rows.first["id"]).to eq(1)
53             end
54         end
55
56         context "with multiple commands" do
57             it "runs all commands" do
58                 subject.seed_initial_data('INSERT INTO table1 (id) values(1); INSERT INTO table1 (id) values(2);')
59
60                 rows = connection.query('SELECT * FROM table1');
61                 expect(rows.size).to eq(2)
62                 expect(rows.map{ |line| line["id"] }).to eq([1, 2])
63             end
64         end
65     end
66
67     context "with an incorrect command" do
68         it "raises an exception" do
69             expect do
70                 subject.seed_initial_data('INSERT INTO table1 (id2) values("ab");')
71             end.to raise_error(
72                 SqlAssess::DatabaseSeedError,
73                 "Unknown column 'id2' in 'field list'"
74             )
75         end
76     end
77 end
78
79 context "#execute_query" do
80     before do
81         connection.query('CREATE TABLE table1 (id integer);')
82         connection.query('INSERT INTO table1 (id) values(1);')
83         connection.query('INSERT INTO table1 (id) values(2);')
84     end
85
86     context "with a wrong query" do
87         let(:query) { "SELECT id2 from table1;" }
88
89         it "raises an exception" do
90             expect { subject.execute_query(query) }.to raise_error(
91                 SqlAssess::DatabaseQueryExecutionFailed
92             )
93         end
94     end
95 end

```

```

96 context "with a correct query" do
97   let(:query) { "SELECT id2 from table1;" }
98
99   it "raises an exception" do
100     expect { subject.execute_query(query) }.to raise_error(
101       SqlAssess::DatabaseQueryExecutionFailed,
102       "Unknown column 'id2' in 'field list'"
103     )
104   end
105 end
106 end
107 end

```

Source code for spec/sql\_assess/grader/order\_by\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::OrderBy do
4   subject do
5     described_class.new(
6       student_attributes: attributes[:student][:order_by],
7       instructor_attributes: attributes[:instructor][:order_by]
8     )
9   end
10
11   before do
12     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
13     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
14   end
15
16   let(:attributes) do
17     SqlAssess::QueryAttributeExtractor.new.extract(
18       instructor_query, student_query
19     )
20   end
21
22   context "with no order by" do
23     let(:student_query) do
24       <<-SQL
25       SELECT a from table1
26       SQL
27     end
28
29     let(:instructor_query) do
30       <<-SQL
31       SELECT a from table1
32       SQL
33     end
34
35     it { expect(subject.rounded_grade).to eq(1) }
36   end
37
38   context "with one equal order by" do
39     let(:student_query) do
40       <<-SQL
41       SELECT a from table1
42       ORDER BY a ASC
43       SQL
44     end
45
46     let(:instructor_query) do
47       <<-SQL
48       SELECT a from table1
49       ORDER BY a ASC

```

```

50     SQL
51 end
52
53 it { expect(subject.rounded_grade).to eq(1) }
54 end
55
56 context "with two equal order by" do
57   let(:student_query) do
58     <<-SQL
59       SELECT a from table1
60       ORDER BY a, b
61     SQL
62   end
63
64   let(:instructor_query) do
65     <<-SQL
66       SELECT a from table1
67       ORDER BY a, b
68     SQL
69   end
70
71   it { expect(subject.rounded_grade).to eq(1) }
72 end
73
74 context "with one equal and one different order by" do
75   let(:student_query) do
76     <<-SQL
77       SELECT a from table1
78       ORDER BY a, b
79     SQL
80   end
81
82   let(:instructor_query) do
83     <<-SQL
84       SELECT a from table1
85       ORDER BY a, c
86     SQL
87   end
88
89   it { expect(subject.rounded_grade).to eq(0.5) }
90 end
91
92 context "with one equal and one different order by" do
93   let(:student_query) do
94     <<-SQL
95       SELECT a from table1
96       ORDER BY a, b
97     SQL
98   end
99
100   let(:instructor_query) do
101     <<-SQL
102       SELECT a from table1
103       ORDER BY a
104     SQL
105   end
106
107   it { expect(subject.rounded_grade).to eq(0.67) }
108 end
109
110 context "with reversed two order by" do
111   let(:student_query) do
112     <<-SQL

```

```

13     SELECT a from table1
14     ORDER BY a, b
15     SQL
16 end
17
18 let(:instructor_query) do
19   <<-SQL
20     SELECT a from table1
21     ORDER BY b, a
22     SQL
23 end
24
25 it { expect(subject.rounded_grade).to eq(0.25) }
26 end
27
28 context "with reversed two order by" do
29   let(:student_query) do
30     <<-SQL
31       SELECT a from table1
32       ORDER BY a ASC, b
33       SQL
34   end
35
36   let(:instructor_query) do
37     <<-SQL
38       SELECT a from table1
39       ORDER BY b, a DESC
40       SQL
41   end
42
43   it { expect(subject.rounded_grade).to eq(0.19) }
44 end
45 end

```

Source code for spec/sql\_assess/grader/where\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::Where do
4   subject do
5     described_class.new(
6       student_attributes: attributes[:student][:where_tree],
7       instructor_attributes: attributes[:instructor][:where_tree]
8     )
9   end
10
11   let(:attributes) do
12     SqlAssess::QueryAttributeExtractor.new.extract(
13       instructor_query, student_query
14     )
15   end
16
17   context "with no where statements" do
18     let(:student_query) do
19       <<-SQL
20         SELECT a from table1
21         SQL
22     end
23
24     let(:instructor_query) do
25       <<-SQL
26         SELECT a from table2
27         SQL
28     end

```

```

29     it { expect(subject.rounded_grade).to eq(1) }
30   end
31
32
33
34   context "with no where for student, but where for teacher" do
35     let(:student_query) do
36       <<-SQL
37         SELECT a from table1
38       SQL
39     end
40
41     let(:instructor_query) do
42       <<-SQL
43         SELECT a from table2
44         WHERE a > 1
45       SQL
46     end
47
48     it { expect(subject.rounded_grade).to eq(0) }
49   end
50
51
52   context "with where for student, but no where for teacher" do
53     let(:student_query) do
54       <<-SQL
55         SELECT a from table1
56         WHERE a > 1
57       SQL
58     end
59
60     let(:instructor_query) do
61       <<-SQL
62         SELECT a from table2
63       SQL
64     end
65
66     it { expect(subject.rounded_grade).to eq(0) }
67   end
68
69   context "with equal where" do
70     let(:student_query) do
71       <<-SQL
72         SELECT a from table1
73         WHERE a > 1
74       SQL
75     end
76
77     let(:instructor_query) do
78       <<-SQL
79         SELECT a from table2
80         WHERE a > 1
81       SQL
82     end
83
84     it { expect(subject.rounded_grade).to eq(1) }
85   end
86
87   context "with different where" do
88     let(:student_query) do
89       <<-SQL
90         SELECT a from table1
91         WHERE a > 2

```



```

92     SQL
93 end
94
95 let(:instructor_query) do
96   <<-SQL
97     SELECT a from table2
98     WHERE a > 1
99   SQL
100 end
101
102 it { expect(subject.rounded_grade).to eq(0) }
103 end
104
105 context "with different but one matching clause" do
106   let(:student_query) do
107     <<-SQL
108       SELECT a from table1
109       WHERE a > 2 AND a > 1
110     SQL
111   end
112
113   let(:instructor_query) do
114     <<-SQL
115       SELECT a from table2
116       WHERE a > 1
117     SQL
118   end
119
120   it { expect(subject.rounded_grade).to eq(0.33) }
121 end
122
123 context "with matching clauses but different boolean operator" do
124   let(:student_query) do
125     <<-SQL
126       SELECT a from table1
127       WHERE a > 2 AND a > 1
128     SQL
129   end
130
131   let(:instructor_query) do
132     <<-SQL
133       SELECT a from table2
134       WHERE a > 2 OR a > 1
135     SQL
136   end
137
138   it { expect(subject.rounded_grade).to eq(0.5) }
139 end
140
141 context "with not matching clauses and different boolean operator" do
142   let(:student_query) do
143     <<-SQL
144       SELECT a from table1
145       WHERE a > 2 AND a > 1 OR a > 3
146     SQL
147   end
148
149   let(:instructor_query) do
150     <<-SQL
151       SELECT a from table2
152       WHERE a > 2 OR a > 3
153     SQL
154   end

```

```

55     it { expect(subject.rounded_grade).to eq(0.73) }
56   end
57 end
58

```

Source code for spec/sql\_assess/grader/having\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Grader::Having do
4    subject do
5      described_class.new(
6        student_attributes: attributes[:student][:having_tree],
7        instructor_attributes: attributes[:instructor][:having_tree]
8      )
9    end
10
11    let(:attributes) do
12      SqlAssess::QueryAttributeExtractor.new.extract(
13        instructor_query, student_query
14      )
15    end
16
17    context "with no having statements" do
18      let(:student_query) do
19        <<-SQL
20          SELECT a from table1
21        SQL
22      end
23
24      let(:instructor_query) do
25        <<-SQL
26          SELECT a from table2
27        SQL
28      end
29
30      it { expect(subject.rounded_grade).to eq(1) }
31    end
32
33
34    context "with no having for student, but having for teacher" do
35      let(:student_query) do
36        <<-SQL
37          SELECT a from table1
38        SQL
39      end
40
41      let(:instructor_query) do
42        <<-SQL
43          SELECT a from table2
44          HAVING a > 1
45        SQL
46      end
47
48      it { expect(subject.rounded_grade).to eq(0) }
49    end
50
51
52    context "with having for student, but no having for teacher" do
53      let(:student_query) do
54        <<-SQL
55          SELECT a from table1
56          HAVING a > 1
57        SQL

```

```

58   end
59
60   let(:instructor_query) do
61     <<-SQL
62       SELECT a from table2
63     SQL
64   end
65
66   it { expect(subject.rounded_grade).to eq(0) }
67 end
68
69 context "with equal having" do
70   let(:student_query) do
71     <<-SQL
72       SELECT a from table1
73       HAVING a > 1
74     SQL
75   end
76
77   let(:instructor_query) do
78     <<-SQL
79       SELECT a from table2
80       HAVING a > 1
81     SQL
82   end
83
84   it { expect(subject.rounded_grade).to eq(1) }
85 end
86
87 context "with different having" do
88   let(:student_query) do
89     <<-SQL
90       SELECT a from table1
91       HAVING a > 2
92     SQL
93   end
94
95   let(:instructor_query) do
96     <<-SQL
97       SELECT a from table2
98       HAVING a > 1
99     SQL
100   end
101
102   it { expect(subject.rounded_grade).to eq(0) }
103 end
104
105 context "with different but one matching clause" do
106   let(:student_query) do
107     <<-SQL
108       SELECT a from table1
109       HAVING a > 2 AND a > 1
110     SQL
111   end
112
113   let(:instructor_query) do
114     <<-SQL
115       SELECT a from table2
116       HAVING a > 1
117     SQL
118   end
119
120   it { expect(subject.rounded_grade).to eq(0.33) }

```

```

21 end
22
23 context "with matching clauses but different boolean operator" do
24   let(:student_query) do
25     <<-SQL
26       SELECT a from table1
27       HAVING a > 2 AND a > 1
28     SQL
29   end
30
31   let(:instructor_query) do
32     <<-SQL
33       SELECT a from table2
34       having a > 2 OR a > 1
35     SQL
36   end
37
38   it { expect(subject.rounded_grade).to eq(0.5) }
39 end
40
41 context "with not matching clauses and different boolean operator" do
42   let(:student_query) do
43     <<-SQL
44       SELECT a from table1
45       HAVING a > 2 AND a > 1 OR a > 3
46     SQL
47   end
48
49   let(:instructor_query) do
50     <<-SQL
51       SELECT a from table2
52       HAVING a > 2 OR a > 3
53     SQL
54   end
55
56   it { expect(subject.rounded_grade).to eq(0.73) }
57 end
58 end

```

Source code for spec/sql\_assess/grader/limit\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::Limit do
4   subject do
5     described_class.new(
6       student_attributes: student_limit,
7       instructor_attributes: instructor_limit
8     )
9   end
10
11   context "same limit and offsert" do
12     let(:student_limit) { { "limit": 1, "offset": 0 } }
13     let(:instructor_limit) { { "limit": 1, "offset": 0 } }
14
15     it { expect(subject.rounded_grade).to eq(1) }
16   end
17
18   context "same limit but different offset" do
19     let(:student_limit) { { "limit": 1, "offset": 1 } }
20     let(:instructor_limit) { { "limit": 1, "offset": 0 } }
21
22     it { expect(subject.rounded_grade).to eq(0.5) }
23   end
24 end

```

```

24 context "different limit but same offset" do
25   let(:student_limit) { { "limit": 2, "offset": 0 } }
26   let(:instructor_limit) { { "limit": 1, "offset": 0 } }
27
28   it { expect(subject.rounded_grade).to eq(0.5) }
29 end
30
31 context "different limit and offset" do
32   let(:student_limit) { { "limit": 2, "offset": 0 } }
33   let(:instructor_limit) { { "limit": 1, "offset": 2 } }
34
35   it { expect(subject.rounded_grade).to eq(0) }
36 end
37 end
38

```

Source code for spec/sql\_assess/grader/group\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::Group do
4   subject do
5     described_class.new(
6       student_attributes: student_group,
7       instructor_attributes: instructor_group
8     )
9   end
10
11   context "example 1 - same columns" do
12     let(:student_group) { ["table1.column"] }
13     let(:instructor_group) { ["table1.column"] }
14
15     it { expect(subject.rounded_grade).to eq(1) }
16   end
17
18   context "example 2 - two of same correct column for student" do
19     let(:student_group) { ["table1.column", "table1.column"] }
20     let(:instructor_group) { ["table1.column"] }
21
22     it { expect(subject.rounded_grade).to eq(0.67) }
23   end
24
25   context "example 3 - one correct column and one incorrect for student" do
26     let(:student_group) { ["table1.column", "table1.column2"] }
27     let(:instructor_group) { ["table1.column"] }
28
29     it { expect(subject.rounded_grade).to eq(0.67) }
30   end
31
32   context "example 4 - slightly different columns" do
33     let(:student_group) { ["table1.column"] }
34     let(:instructor_group) { ["table1.column_2"] }
35
36     it { expect(subject.rounded_grade).to eq(0.17) }
37   end
38
39   context "example 5 - totally different columns" do
40     let(:student_group) { ["table1.column"] }
41     let(:instructor_group) { ["table2.column_2"] }
42
43     it { expect(subject.rounded_grade).to eq(0) }
44   end
45 end

```

Source code for spec/sql\_assess/grader/columns\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::Columns do
4   subject do
5     described_class.new(
6       student_attributes: student_columns,
7       instructor_attributes: instructor_columns
8     )
9   end
10
11   context "example 1 - same columns" do
12     let(:student_columns) { ["table1.column"] }
13     let(:instructor_columns) { ["table1.column"] }
14
15     it { expect(subject.rounded_grade).to eq(1) }
16   end
17
18   context "example 2 - two of same correct column for student" do
19     let(:student_columns) { ["table1.column", "table1.column"] }
20     let(:instructor_columns) { ["table1.column"] }
21
22     it { expect(subject.rounded_grade).to eq(0.67) }
23   end
24
25   context "example 3 - one correct column and one incorrect for student" do
26     let(:student_columns) { ["table1.column", "table1.column2"] }
27     let(:instructor_columns) { ["table1.column"] }
28
29     it { expect(subject.rounded_grade).to eq(0.67) }
30   end
31
32   context "example 4 - slightly different columns" do
33     let(:student_columns) { ["table1.column"] }
34     let(:instructor_columns) { ["table1.column_2"] }
35
36     it { expect(subject.rounded_grade).to eq(0.17) }
37   end
38
39   context "example 5 - totally different columns" do
40     let(:student_columns) { ["table1.column"] }
41     let(:instructor_columns) { ["table2.column_2"] }
42
43     it { expect(subject.rounded_grade).to eq(0) }
44   end
45 end

```

Source code for spec/sql\_assess/grader/base\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::Base do
4   subject { described_class.new(student_attributes: double, instructor_attributes: double) }
5   context "#levenshtein_distance" do
6     it "returns the correct distance for a and empty string" do
7       expect(subject.levenshtein_distance("a", "")).to eq(1)
8     end
9
10    it "returns the correct distance for a and a" do
11      expect(subject.levenshtein_distance("a", "a")).to eq(0)
12    end
13
14    it "returns the correct distance for a and b" do
15      expect(subject.levenshtein_distance("a", "b")).to eq(1)
16    end
17  end
18 end

```



```

17 it "returns the correct distance for a and ab" do
18   expect(subject.levenshtein_distance("a", "ab")).to eq(1)
19 end
20
21 it "returns the correct distance for ab and ab" do
22   expect(subject.levenshtein_distance("ab", "ab")).to eq(0)
23 end
24
25 it "returns the correct distance for ab and empty string" do
26   expect(subject.levenshtein_distance("ab", "")).to eq(2)
27 end
28
29 it "returns the correct distance for ab and b" do
30   expect(subject.levenshtein_distance("ab", "b")).to eq(1)
31 end
32 end
33 end
34 end

```

Source code for spec/sql\_assess/grader/distinct\_filter\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::DistinctFilter do
4   subject do
5     described_class.new(
6       student_attributes: student_distinct_filter,
7       instructor_attributes: instructor_distinct_filter
8     )
9   end
10
11   context "same filter" do
12     let(:student_distinct_filter) { "ALL" }
13     let(:instructor_distinct_filter) { "ALL" }
14
15     it { expect(subject.rounded_grade).to eq(1) }
16   end
17
18   context "different filter" do
19     let(:student_distinct_filter) { "ALL" }
20     let(:instructor_distinct_filter) { "DISTINCT" }
21
22     it { expect(subject.rounded_grade).to eq(0) }
23   end
24
25   context "different filter - but both including distinct" do
26     let(:student_distinct_filter) { "DISTINCTROW" }
27     let(:instructor_distinct_filter) { "DISTINCT" }
28
29     it { expect(subject.rounded_grade).to eq(0.5) }
30   end
31
32   context "different filter - but both including distinct" do
33     let(:student_distinct_filter) { "DISTINCT" }
34     let(:instructor_distinct_filter) { "DISTINCTROW" }
35
36     it { expect(subject.rounded_grade).to eq(0.5) }
37   end
38 end

```

Source code for spec/sql\_assess/grader/tables\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Grader::Tables do

```

```

4  subject do
5    described_class.new(
6      student_attributes: attributes[:student][:tables],
7      instructor_attributes: attributes[:instructor][:tables]
8    )
9  end
10
11 before do
12   connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
13   connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
14 end
15
16 let(:attributes) do
17   SqlAssess::QueryAttributeExtractor.new.extract(
18     instructor_query, student_query
19   )
20 end
21
22 context "with only base table but different" do
23   let(:student_query) do
24     <<-SQL
25     SELECT a from table1
26     SQL
27   end
28
29   let(:instructor_query) do
30     <<-SQL
31     SELECT a from table2
32     SQL
33   end
34
35   it { expect(subject.rounded_grade).to eq(0) }
36 end
37
38 context "with only base table equal" do
39   let(:student_query) do
40     <<-SQL
41     SELECT a from table1
42     SQL
43   end
44
45   let(:instructor_query) do
46     <<-SQL
47     SELECT a from table1
48     SQL
49   end
50
51   it { expect(subject.rounded_grade).to eq(1) }
52 end
53
54 context "with only base table and join equal" do
55   let(:student_query) do
56     <<-SQL
57     SELECT a from table1 left join table2 on table2.id = table1.id
58     SQL
59   end
60
61   let(:instructor_query) do
62     <<-SQL
63     SELECT a from table1 left join table2 on table2.id = table1.id
64     SQL
65   end
66 end

```

```

67   it { expect(subject.rounded_grade).to eq(1) }
68 end
69
70 context "with base equal but join condition totally different" do
71   let(:student_query) do
72     <<-SQL
73       SELECT a from table1 left join table2 on table2.id = table1.id
74     SQL
75   end
76
77   let(:instructor_query) do
78     <<-SQL
79       SELECT a from table1 left join table3 on table3.id = table1.id2
80     SQL
81   end
82
83   it { expect(subject.rounded_grade).to eq(0.5) }
84 end
85
86 context "with base equal but join type different" do
87   let(:student_query) do
88     <<-SQL
89       SELECT a from table1 left join table2 on table2.id = table1.id
90     SQL
91   end
92
93   let(:instructor_query) do
94     <<-SQL
95       SELECT a from table1 right join table2 on table2.id = table1.id
96     SQL
97   end
98
99   it { expect(subject.rounded_grade).to eq(BigDecimal(0.69, 2)) }
100 end
101
102 context "with base equal but join condition different" do
103   let(:student_query) do
104     <<-SQL
105       SELECT a from table1 left join table2 on table2.id = table1.id
106     SQL
107   end
108
109   let(:instructor_query) do
110     <<-SQL
111       SELECT a from table1 left join table2 on table2.id2 = table1.id
112     SQL
113   end
114
115   it { expect(subject.rounded_grade).to eq(BigDecimal.new(0.69, 2)) }
116 end
117
118 context "with base equal but join condition and type different" do
119   let(:student_query) do
120     <<-SQL
121       SELECT a from table1 left join table2 on table2.id = table1.id
122     SQL
123   end
124
125   let(:instructor_query) do
126     <<-SQL
127       SELECT a from table1 right join table2 on table2.id2 = table1.id
128     SQL
129   end

```

```

30   it { expect(subject.rounded_grade).to eq(0.63) }
31 end
32
33
34 context "with two equal subquery" do
35   let(:student_query) do
36     <<-SQL
37     SELECT id1 from (SELECT id1 from table1)
38     SQL
39   end
40
41   let(:instructor_query) do
42     <<-SQL
43     SELECT id1 from (SELECT id1 from table1)
44     SQL
45   end
46
47   it { expect(subject.rounded_grade).to eq(1) }
48 end
49 end

```

Source code for spec/sql\_assess/query\_comparison\_result\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::QueryComparisonResult do
4    subject { described_class.new(success: true, attributes: attributes) }
5
6    let(:attributes) do
7      SqlAssess::QueryAttributeExtractor.new.extract(
8        (
9          <<-SQL.squish
10          SELECT table1.a
11          FROM table1
12          SQL
13        ), (
14          <<-SQL.squish
15          SELECT table1.a
16          FROM table1
17          SQL
18        )
19      )
20    end
21    context "#attributes_grade" do
22      it "returns a hash" do
23        expect(subject.attributes_grade).to match({
24          columns: an_instance_of(BigDecimal),
25          order_by: an_instance_of(BigDecimal),
26          where: an_instance_of(BigDecimal),
27          distinct_filter: an_instance_of(BigDecimal),
28          limit: an_instance_of(BigDecimal),
29          tables: an_instance_of(BigDecimal),
30          group: an_instance_of(BigDecimal),
31          having: an_instance_of(BigDecimal),
32        })
33      end
34    end
35
36    context "#message" do
37      context "with grade = 100" do
38        before do
39          allow_any_instance_of(described_class).to receive(:calculate_grade).and_return(1)
40        end
41

```

```

42   it { expect(subject.message).to eq("Congratulations! Your solution is correct") }
43 end
44
45 context "with grade < 100" do
46   before do
47     allow_any_instance_of(described_class).to receive(:calculate_grade).and_return(0.9)
48     allow_any_instance_of(described_class).to receive(:first_wrong_component).and_return(component)
49   end
50
51   context "with columns first_wrong_attribute" do
52     let(:component) { :columns }
53
54     it { expect(subject.message).to eq("Your query is not correct. Check what columns you are selecting.") }
55   end
56
57   context "with tables first_wrong_attribute" do
58     let(:component) { :tables }
59
60     it { expect(subject.message).to eq("Your query is not correct. Are you sure you are selecting the
61     ↪ right tables?") }
62   end
63
64   context "with order_by first_wrong_attribute" do
65     let(:component) { :order_by }
66
67     it { expect(subject.message).to eq("Your query is not correct. Are you ordering the rows correctly?") }
68   end
69
70   context "with where first_wrong_attribute" do
71     let(:component) { :where }
72
73     it { expect(subject.message).to eq("Your query is not correct. Looks like you are selecting the right
74     ↪ columns, but you are not selecting only the correct rows.") }
75   end
76
77   context "with distinct_filter first_wrong_attribute" do
78     let(:component) { :distinct_filter }
79
80     it { expect(subject.message).to eq("Your query is not correct. What about duplicates? What does the
81     ↪ exercise say?") }
82   end
83
84   context "with limit first_wrong_attribute" do
85     let(:component) { :limit }
86
87     it { expect(subject.message).to eq("Your query is not correct. Are you selecting the correct number of
88     ↪ rows?") }
89   end
90
91   context "with group first_wrong_attribute" do
92     let(:component) { :group }
93
94     it { expect(subject.message).to eq("Your query is not correct. Are you grouping by the correct
95     ↪ columns?") }
96   end
97 end
98 end
99 end
100 end

```

Source code for spec/sql\_assess/parsers/order\_by\_spec.rb

```

1 require "spec_helper"

```

```

2
3 RSpec.describe SqlAssess::Parsers::OrderBy do
4   subject { described_class.new(query) }
5
6   context "with order by one column" do
7     let(:query) { "SELECT * from table1 ORDER BY id" }
8     it "returns the order by clause" do
9       expect(subject.order).to eq([
10         column: "`id` ASC",
11         position: 0,
12       ])
13     end
14   end
15
16   context "with order by multiple columns" do
17     let(:query) { "SELECT * from table1 ORDER BY id, id2 DESC" }
18     it "returns the order by clause" do
19       expect(subject.order).to eq([
20         {
21           column: "`id` ASC",
22           position: 0,
23         }, {
24           column: "`id2` DESC",
25           position: 1,
26         }
27       ])
28     end
29   end
30
31   context "with order by not present" do
32     let(:query) { "SELECT * from table1" }
33
34     it "returns empty array" do
35       expect(subject.order).to eq([])
36     end
37   end
38 end

```

Source code for spec/sql\_assess/parsers/where\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Parsers::Where do
4   subject { described_class.new(query) }
5
6   context "#where" do
7     context "with no where clause" do
8       let(:query) { "SELECT * from table1" }
9
10      it "returns an empty hash" do
11        expect(subject.where).to eq({})
12      end
13    end
14
15    context "with a single where condition" do
16      context "equal condition" do
17        let(:query) { "SELECT * from table1 WHERE id = 1" }
18
19        it "returns the correct result" do
20          expect(subject.where).to eq({
21            type: "EQUALS",
22            left: "`id`",
23            right: "1",
24            sql: "`id` = 1",

```



```

25     })
26   end
27 end
28
29 context "less condition" do
30   let(:query) { "SELECT * from table1 WHERE id < 1" }
31
32   it "returns the correct result" do
33     expect(subject.where).to eq({
34       type: "LESS",
35       left: "`id`",
36       right: "1",
37       sql: "`id` < 1",
38     })
39   end
40 end
41 end
42
43 context "with an AND conidtion" do
44   context "with two queries" do
45     let(:query) { "SELECT * from table1 WHERE id = 1 AND id < 3" }
46
47     it "returns the correct result" do
48       expect(subject.where).to eq({
49         type: "AND",
50         clauses: [
51           {
52             type: "EQUALS",
53             left: "`id`",
54             right: "1",
55             sql: "`id` = 1",
56           },
57           {
58             type: "LESS",
59             left: "`id`",
60             right: "3",
61             sql: "`id` < 3",
62           }
63         ]
64       })
65     end
66   end
67
68   context "with three queries" do
69     let(:query) { "SELECT * from table1 WHERE id = 1 AND id < 3 AND id < 4" }
70
71     it "returns the correct result" do
72       expect(subject.where).to eq({
73         type: "AND",
74         clauses: [
75           {
76             type: "EQUALS",
77             left: "`id`",
78             right: "1",
79             sql: "`id` = 1",
80           },
81           {
82             type: "LESS",
83             left: "`id`",
84             right: "3",
85             sql: "`id` < 3",
86           },
87           {

```

```

88         type: "LESS",
89         left: "`id`",
90         right: "4",
91         sql: "`id` < 4",
92     }
93 ]
94 })
95 end
96 end
97 end
98
99 context "with an OR condition" do
100   context "with two queries" do
101     let(:query) { "SELECT * from table1 WHERE id = 1 OR id < 3" }
102
103     it "returns the correct result" do
104       expect(subject.where).to eq({
105         type: "OR",
106         clauses: [
107           {
108             type: "EQUALS",
109             left: "`id`",
110             right: "1",
111             sql: "`id` = 1",
112           },
113           {
114             type: "LESS",
115             left: "`id`",
116             right: "3",
117             sql: "`id` < 3",
118           }
119         ]
120       })
121     end
122   end
123
124   context "with three queries" do
125     let(:query) { "SELECT * from table1 WHERE id = 1 OR id < 3 OR id < 4" }
126
127     it "returns the correct result" do
128       expect(subject.where).to eq({
129         type: "OR",
130         clauses: [
131           {
132             type: "EQUALS",
133             left: "`id`",
134             right: "1",
135             sql: "`id` = 1",
136           },
137           {
138             type: "LESS",
139             left: "`id`",
140             right: "3",
141             sql: "`id` < 3",
142           },
143           {
144             type: "LESS",
145             left: "`id`",
146             right: "4",
147             sql: "`id` < 4",
148           }
149         ]
150       })

```

```

51     end
52   end
53 end
54
55 context "with an AND and OR conditions" do
56   let(:query) { "SELECT * from table1 WHERE id = 1 AND id < 3 OR id < 4" }
57
58   it "returns the correct hash" do
59     expect(subject.where).to eq({
60       type: "OR",
61       clauses: [
62         {
63           type: "AND",
64           clauses: [
65             {
66               type: "EQUALS",
67               left: "`id`",
68               right: "1",
69               sql: "`id` = 1",
70             },
71             {
72               type: "LESS",
73               left: "`id`",
74               right: "3",
75               sql: "`id` < 3",
76             },
77           ]
78         },
79         {
80           type: "LESS",
81           left: "`id`",
82           right: "4",
83           sql: "`id` < 4",
84         }
85       ]
86     })
87   end
88 end
89 end
90
91 context '#where_tree' do
92   context 'with no clause' do
93     let(:query) { 'SELECT * from table1' }
94
95     it { expect(subject.where_tree).to eq({}) }
96   end
97
98   context 'with a where clause' do
99     let(:query) { "SELECT * from table1 WHERE #{conditions}" }
100
101     context 'with only one condition' do
102       let(:conditions) { 'a > 1' }
103
104       it 'returns the appropriate tree' do
105         expect(subject.where_tree).to eq({
106           is_inner: false,
107           type: "GREATER",
108           left: "`a`",
109           right: "1",
110           sql: "`a` > 1"
111         })
112       end
113     end
114   end
115 end

```

```

14 context 'with two condition a ^ b' do
15   let(:conditions) { 'a > 1 AND b > 1' }
16
17
18   it 'returns the appropriate tree' do
19     expect(subject.where_tree).to eq({
20       is_inner: true,
21       type: "AND",
22       left_clause: {
23         is_inner: false,
24         type: "GREATER",
25         left: "`a`",
26         right: "1",
27         sql: "`a` > 1"
28       },
29       right_clause: {
30         is_inner: false,
31         type: "GREATER",
32         left: "`b`",
33         right: "1",
34         sql: "`b` > 1"
35       }
36     })
37   end
38 end
39
40 context 'with three conditions a ^ b ^ c' do
41   let(:conditions) { 'a > 1 AND b > 1 AND c > 1' }
42
43   it 'returns the appropriate tree' do
44     expect(subject.where_tree).to eq({
45       is_inner: true,
46       type: "AND",
47       left_clause: {
48         is_inner: true,
49         type: "AND",
50         left_clause: {
51           is_inner: false,
52           type: "GREATER",
53           left: "`a`",
54           right: "1",
55           sql: "`a` > 1"
56         },
57         right_clause: {
58           is_inner: false,
59           type: "GREATER",
60           left: "`b`",
61           right: "1",
62           sql: "`b` > 1"
63         }
64       },
65       right_clause: {
66         is_inner: false,
67         type: "GREATER",
68         left: "`c`",
69         right: "1",
70         sql: "`c` > 1"
71       }
72     })
73   end
74 end
75
76 context 'with three conditions a ^ b V C' do

```

```

77 let(:conditions) { 'a > 1 AND b > 1 OR c > 1' }
78
79 it 'returns the appropriate tree' do
80   expect(subject.where_tree).to eq({
81     is_inner: true,
82     type: "OR",
83     left_clause: {
84       is_inner: true,
85       type: "AND",
86       left_clause: {
87         is_inner: false,
88         type: "GREATER",
89         left: "`a`",
90         right: "1",
91         sql: "`a` > 1"
92       },
93       right_clause: {
94         is_inner: false,
95         type: "GREATER",
96         left: "`b`",
97         right: "1",
98         sql: "`b` > 1"
99       }
100     },
101     right_clause: {
102       is_inner: false,
103       type: "GREATER",
104       left: "`c`",
105       right: "1",
106       sql: "`c` > 1"
107     }
108   })
109 end
110 end
111
112 context 'with three conditions a ^ (b V C)' do
113   let(:conditions) { 'a > 1 AND (b > 1 OR c > 1)' }
114
115   it 'returns the appropriate tree' do
116     expect(subject.where_tree).to eq({
117       is_inner: true,
118       type: "AND",
119       left_clause: {
120         is_inner: false,
121         type: "GREATER",
122         left: "`a`",
123         right: "1",
124         sql: "`a` > 1"
125       },
126       right_clause: {
127         is_inner: true,
128         type: "OR",
129         left_clause: {
130           is_inner: false,
131           type: "GREATER",
132           left: "`b`",
133           right: "1",
134           sql: "`b` > 1"
135         },
136         right_clause: {
137           is_inner: false,
138           type: "GREATER",
139           left: "`c`",

```

```

40         right: "1",
41         sql: "`c` > 1"
42     }
43 },
44 })
45 end
46 end
47
48 context 'with four conditions (a V c)^ (b V C)' do
49   let(:conditions) { '(a > 1 OR c > 1) AND (b > 1 OR c > 1)' }
50
51   it 'returns the appropriate tree' do
52     expect(subject.where_tree).to eq({
53       is_inner: true,
54       type: "AND",
55       left_clause: {
56         is_inner: true,
57         type: "OR",
58         left_clause: {
59           is_inner: false,
60           type: "GREATER",
61           left: "`a`",
62           right: "1",
63           sql: "`a` > 1"
64         },
65         right_clause: {
66           is_inner: false,
67           type: "GREATER",
68           left: "`c`",
69           right: "1",
70           sql: "`c` > 1"
71         }
72       },
73       right_clause: {
74         is_inner: true,
75         type: "OR",
76         left_clause: {
77           is_inner: false,
78           type: "GREATER",
79           left: "`b`",
80           right: "1",
81           sql: "`b` > 1"
82         },
83         right_clause: {
84           is_inner: false,
85           type: "GREATER",
86           left: "`c`",
87           right: "1",
88           sql: "`c` > 1"
89         }
90       },
91     })
92   end
93 end
94 end
95 end
96 end

```

Source code for spec/sql\_assess/parsers/having\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Parsers::Having do
4   subject { described_class.new(query) }

```



```

5 context "#having" do
6   context "with no having clause" do
7     let(:query) { "SELECT * from table1" }
8
9
10    it "returns an empty hash" do
11      expect(subject.having).to eq({})
12    end
13  end
14
15  context "with a single having condition" do
16    context "equal condition" do
17      let(:query) { "SELECT * from table1 HAVING id = 1" }
18
19      it "returns the correct result" do
20        expect(subject.having).to eq({
21          type: "EQUALS",
22          left: "`id`",
23          right: "1",
24          sql: "`id` = 1",
25        })
26      end
27    end
28
29    context "less condition" do
30      let(:query) { "SELECT * from table1 HAVING id < 1" }
31
32      it "returns the correct result" do
33        expect(subject.having).to eq({
34          type: "LESS",
35          left: "`id`",
36          right: "1",
37          sql: "`id` < 1",
38        })
39      end
40    end
41  end
42
43  context "with an AND condition" do
44    context "with two queries" do
45      let(:query) { "SELECT * from table1 HAVING id = 1 AND id < 3" }
46
47      it "returns the correct result" do
48        expect(subject.having).to eq({
49          type: "AND",
50          clauses: [
51            {
52              type: "EQUALS",
53              left: "`id`",
54              right: "1",
55              sql: "`id` = 1",
56            },
57            {
58              type: "LESS",
59              left: "`id`",
60              right: "3",
61              sql: "`id` < 3",
62            }
63          ]
64        })
65      end
66    end
67  end

```

```

68 context "with three queries" do
69   let(:query) { "SELECT * from table1 HAVING id = 1 AND id < 3 AND id < 4" }
70
71   it "returns the correct result" do
72     expect(subject.having).to eq({
73       type: "AND",
74       clauses: [
75         {
76           type: "EQUALS",
77           left: "`id`",
78           right: "1",
79           sql: "`id` = 1",
80         },
81         {
82           type: "LESS",
83           left: "`id`",
84           right: "3",
85           sql: "`id` < 3",
86         },
87         {
88           type: "LESS",
89           left: "`id`",
90           right: "4",
91           sql: "`id` < 4",
92         }
93       ]
94     })
95   end
96 end
97
98
99 context "with an OR condition" do
100   context "with two queries" do
101     let(:query) { "SELECT * from table1 HAVING id = 1 OR id < 3" }
102
103     it "returns the correct result" do
104       expect(subject.having).to eq({
105         type: "OR",
106         clauses: [
107           {
108             type: "EQUALS",
109             left: "`id`",
110             right: "1",
111             sql: "`id` = 1",
112           },
113           {
114             type: "LESS",
115             left: "`id`",
116             right: "3",
117             sql: "`id` < 3",
118           }
119         ]
120       })
121     end
122   end
123
124   context "with three queries" do
125     let(:query) { "SELECT * from table1 HAVING id = 1 OR id < 3 OR id < 4" }
126
127     it "returns the correct result" do
128       expect(subject.having).to eq({
129         type: "OR",
130         clauses: [

```

```

31     {
32       type: "EQUALS",
33       left: "`id`",
34       right: "1",
35       sql: "`id` = 1",
36     },
37     {
38       type: "LESS",
39       left: "`id`",
40       right: "3",
41       sql: "`id` < 3",
42     },
43     {
44       type: "LESS",
45       left: "`id`",
46       right: "4",
47       sql: "`id` < 4",
48     }
49   ]
50 })
51 end
52 end
53 end
54
55 context "with an AND and OR conditions" do
56   let(:query) { "SELECT * from table1 HAVING id = 1 AND id < 3 OR id < 4" }
57
58   it "returns the correct hash" do
59     expect(subject.having).to eq({
60       type: "OR",
61       clauses: [
62         {
63           type: "AND",
64           clauses: [
65             {
66               type: "EQUALS",
67               left: "`id`",
68               right: "1",
69               sql: "`id` = 1",
70             },
71             {
72               type: "LESS",
73               left: "`id`",
74               right: "3",
75               sql: "`id` < 3",
76             }
77           ],
78         },
79         {
80           type: "LESS",
81           left: "`id`",
82           right: "4",
83           sql: "`id` < 4",
84         }
85       ]
86     })
87   end
88 end
89 end
90
91 context '#having_tree' do
92   context 'with no clause' do
93     let(:query) { 'SELECT * from table1' }

```

```

94   it { expect(subject.having_tree).to eq({}) }
95 end
96
97
98 context 'with a having clause' do
99   let(:query) { "SELECT * from table1 HAVING #{conditions}" }
100
101   context 'with only one condition' do
102     let(:conditions) { 'a > 1' }
103
104     it 'returns the appropriate tree' do
105       expect(subject.having_tree).to eq({
106         is_inner: false,
107         type: "GREATER",
108         left: "`a`",
109         right: "1",
110         sql: "`a` > 1"
111       })
112     end
113   end
114
115   context 'with two condition a ^ b' do
116     let(:conditions) { 'a > 1 AND b > 1' }
117
118     it 'returns the appropriate tree' do
119       expect(subject.having_tree).to eq({
120         is_inner: true,
121         type: "AND",
122         left_clause: {
123           is_inner: false,
124           type: "GREATER",
125           left: "`a`",
126           right: "1",
127           sql: "`a` > 1"
128         },
129         right_clause: {
130           is_inner: false,
131           type: "GREATER",
132           left: "`b`",
133           right: "1",
134           sql: "`b` > 1"
135         }
136       })
137     end
138   end
139
140   context 'with three conditions a ^ b ^ c' do
141     let(:conditions) { 'a > 1 AND b > 1 AND c > 1' }
142
143     it 'returns the appropriate tree' do
144       expect(subject.having_tree).to eq({
145         is_inner: true,
146         type: "AND",
147         left_clause: {
148           is_inner: true,
149           type: "AND",
150           left_clause: {
151             is_inner: false,
152             type: "GREATER",
153             left: "`a`",
154             right: "1",
155             sql: "`a` > 1"
156           },

```

```

57     right_clause: {
58       is_inner: false,
59       type: "GREATER",
60       left: "`b`",
61       right: "1",
62       sql: "`b` > 1"
63     }
64   },
65   right_clause: {
66     is_inner: false,
67     type: "GREATER",
68     left: "`c`",
69     right: "1",
70     sql: "`c` > 1"
71   }
72 })
73 end
74 end
75
76 context 'with three conditions a ^ b V C' do
77   let(:conditions) { 'a > 1 AND b > 1 OR c > 1' }
78
79   it 'returns the appropriate tree' do
80     expect(subject.having_tree).to eq({
81       is_inner: true,
82       type: "OR",
83       left_clause: {
84         is_inner: true,
85         type: "AND",
86         left_clause: {
87           is_inner: false,
88           type: "GREATER",
89           left: "`a`",
90           right: "1",
91           sql: "`a` > 1"
92         },
93         right_clause: {
94           is_inner: false,
95           type: "GREATER",
96           left: "`b`",
97           right: "1",
98           sql: "`b` > 1"
99         }
100       },
101       right_clause: {
102         is_inner: false,
103         type: "GREATER",
104         left: "`c`",
105         right: "1",
106         sql: "`c` > 1"
107       }
108     })
109   end
110 end
111
112 context 'with three conditions a ^ (b V C)' do
113   let(:conditions) { 'a > 1 AND (b > 1 OR c > 1)' }
114
115   it 'returns the appropriate tree' do
116     expect(subject.having_tree).to eq({
117       is_inner: true,
118       type: "AND",
119       left_clause: {

```

```

20     is_inner: false,
21     type: "GREATER",
22     left: "`a`",
23     right: "1",
24     sql: "`a` > 1"
25 },
26 right_clause: {
27   is_inner: true,
28   type: "OR",
29   left_clause: {
30     is_inner: false,
31     type: "GREATER",
32     left: "`b`",
33     right: "1",
34     sql: "`b` > 1"
35   },
36   right_clause: {
37     is_inner: false,
38     type: "GREATER",
39     left: "`c`",
40     right: "1",
41     sql: "`c` > 1"
42   }
43 },
44 })
45 end
46 end
47
48 context 'with four conditions (a V c)^ (b V C)' do
49   let(:conditions) { '(a > 1 OR c > 1) AND (b > 1 OR c > 1)' }
50
51   it 'returns the appropriate tree' do
52     expect(subject.having_tree).to eq({
53       is_inner: true,
54       type: "AND",
55       left_clause: {
56         is_inner: true,
57         type: "OR",
58         left_clause: {
59           is_inner: false,
60           type: "GREATER",
61           left: "`a`",
62           right: "1",
63           sql: "`a` > 1"
64         },
65         right_clause: {
66           is_inner: false,
67           type: "GREATER",
68           left: "`c`",
69           right: "1",
70           sql: "`c` > 1"
71         }
72       },
73       right_clause: {
74         is_inner: true,
75         type: "OR",
76         left_clause: {
77           is_inner: false,
78           type: "GREATER",
79           left: "`b`",
80           right: "1",
81           sql: "`b` > 1"
82         },

```

```

83     right_clause: {
84       is_inner: false,
85       type: "GREATER",
86       left: "`c`",
87       right: "1",
88       sql: "`c` > 1"
89     }
90   },
91 })
92 end
93 end
94 end
95 end
96 end

```

Source code for spec/sql\_assess/parsers/limit\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Parsers::Limit do
4    subject { described_class.new(query) }
5
6    context "with no limit" do
7      let(:query) { "SELECT * from table1" }
8      it "returns the correct limit" do
9        expect(subject.limit).to eq({
10          "limit": "inf",
11          "offset": 0
12        })
13      end
14    end
15
16    context "with limit but no offset" do
17      let(:query) { "SELECT * from table1 LIMIT 1" }
18      it "returns the correct limit" do
19        expect(subject.limit).to eq({
20          "limit": 1,
21          "offset": 0
22        })
23      end
24    end
25
26    context "with limit and offsert" do
27      let(:query) { "SELECT * from table1 LIMIT 1 OFFSET 2" }
28
29      it "returns the correct limit" do
30        expect(subject.limit).to eq({
31          "limit": 1,
32          "offset": 2
33        })
34      end
35    end
36  end

```

Source code for spec/sql\_assess/parsers/group\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Parsers::Group do
4    subject { described_class.new(query) }
5
6    context "with no group" do
7      let(:query) { "SELECT id FROM t1 " }
8      it "returns star" do
9        expect(subject.group).to eq([])

```

```

10   end
11 end
12
13 context "with one column in group" do
14   let(:query) { "SELECT id, id2 FROM t1 GROUP BY id" }
15   it "returns star" do
16     expect(subject.group).to eq(["`id`"])
17   end
18 end
19
20 context "with two columns in group" do
21   let(:query) { "SELECT id, id2 FROM t1 GROUP BY id, id2" }
22   it "returns star" do
23     expect(subject.group).to eq(["`id`", "`id2`"])
24   end
25 end
26 end

```

Source code for spec/sql\_assess/parsers/columns\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Parsers::Columns do
4   subject { described_class.new(query) }
5
6   context "with one column in select" do
7     let(:query) { "SELECT id" }
8     it "returns star" do
9       expect(subject.columns).to eq(["`id`"])
10     end
11   end
12
13   context "with two column in select" do
14     let(:query) { "SELECT id, id2" }
15     it "returns star" do
16       expect(subject.columns).to eq(["`id`", "`id2`"])
17     end
18   end
19 end

```

Source code for spec/sql\_assess/parsers/distinct\_filter\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Parsers::DistinctFilter do
4   subject { described_class.new(query) }
5
6   context "with no filter in select" do
7     let(:query) { "SELECT id, id2" }
8     it "returns ALL" do
9       expect(subject.distinct_filter).to eq("ALL")
10     end
11   end
12
13   context "with ALL in select" do
14     let(:query) { "SELECT ALL id, id2" }
15     it "returns ALL" do
16       expect(subject.distinct_filter).to eq("ALL")
17     end
18   end
19
20   context "with DISTINCTROW in select" do
21     let(:query) { "SELECT DISTINCTROW id, id3" }
22     it "returns DISTINCTROW" do
23       expect(subject.distinct_filter).to eq("DISTINCTROW")
24     end
25   end
26 end

```



```

24   end
25 end
26
27 context "with DISTINCT in select" do
28   let(:query) { "SELECT DISTINCT C1, c2, c3 FROM t1" }
29   it "returns DISTINCT" do
30     expect(subject.distinct_filter).to eq("DISTINCT")
31   end
32 end
33 end

```

Source code for spec/sql\_assess/parsers/tables\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Parsers::Tables do
4    subject { described_class.new(query) }
5
6    context "with no table" do
7      let(:query) { "SELECT 1" }
8
9      it "returns an empty array" do
10        expect(subject.tables).to eq([])
11      end
12    end
13
14    context "with one table" do
15      let(:query) { "SELECT * from table1" }
16      it "returns an array containing the tables" do
17        expect(subject.tables).to eq([
18          {
19            type: "table",
20            table: "`table1`",
21            sql: "`table1`",
22          }
23        ])
24      end
25    end
26
27    context "with a cross join" do
28      let(:query) { "SELECT * from table1, table2" }
29
30      it "returns an array containing the tables" do
31        expect(subject.tables).to eq([
32          {
33            type: "table",
34            table: "`table1`",
35            sql: "`table1`",
36          },
37          {
38            join_type: "CROSS JOIN",
39            table: {
40              type: "table",
41              table: "`table2`",
42              sql: "`table2`",
43            },
44            sql: "CROSS JOIN `table2`",
45          }
46        ])
47      end
48    end
49
50    context "a table and a inner join" do
51      let(:query) { "SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.id" }

```

```

52 it "returns an array containing the tables" do
53   expect(subject.tables).to eq([
54     {
55       type: "table",
56       table: "`table1`",
57       sql: "`table1`",
58     },
59     {
60       join_type: "INNER JOIN",
61       table: {
62         type: "table",
63         table: "`table2`",
64         sql: "`table2`",
65       },
66       sql: "INNER JOIN `table2` ON `table1`.`id` = `table2`.`id`",
67       condition: {
68         type: "EQUALS",
69         left: "`table1`.`id`",
70         right: "`table2`.`id`",
71         sql: "`table1`.`id` = `table2`.`id`"
72       }
73     }
74   ])
75 end
76 end
77
78 context "a table and two left join" do
79   let(:query) do
80     <<-SQL.squish
81     SELECT *
82     FROM
83       table1
84       LEFT JOIN table2 ON table1.id = table2.id
85       LEFT JOIN table3 ON table3.id = table2.id
86     SQL
87   end
88 end
89
90 it "returns an array containing the tables" do
91   expect(subject.tables).to eq([
92     {
93       type: "table",
94       table: "`table1`",
95       sql: "`table1`",
96     },
97     {
98       join_type: "LEFT JOIN",
99       table: {
100         type: "table",
101         table: "`table2`",
102         sql: "`table2`",
103       },
104       condition: {
105         type: "EQUALS",
106         left: "`table1`.`id`",
107         right: "`table2`.`id`",
108         sql: "`table1`.`id` = `table2`.`id`"
109       },
110       sql: "LEFT JOIN `table2` ON `table1`.`id` = `table2`.`id`"
111     },
112     {
113       join_type: "LEFT JOIN",
114       table: {

```

```

15     type: "table",
16     table: "`table3`",
17     sql: "`table3`",
18   },
19   condition: {
20     type: "EQUALS",
21     left: "`table3`.`id`",
22     right: "`table2`.`id`",
23     sql: "`table3`.`id` = `table2`.`id`"
24   },
25   sql: "LEFT JOIN `table3` ON `table3`.`id` = `table2`.`id`"
26 }
27 ])
28 end
29 end
30
31 context "a subquery" do
32   let(:query) do
33     <<-SQL.squish
34     SELECT *
35     FROM (SELECT id FROM table1)
36     SQL
37   end
38
39   it "returns an array containing the tables" do
40     expect(subject.tables).to eq([
41       {
42         type: "Subquery",
43         sql: "(SELECT `id` FROM `table1`)",
44         attributes: {
45           columns: ["`id`"],
46           order_by: [],
47           where: {},
48           where_tree: {},
49           tables: [{type: "table", table: "`table1`", sql: "`table1`"}],
50           distinct_filter: "ALL",
51           limit: {limit: "inf", offset: 0},
52           group: [],
53           having: {},
54           having_tree: {},
55         }
56       }
57     ])
58   end
59 end
60 end

```

Source code for spec/sql\_assess/assessor\_spec.rb

```

1 require "spec_helper"
2 require "yaml"
3
4 RSpec.describe SqlAssess::Assessor do
5   before do
6     allow(SqlAssess::DatabaseConnection).to receive(:new).and_return(@shared_connection)
7   end
8
9   context "#compile" do
10     context "without any errors" do
11       it "returns the result from data extractor" do
12         result = subject.compile(
13           create_schema_sql_query: "CREATE TABLE table1 (id integer)",
14           instructor_sql_query: "SELECT * from table1",
15           seed_sql_query: "INSERT INTO table1 (id) VALUES (1)"

```

```

16     )
17
18     expect(result).to eq([
19       {
20         name: "table1",
21         columns: [
22           {
23             name: "id",
24             type: "int(11)"
25           },
26         ],
27         data: [
28           { "id" => 1 }
29         ],
30       }
31     ])
32   end
33 end
34
35 context "#assess" do
36   let(:schema_sql_query) { "CREATE TABLE table1 (id integer)" }
37   let(:instructor_sql_query) { "SELECT * from table1" }
38   let(:seed_sql_query) { "INSERT INTO table1 (id) VALUES (1)" }
39
40   context "with a wrong student query" do
41     let(:student_sql_query) { "SELECT * from table2" }
42     it "raises an error and clears the database" do
43       expect { do_assess }.to raise_error(SqlAssess::DatabaseQueryExecutionFailed)
44
45       tables = subject.connection.query("SHOW tables");
46       expect(tables.size).to eq(0)
47     end
48   end
49
50   context "with a correct student query" do
51     let(:student_sql_query) { "SELECT * from table1" }
52     it "returns a result" do
53       expect(do_assess).to be_a(SqlAssess::QueryComparisonResult)
54     end
55   end
56 end
57
58 yaml = YAML.load_file("spec/fixtures/assessor_integration_tests.yml")
59
60 yaml.each_with_index do |test, i|
61   it "correctly assess integration test #{i}" do
62     result = subject.assess(
63       create_schema_sql_query: test["schema"],
64       instructor_sql_query: test["instructor_query"],
65       seed_sql_query: test["seed"],
66       student_sql_query: test["student_query"]
67     )
68     expect(result.message).to eq(test["message"])
69   end
70 end
71
72 private
73
74 def do_assess
75   subject.assess(
76     create_schema_sql_query: schema_sql_query,
77     instructor_sql_query: instructor_sql_query,
78     seed_sql_query: seed_sql_query,
79     student_sql_query: student_sql_query

```

```

79 )
80 end
81 end

Source code for spec/sql_assess/query_comparator_spec.rb

```

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::QueryComparator do
4   subject { described_class.new(connection) }
5
6   context "success" do
7     before do
8       connection.query('CREATE TABLE table1 (id integer);')
9       connection.query('INSERT INTO table1 (id) values(1);')
10      connection.query('INSERT INTO table1 (id) values(2);')
11    end
12
13    context "when the results are the same" do
14      it "returns the right result" do
15        query = "SELECT * from table1 WHERE id = 1";
16
17        expect(subject.compare(query, query)).to eq(true)
18      end
19    end
20
21    context "when the results are different" do
22      context "when the count is different" do
23        it "returns the right result" do
24          query = "SELECT * from table1 WHERE id = 1";
25          wrong_query = "SELECT * from table1 WHERE id = 3";
26
27          expect(subject.compare(query, wrong_query)).to eq(false)
28        end
29      end
30
31      context "when the count is the same" do
32        it "returns the right result" do
33          query = "SELECT * from table1 WHERE id = 1";
34          wrong_query = "SELECT * from table1 WHERE id = 2";
35
36          expect(subject.compare(query, wrong_query)).to eq(false)
37        end
38      end
39    end
40  end
41 end

```

Source code for spec/sql\_assess/query\_attribute\_extractor\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::QueryAttributeExtractor do
4   subject { described_class.new }
5
6   context "columns" do
7     before do
8       connection.query('CREATE TABLE table1 (id integer, second integer);')
9       connection.query('INSERT INTO table1 (id, second) values(1, 3);')
10      connection.query('INSERT INTO table1 (id, second) values(2, 4);')
11    end
12
13    let(:instructor_query) { "SELECT id from table1" }
14    let(:student_query) { "SELECT second from table1" }
15

```

```

16 it "returns the correct format" do
17   result = subject.extract(instructor_query, student_query)
18   expect(result).to match({
19     student: {
20       columns: an_instance_of(Array),
21       order_by: an_instance_of(Array),
22       where: an_instance_of(Hash),
23       where_tree: an_instance_of(Hash),
24       tables: an_instance_of(Array),
25       distinct_filter: an_instance_of(String),
26       limit: an_instance_of(Hash),
27       group: an_instance_of(Array),
28       having: an_instance_of(Hash),
29       having_tree: an_instance_of(Hash),
30     },
31     instructor: {
32       columns: an_instance_of(Array),
33       order_by: an_instance_of(Array),
34       where: an_instance_of(Hash),
35       where_tree: an_instance_of(Hash),
36       tables: an_instance_of(Array),
37       distinct_filter: an_instance_of(String),
38       limit: an_instance_of(Hash),
39       group: an_instance_of(Array),
40       having: an_instance_of(Hash),
41       having_tree: an_instance_of(Hash),
42     },
43   })
44 end
45 end
46 end

```

Source code for spec/sql\_assess/query\_transformer\_spec.rb

```

1 require "spec_helper"
2 require 'yaml'
3
4 RSpec.describe SqlAssess::QueryTransformer do
5   subject { described_class.new(connection) }
6
7   context "when encountering an error" do
8     it "raises a CanonicalizationError" do
9       expect { subject.transform("adad * from a") }
10        .to raise_error(SqlAssess::CanonicalizationError)
11     end
12   end
13
14   yaml = YAML.load_file("spec/fixtures/transformer_integration_tests.yml")
15
16   yaml.each do |test|
17     it "transform #{test['query']} to #{test['expected_result']} do
18       # Seed data
19       connection.multiple_query(test["schema"])
20       # Check if queries from file are correct
21       connection.query(test["query"])
22       connection.query(test["expected_result"])
23       # Check transformation
24       expect(subject.transform(test["query"])).to eq(test["expected_result"])
25     end
26   end
27
28   yaml2 = YAML.load_file("spec/fixtures/transformer_hacker_rank_integration_tests.yml")
29
30   yaml2.each do |test|

```

```

31 if test["support"] == false
32  xit "#{test['name']}" do
33   execute_query(test)
34 end
35 else
36   it "#{test['name']}: transform #{test['query'].squish} to #{test['expected_result'].squish}" do
37     execute_query(test)
38   end
39 end
40 end
41
42 def execute_query(test)
43   # Seed data
44   connection.multiple_query(test["schema"])
45   # Check if queries from file are correct
46   connection.query(test["query"])
47   connection.query(test["expected_result"])
48   # Check transformation
49   expect(subject.transform(test["query"])).to eq(test["expected_result"].squish)
50 end
51 end

```

Source code for spec/sql\_assess/transformers/ambiguous\_columns/order\_by\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::AmbiguousColumns::OrderBy do
4   subject { described_class.new(connection) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9   end
10
11   context "with no order clause" do
12     let(:query) do
13       <<-SQL.squish
14         SELECT `table1`.`id`, `table1`.`id2`
15         FROM `table1`
16       SQL
17     end
18
19     it "doesn't change the query" do
20       expect(subject.transform(query)).to eq(query)
21     end
22   end
23
24   context "with order clause but no ambiguous column" do
25     let(:query) do
26       <<-SQL.squish
27         SELECT `table1`.`id`, `table1`.`id2`
28         FROM `table1`
29         ORDER BY `table1`.`id1` ASC
30       SQL
31     end
32
33     it "doesn't change the query" do
34       expect(subject.transform(query)).to eq(query)
35     end
36   end
37
38   context "with order clause but with an ambiguous column" do
39     let(:query) do
40       <<-SQL.squish

```

```

41     SELECT `table1`.`id`, `table1`.`id2`
42     FROM `table1`
43     ORDER BY `id1` ASC
44   SQL
45 end
46
47 it "changes the query" do
48   expect(subject.transform(query)).to eq(
49     <<-SQL.squish
50       SELECT `table1`.`id`, `table1`.`id2`
51       FROM `table1`
52       ORDER BY `table1`.`id1` ASC
53     SQL
54   )
55 end
56 end
57
58 context "with order clause but with a column number" do
59   let(:query) do
60     <<-SQL.squish
61       SELECT `table1`.`id`, `table1`.`id2`
62       FROM `table1`
63       ORDER BY 1 ASC
64     SQL
65   end
66
67   it "changes the query" do
68     expect(subject.transform(query)).to eq(
69       <<-SQL.squish
70         SELECT `table1`.`id`, `table1`.`id2`
71         FROM `table1`
72         ORDER BY `table1`.`id` ASC
73       SQL
74     )
75   end
76 end
77 end

```

Source code for spec/sql\_assess/transformers/ambiguous\_columns/where\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::AmbiguousColumns::Where do
4   subject { described_class.new(connection) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9   end
10
11   context "with no where clause" do
12     let(:query) do
13       <<-SQL.squish
14         SELECT `table1`.`id`, `table1`.`id2`
15         FROM `table1`
16       SQL
17     end
18
19     it "doesn't change the query" do
20       expect(subject.transform(query)).to eq(query)
21     end
22   end
23
24   context "with WHERE clause but no ambiguous column" do

```



```

25 let(:query) do
26   <<-SQL.squish
27     SELECT `table1`.`id`, `table1`.`id2`
28     FROM `table1`
29     WHERE `table1`.`id1` > 1
30   SQL
31 end
32
33 it "doesn't change the query" do
34   expect(subject.transform(query)).to eq(query)
35 end
36 end
37
38 context "with where clause but with an ambiguous column" do
39   let(:query) do
40     <<-SQL.squish
41       SELECT `table1`.`id`, `table1`.`id2`
42       FROM `table1`
43       WHERE `id1` > 1
44     SQL
45   end
46
47   it "changes the query" do
48     expect(subject.transform(query)).to eq(
49       <<-SQL.squish
50         SELECT `table1`.`id`, `table1`.`id2`
51         FROM `table1`
52         WHERE `table1`.`id1` > 1
53       SQL
54     )
55   end
56 end
57
58 context "with where clause but with an ambiguous column" do
59   let(:query) do
60     <<-SQL.squish
61       SELECT `table1`.`id`, `table1`.`id2`
62       FROM `table1`
63       WHERE `id1` > 1 AND `id2` > 1
64     SQL
65   end
66
67   it "changes the query" do
68     expect(subject.transform(query)).to eq(
69       <<-SQL.squish
70         SELECT `table1`.`id`, `table1`.`id2`
71         FROM `table1`
72         WHERE (`table1`.`id1` > 1 AND `table1`.`id2` > 1)
73       SQL
74     )
75   end
76 end
77 end

```

Source code for spec/sql\_assess/transformers/ambiguous\_columns/having\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::AmbiguousColumns::Having do
4   subject { described_class.new(connection) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")

```

```

9   end
10
11  context "with no HAVING clause" do
12    let(:query) do
13      <<-SQL.squish
14        SELECT `table1`.`id`, `table1`.`id2`
15        FROM `table1`
16      SQL
17    end
18
19    it "doesn't change the query" do
20      expect(subject.transform(query)).to eq(query)
21    end
22  end
23
24  context "with HAVING clause but no ambiguous column" do
25    let(:query) do
26      <<-SQL.squish
27        SELECT `table1`.`id`, `table1`.`id2`
28        FROM `table1`
29        HAVING `table1`.`id1` > 1
30      SQL
31    end
32
33    it "doesn't change the query" do
34      expect(subject.transform(query)).to eq(query)
35    end
36  end
37
38  context "with HAVING clause but with an ambiguous column" do
39    let(:query) do
40      <<-SQL.squish
41        SELECT `table1`.`id`, `table1`.`id2`
42        FROM `table1`
43        HAVING `id1` > 1
44      SQL
45    end
46
47    it "changes the query" do
48      expect(subject.transform(query)).to eq(
49        <<-SQL.squish
50          SELECT `table1`.`id`, `table1`.`id2`
51          FROM `table1`
52          HAVING `table1`.`id1` > 1
53        SQL
54      )
55    end
56  end
57
58  context "with HAVING clause but with an ambiguous column" do
59    let(:query) do
60      <<-SQL.squish
61        SELECT `table1`.`id`, `table1`.`id2`
62        FROM `table1`
63        HAVING `id1` > 1 AND `id2` > 1
64      SQL
65    end
66
67    it "changes the query" do
68      expect(subject.transform(query)).to eq(
69        <<-SQL.squish
70          SELECT `table1`.`id`, `table1`.`id2`
71          FROM `table1`

```

```

72         HAVING (`table1`.`id1` > 1 AND `table1`.`id2` > 1)
73         SQL
74     )
75 end
76 end
77
78 end

```

Source code for spec/sql\_assess/transformers/ambiguous\_columns/group\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Transformers::AmbiguousColumns::Group do
4    subject { described_class.new(connection) }
5
6    before do
7      connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8      connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9    end
10
11    context "with no group clause" do
12      let(:query) do
13        <<-SQL.squish
14          SELECT `table1`.`id`, `table1`.`id2`
15          FROM `table1`
16        SQL
17      end
18
19      it "doesn't change the query" do
20        expect(subject.transform(query)).to eq(query)
21      end
22    end
23
24    context "with group clause but no ambiguous column" do
25      let(:query) do
26        <<-SQL.squish
27          SELECT `table1`.`id`, `table1`.`id2`
28          FROM `table1`
29          GROUP BY `table1`.`id1`
30        SQL
31      end
32
33      it "doesn't change the query" do
34        expect(subject.transform(query)).to eq(query)
35      end
36    end
37
38    context "with group clause but with an ambiguous column" do
39      let(:query) do
40        <<-SQL.squish
41          SELECT `table1`.`id`, `table1`.`id2`
42          FROM `table1`
43          GROUP BY `id1`
44        SQL
45      end
46
47      it "changes the query" do
48        expect(subject.transform(query)).to eq(
49          <<-SQL.squish
50            SELECT `table1`.`id`, `table1`.`id2`
51            FROM `table1`
52            GROUP BY `table1`.`id1`
53          SQL
54        )

```

```

55     end
56 end
57
58 context "with group clause but with a column number" do
59   let(:query) do
60     <<-SQL.squish
61       SELECT `table1`.`id`, `table1`.`id2`
62       FROM `table1`
63       GROUP BY 1
64     SQL
65   end
66
67   it "changes the query" do
68     expect(subject.transform(query)).to eq(
69       <<-SQL.squish
70         SELECT `table1`.`id`, `table1`.`id2`
71         FROM `table1`
72         GROUP BY `table1`.`id`
73       SQL
74     )
75   end
76 end
77 end

```

Source code for spec/sql\_assess/transformers/ambiguous\_columns/select\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::AmbiguousColumns::Select do
4   subject { described_class.new(connection) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9   end
10
11   it "adds the table name in front of the column" do
12     expect(subject.transform("SELECT id1 FROM table1"))
13       .to eq("SELECT `table1`.`id1` FROM `table1`")
14   end
15
16   it "leaves existing qualified columns unchanged" do
17     expect(subject.transform("SELECT table1.id1 FROM table1"))
18       .to eq("SELECT `table1`.`id1` FROM `table1`")
19   end
20
21   it "adds the table name in front of the column" do
22     expect(subject.transform("SELECT id1, id3 FROM table1, table2"))
23       .to eq("SELECT `table1`.`id1`, `table2`.`id3` FROM `table1` CROSS JOIN `table2`")
24   end
25
26   it "transforms the query" do
27     expect(subject.transform("SELECT SUM(id1) FROM table1")).to eq("SELECT SUM(`table1`.`id1`) FROM `table1`")
28   end
29 end

```

Source code for spec/sql\_assess/transformers/ambiguous\_columns/from\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::AmbiguousColumns::From do
4   subject { described_class.new(connection) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")

```

```

8 connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9 end
10
11 context "with no join" do
12   let(:query) do
13     <<-SQL.squish
14       SELECT `table1`.`id`, `table1`.`id2`
15       FROM `table1`
16     SQL
17   end
18
19   it "doesn't change the query" do
20     expect(subject.transform(query)).to eq(query)
21   end
22 end
23
24 context "with join clause but no ambiguous column" do
25   let(:query) do
26     <<-SQL.squish
27       SELECT `table1`.`id`, `table1`.`id2`
28       FROM `table1` LEFT JOIN `table2` ON `table2`.`id3` = `table1`.`id1`
29     SQL
30   end
31
32   it "doesn't change the query" do
33     expect(subject.transform(query)).to eq(query)
34   end
35 end
36
37 context "with join clause but with ambiguous column" do
38   let(:query) do
39     <<-SQL.squish
40       SELECT `table1`.`id`, `table1`.`id2`
41       FROM `table1` LEFT JOIN `table2` ON `id3` = `id1`
42       GROUP BY `id1`
43     SQL
44   end
45
46   it "changes the query" do
47     expect(subject.transform(query)).to eq(
48       <<-SQL.squish
49         SELECT `table1`.`id`, `table1`.`id2`
50         FROM `table1` LEFT JOIN `table2` ON `table2`.`id3` = `table1`.`id1`
51         GROUP BY `id1`
52       SQL
53     )
54   end
55 end
56
57 context "with join clause but with ambiguous column" do
58   let(:query) do
59     <<-SQL.squish
60       SELECT `table1`.`id`, `table1`.`id2`
61       FROM `table1` LEFT JOIN `table2` ON `id3` = `id1` AND `id4` = `id2`
62       GROUP BY `id1`
63     SQL
64   end
65
66   it "changes the query" do
67     expect(subject.transform(query)).to eq(
68       <<-SQL.squish
69         SELECT `table1`.`id`, `table1`.`id2`

```

```

70         FROM `table1` LEFT JOIN `table2` ON (`table2`.`id3` = `table1`.`id1` AND `table2`.`id4` =
↪ `table1`.`id2`)
71         GROUP BY `id1`
72     SQL
73 )
74 end
75 end
76 end

```

Source code for spec/sql\_assess/transformers/from\_subquery\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::FromSubquery do
4   subject { described_class.new(connection).transform(sql) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9   end
10
11   context "no subquery" do
12     let(:sql) do
13       <<-SQL.squish
14       SELECT `table1`.`id`
15       FROM
16         `table1`
17       LEFT JOIN `table2` ON `table1`.`id1` = `table2`.`id3`
18     SQL
19   end
20
21   it "returns the same query" do
22     expect(subject).to eq(sql)
23   end
24 end
25
26
27 context "with subquery" do
28   let(:sql) do
29     <<-SQL.squish
30     SELECT `table1`.`id`
31     FROM (SELECT * from table1)
32   SQL
33 end
34
35 it "returns the same query" do
36   expect(subject).to eq("SELECT `table1`.`id` FROM (SELECT `table1`.`id1`, `table1`.`id2` FROM `table1`)")
37 end
38 end
39
40 context "with subquery left join" do
41   let(:sql) do
42     <<-SQL.squish
43     SELECT `table1`.`id`
44
45     FROM table2 LEFT JOIN (SELECT * from table1) ON table1.id1 = table2.id3
46   SQL
47 end
48
49 it "returns the same query" do
50   expect(subject).to eq("SELECT `table1`.`id` FROM `table2` LEFT JOIN (SELECT `table1`.`id1`,
↪ `table1`.`id2` FROM `table1`) ON `table1`.`id1` = `table2`.`id3`")
51 end
52 end

```

```
53 end

Source code for spec/sql_assess/transformers/not/where_spec.rb
```

```
1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::Not::Where do
4   subject { described_class.new(connection) }
5
6   context "when there is no where clause" do
7     it "returns the same query" do
8       expect(subject.transform("SELECT * FROM t1")).to eq("SELECT * FROM t1")
9     end
10  end
11
12  context "when there is a where clause" do
13    context "with no NOT query" do
14      it "returns the same query" do
15        expect(subject.transform("SELECT * FROM t1 WHERE a = 1"))
16          .to eq("SELECT * FROM `t1` WHERE `a` = 1")
17      end
18    end
19
20    context "with only a between query" do
21      context "with a > clause" do
22        it "returns the updated query" do
23          expect(subject.transform("SELECT * FROM t1 WHERE NOT a > 1"))
24            .to eq("SELECT * FROM `t1` WHERE `a` <= 1")
25        end
26      end
27
28      context "with a < clause" do
29        it "returns the updated query" do
30          expect(subject.transform("SELECT * FROM t1 WHERE NOT a < 1"))
31            .to eq("SELECT * FROM `t1` WHERE `a` >= 1")
32        end
33      end
34
35      context "with a <= clause" do
36        it "returns the updated query" do
37          expect(subject.transform("SELECT * FROM t1 WHERE NOT a <= 1"))
38            .to eq("SELECT * FROM `t1` WHERE `a` > 1")
39        end
40      end
41
42      context "with a >= clause" do
43        it "returns the updated query" do
44          expect(subject.transform("SELECT * FROM t1 WHERE NOT a >= 1"))
45            .to eq("SELECT * FROM `t1` WHERE `a` < 1")
46        end
47      end
48    end
49
50    context "with a not query and another type of query" do
51      it "returns the updated query" do
52        expect(subject.transform("SELECT * FROM t1 WHERE NOT a > 1 AND b = 2"))
53          .to eq("SELECT * FROM `t1` WHERE (`a` <= 1 AND `b` = 2)")
54      end
55    end
56
57    context "with a not query and two other type of query" do
58      it "returns the updated query" do
59        expect(subject.transform("SELECT * FROM t1 WHERE NOT a > 1 AND b = 2 AND c = 3"))
60          .to eq("SELECT * FROM `t1` WHERE ((`a` <= 1 AND `b` = 2) AND `c` = 3)")
61      end
62    end
63  end
64 end
```

```

61   end
62 end
63
64 context "with a not which is not transformable" do
65   it "returns the updated query" do
66     expect(subject.transform("SELECT * FROM t1 WHERE NOT a LIKE 'a'"))
67       .to eq("SELECT * FROM `t1` WHERE `a` NOT LIKE 'a'")
68   end
69 end
70 end
71 end

```

Source code for spec/sql\_assess/transformers/not/having\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Transformers::Not::Having do
4    subject { described_class.new(connection) }
5
6    context "when there is no HAVING clause" do
7      it "returns the same query" do
8        expect(subject.transform("SELECT * FROM t1")).to eq("SELECT * FROM t1")
9      end
10    end
11
12    context "when there is a HAVING clause" do
13      context "with no NOT query" do
14        it "returns the same query" do
15          expect(subject.transform("SELECT * FROM t1 HAVING a = 1"))
16            .to eq("SELECT * FROM `t1` HAVING `a` = 1")
17        end
18      end
19
20      context "with only a between query" do
21        context "with a > clause" do
22          it "returns the updated query" do
23            expect(subject.transform("SELECT * FROM t1 HAVING NOT a > 1"))
24              .to eq("SELECT * FROM `t1` HAVING `a` <= 1")
25          end
26        end
27
28        context "with a < clause" do
29          it "returns the updated query" do
30            expect(subject.transform("SELECT * FROM t1 HAVING NOT a < 1"))
31              .to eq("SELECT * FROM `t1` HAVING `a` >= 1")
32          end
33        end
34
35        context "with a <= clause" do
36          it "returns the updated query" do
37            expect(subject.transform("SELECT * FROM t1 HAVING NOT a <= 1"))
38              .to eq("SELECT * FROM `t1` HAVING `a` > 1")
39          end
40        end
41
42        context "with a >= clause" do
43          it "returns the updated query" do
44            expect(subject.transform("SELECT * FROM t1 HAVING NOT a >= 1"))
45              .to eq("SELECT * FROM `t1` HAVING `a` < 1")
46          end
47        end
48      end
49
50    context "with a not query and another type of query" do

```



```

51   it "returns the updated query" do
52     expect(subject.transform("SELECT * FROM t1 HAVING NOT a > 1 AND b = 2"))
53       .to eq("SELECT * FROM `t1` HAVING (`a` <= 1 AND `b` = 2)")
54   end
55 end
56
57 context "with a not query and two other type of query" do
58   it "returns the updated query" do
59     expect(subject.transform("SELECT * FROM t1 HAVING NOT a > 1 AND b = 2 AND c = 3"))
60       .to eq("SELECT * FROM `t1` HAVING ((`a` <= 1 AND `b` = 2) AND `c` = 3)")
61   end
62 end
63
64 context "with a not which is not transformable" do
65   it "returns the updated query" do
66     expect(subject.transform("SELECT * FROM t1 HAVING NOT a LIKE 'a'"))
67       .to eq("SELECT * FROM `t1` HAVING `a` NOT LIKE 'a'")
68   end
69 end
70 end
71 end

```

Source code for spec/sql\_assess/transformers/not/from\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Transformers::Not::From do
4    subject { described_class.new(connection) }
5
6    context "with no join" do
7      let(:query) do
8        <<-SQL.squish
9          SELECT `table1`.`id`, `table1`.`id2`
10          FROM `table1`
11        SQL
12      end
13
14      it "doesn't change the query" do
15        expect(subject.transform(query)).to eq(query)
16      end
17    end
18
19    context "with join clause but no not" do
20      let(:query) do
21        <<-SQL.squish
22          SELECT `table1`.`id`, `table1`.`id2`
23          FROM `table1` LEFT JOIN `table2` ON `table2`.`id3` = `table1`.`id1`
24        SQL
25      end
26
27      it "doesn't change the query" do
28        expect(subject.transform(query)).to eq(query)
29      end
30    end
31
32    context "with join clause with not" do
33      let(:query) do
34        <<-SQL.squish
35          SELECT `table1`.`id`, `table1`.`id2`
36          FROM `table1` LEFT JOIN `table2` ON NOT `id3` > `id1`
37          GROUP BY `id1`
38        SQL
39      end
40    end

```

```

41 it "changes the query" do
42   expect(subject.transform(query)).to eq(
43     <<-SQL.squish
44       SELECT `table1`.`id`, `table1`.`id2`
45       FROM `table1` LEFT JOIN `table2` ON `id3` <= `id1`
46       GROUP BY `id1`
47     SQL
48   )
49 end
50 end
51
52 context "with join clause with not" do
53   let(:query) do
54     <<-SQL.squish
55       SELECT `table1`.`id`, `table1`.`id2`
56       FROM `table1` LEFT JOIN `table2` ON NOT `id3` > `id1` AND NOT `id3` >= `id1`
57       GROUP BY `id1`
58     SQL
59   end
60
61   it "changes the query" do
62     expect(subject.transform(query)).to eq(
63       <<-SQL.squish
64         SELECT `table1`.`id`, `table1`.`id2`
65         FROM `table1` LEFT JOIN `table2` ON (`id3` <= `id1` AND `id3` < `id1`)
66         GROUP BY `id1`
67       SQL
68     )
69   end
70 end
71 end

```

Source code for spec/sql\_assess/transformers/all\_columns\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::AllColumns do
4   subject { described_class.new(connection) }
5
6   before do
7     connection.query("CREATE TABLE table1 (id1 integer, id2 integer)")
8     connection.query("CREATE TABLE table2 (id3 integer, id4 integer)")
9   end
10
11   context "for a non-join query" do
12     context "when there is no *" do
13       it "returns the same query" do
14         expect(subject.transform("SELECT id1 FROM table1")).to eq("SELECT `id1` FROM `table1`")
15       end
16
17       it "returns the same query" do
18         expect(subject.transform("SELECT id2 FROM table1")).to eq("SELECT `id2` FROM `table1`")
19       end
20
21       it "returns the same query" do
22         expect(subject.transform("SELECT id1, id2 FROM table1")).to eq("SELECT `id1`, `id2` FROM `table1`")
23       end
24     end
25
26     context "when there is *" do
27       it "returns the query containing all columns in select" do
28         expect(subject.transform("SELECT * FROM table1")).to eq("SELECT `table1`.`id1`, `table1`.`id2` FROM
29           ↳ `table1`")
30       end
31     end
32   end
33 end

```

```

30 end
31 end
32
33 context "for a join query" do
34   context "when there is no *" do
35     it "returns the same query" do
36       expect(subject.transform("SELECT id1 FROM table1, table2")).to eq("SELECT `id1` FROM `table1` CROSS
37         ↳ JOIN `table2`")
38     end
39
40     it "returns the same query" do
41       expect(subject.transform("SELECT id4 FROM table1, table2")).to eq("SELECT `id4` FROM `table1` CROSS
42         ↳ JOIN `table2`")
43     end
44
45     it "returns the same query" do
46       expect(subject.transform("SELECT id1, id2, id3 FROM table1, table2")).to eq("SELECT `id1`, `id2`,
47         ↳ `id3` FROM `table1` CROSS JOIN `table2`")
48     end
49   end
50 end
51
52 context "when there is *" do
53   it "returns the query containing all columns in select" do
54     expect(subject.transform("SELECT * FROM table1, table2"))
55       .to eq("SELECT `table1`.`id1`, `table1`.`id2`, `table2`.`id3`, `table2`.`id4` FROM `table1` CROSS
56         ↳ JOIN `table2`")
57   end
58 end
59 end
60 end

```

Source code for spec/sql\_assess/transformers/comparison\_predicate/where\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::ComparisonPredicate::Where do
4   subject { described_class.new(connection) }
5
6   context "when there is no where clause" do
7     it "returns the same query" do
8       expect(subject.transform("SELECT * FROM table")).to eq("SELECT * FROM table")
9     end
10  end
11
12  context "when there is a where clause" do
13    context "with no comparison predicate query" do
14      it "returns the same query" do
15        expect(subject.transform("SELECT * FROM table WHERE a BETWEEN 1 AND 3"))
16          .to eq("SELECT * FROM `table` WHERE `a` BETWEEN 1 AND 3")
17      end
18    end
19
20    context "with a >" do
21      it "returns the updated query" do
22        expect(subject.transform("SELECT * FROM table WHERE a > 1"))
23          .to eq("SELECT * FROM `table` WHERE 1 < `a`")
24      end
25    end
26
27    context "with a >=" do
28      it "returns the updated query" do
29        expect(subject.transform("SELECT * FROM table WHERE a >= 1"))
30          .to eq("SELECT * FROM `table` WHERE 1 <= `a`")
31      end
32    end
33  end
34 end

```

```

32 end
33
34 context "with a <" do
35   it "returns the updated query" do
36     expect(subject.transform("SELECT * FROM table WHERE a < 1"))
37       .to eq("SELECT * FROM `table` WHERE `a` < 1")
38   end
39 end
40
41 context "with a <=" do
42   it "returns the updated query" do
43     expect(subject.transform("SELECT * FROM table WHERE a <= 1"))
44       .to eq("SELECT * FROM `table` WHERE `a` <= 1")
45   end
46 end
47
48 context "with a <= and a >" do
49   it "returns the updated query" do
50     expect(subject.transform("SELECT * FROM table WHERE a <= 1 AND a > 1"))
51       .to eq("SELECT * FROM `table` WHERE (`a` <= 1 AND 1 < `a`)")
52   end
53 end
54 end
55 end

```

Source code for spec/sql\_assess/transformers/comparison\_predicate/having\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::ComparisonPredicate::Having do
4   subject { described_class.new(connection) }
5
6   context "when there is no having clause" do
7     it "returns the same query" do
8       expect(subject.transform("SELECT * FROM table")).to eq("SELECT * FROM table")
9     end
10  end
11
12  context "when there is a having clause" do
13    context "with no comparison predicate query" do
14      it "returns the same query" do
15        expect(subject.transform("SELECT * FROM table HAVING a BETWEEN 1 AND 3"))
16          .to eq("SELECT * FROM `table` HAVING `a` BETWEEN 1 AND 3")
17      end
18    end
19
20    context "with a >" do
21      it "returns the updated query" do
22        expect(subject.transform("SELECT * FROM table HAVING a > 1"))
23          .to eq("SELECT * FROM `table` HAVING 1 < `a`")
24      end
25    end
26
27    context "with a >=" do
28      it "returns the updated query" do
29        expect(subject.transform("SELECT * FROM table HAVING a >= 1"))
30          .to eq("SELECT * FROM `table` HAVING 1 <= `a`")
31      end
32    end
33
34    context "with a <" do
35      it "returns the updated query" do
36        expect(subject.transform("SELECT * FROM table HAVING a < 1"))
37          .to eq("SELECT * FROM `table` HAVING `a` < 1")

```

```

38   end
39 end
40
41 context "with a <=" do
42   it "returns the updated query" do
43     expect(subject.transform("SELECT * FROM table HAVING a <= 1"))
44       .to eq("SELECT * FROM `table` HAVING `a` <= 1")
45   end
46 end
47
48 context "with a <= and a >" do
49   it "returns the updated query" do
50     expect(subject.transform("SELECT * FROM table HAVING a <= 1 AND a > 1"))
51       .to eq("SELECT * FROM `table` HAVING (`a` <= 1 AND 1 < `a`)")
52   end
53 end
54 end
55 end

```

Source code for spec/sql\_assess/transformers/comparison\_predicate/from\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::ComparisonPredicate::From do
4   subject { described_class.new(connection) }
5
6   context "when there is no join clause" do
7     it "returns the same query" do
8       expect(subject.transform("SELECT * FROM table")).to eq("SELECT * FROM `table`")
9     end
10  end
11
12  context "when there is a join clause" do
13    context "with no comparison predicate query" do
14      it "returns the same query" do
15        expect(subject.transform("SELECT * FROM table LEFT JOIN t1, t3 ON a BETWEEN 1 AND 3"))
16          .to eq("SELECT * FROM `table` LEFT JOIN `t1` CROSS JOIN `t3` ON `a` BETWEEN 1 AND 3")
17      end
18    end
19
20    context "with a >" do
21      it "returns the updated query" do
22        expect(subject.transform("SELECT * FROM table, t3 LEFT JOIN t1 ON a > 1"))
23          .to eq("SELECT * FROM `table` CROSS JOIN `t3` LEFT JOIN `t1` ON 1 < `a`")
24      end
25    end
26
27    context "with a > and a <" do
28      it "returns the updated query" do
29        expect(subject.transform("SELECT * FROM table LEFT JOIN t1 ON a > 1 AND a < 2"))
30          .to eq("SELECT * FROM `table` LEFT JOIN `t1` ON (1 < `a` AND `a` < 2)")
31      end
32    end
33
34    context "with a >=" do
35      it "returns the updated query" do
36        expect(subject.transform("SELECT * FROM table LEFT JOIN t1 ON a >= 1"))
37          .to eq("SELECT * FROM `table` LEFT JOIN `t1` ON 1 <= `a`")
38      end
39    end
40
41    context "with a <" do
42      it "returns the updated query" do
43        expect(subject.transform("SELECT * FROM table LEFT JOIN t1 ON a < 1"))

```

```

44 .to eq("SELECT * FROM `table` LEFT JOIN `t1` ON `a` < 1")
45 end
46 end
47
48 context "with a <=" do
49   it "returns the updated query" do
50     expect(subject.transform("SELECT * FROM table LEFT JOIN t1 ON a <= 1"))
51       .to eq("SELECT * FROM `table` LEFT JOIN `t1` ON `a` <= 1")
52   end
53 end
54 end
55 end

```

Source code for spec/sql\_assess/transformers/between\_prediate/where\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::BetweenPredicate::Where do
4   subject { described_class.new(connection) }
5
6   context "when there is no where clause" do
7     it "returns the same query" do
8       expect(subject.transform("SELECT * FROM table")).to eq("SELECT * FROM table")
9     end
10  end
11
12  context "when there is a where clause" do
13    context "with no between query" do
14      it "returns the same query" do
15        expect(subject.transform("SELECT * FROM table WHERE a = 1"))
16          .to eq("SELECT * FROM `table` WHERE `a` = 1")
17      end
18    end
19
20    context "with only a between query" do
21      it "returns the updated query" do
22        expect(subject.transform("SELECT * FROM table WHERE a BETWEEN 1 and 3"))
23          .to eq("SELECT * FROM `table` WHERE (`a` >= 1 AND `a` <= 3)")
24      end
25    end
26
27    context "with a between query and another type of query" do
28      it "returns the updated query" do
29        expect(subject.transform("SELECT * FROM table WHERE (a BETWEEN 1 and 3) AND b = 2"))
30          .to eq("SELECT * FROM `table` WHERE (((`a` >= 1 AND `a` <= 3) AND `b` = 2)")
31      end
32    end
33
34    context "with a between query and two other type of query" do
35      it "returns the updated query" do
36        expect(subject.transform("SELECT * FROM table WHERE a BETWEEN 1 and 3 AND b = 2 AND c = 3"))
37          .to eq("SELECT * FROM `table` WHERE (((`a` >= 1 AND `a` <= 3) AND `b` = 2) AND `c` = 3)")
38      end
39    end
40  end
41 end

```

Source code for spec/sql\_assess/transformers/between\_prediate/having\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::BetweenPredicate::Having do
4   subject { described_class.new(connection) }
5
6   context "when there is no having clause" do

```

```

7   it "returns the same query" do
8     expect(subject.transform("SELECT * FROM table")).to eq("SELECT * FROM table")
9   end
10 end
11
12 context "when there is a having clause" do
13   context "with no between query" do
14     it "returns the same query" do
15       expect(subject.transform("SELECT * FROM table HAVING a = 1"))
16         .to eq("SELECT * FROM `table` HAVING `a` = 1")
17     end
18   end
19
20   context "with only a between query" do
21     it "returns the updated query" do
22       expect(subject.transform("SELECT * FROM table HAVING a BETWEEN 1 and 3"))
23         .to eq("SELECT * FROM `table` HAVING (`a` >= 1 AND `a` <= 3)")
24     end
25   end
26
27   context "with a between query and another type of query" do
28     it "returns the updated query" do
29       expect(subject.transform("SELECT * FROM table HAVING (a BETWEEN 1 and 3) AND b = 2"))
30         .to eq("SELECT * FROM `table` HAVING ((`a` >= 1 AND `a` <= 3) AND `b` = 2)")
31     end
32   end
33
34   context "with a between query and two other type of query" do
35     it "returns the updated query" do
36       expect(subject.transform("SELECT * FROM table HAVING a BETWEEN 1 and 3 AND b = 2 AND c = 3"))
37         .to eq("SELECT * FROM `table` HAVING (((`a` >= 1 AND `a` <= 3) AND `b` = 2) AND `c` = 3)")
38     end
39   end
40 end
41 end

```

Source code for spec/sql\_assess/transformers/between\_prediate/from\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::BetweenPredicate::From do
4   subject { described_class.new(connection) }
5
6   context "when there is no having clause" do
7     it "returns the same query" do
8       expect(subject.transform("SELECT * FROM table")).to eq("SELECT * FROM `table`")
9     end
10  end
11
12  context "when there is a having clause" do
13    context "with no between query" do
14      it "returns the same query" do
15        expect(subject.transform("SELECT * FROM table, t3 LEFT JOIN t2 ON a = 1"))
16          .to eq("SELECT * FROM `table` CROSS JOIN `t3` LEFT JOIN `t2` ON `a` = 1")
17      end
18    end
19
20    context "with only a between query" do
21      it "returns the updated query" do
22        expect(subject.transform("SELECT * FROM table, t3 LEFT JOIN t2 ON a BETWEEN 1 and 3"))
23          .to eq("SELECT * FROM `table` CROSS JOIN `t3` LEFT JOIN `t2` ON (`a` >= 1 AND `a` <= 3)")
24      end
25    end
26  end

```

```

27 context "with a between query and another type of query" do
28   it "returns the updated query" do
29     expect(subject.transform("SELECT * FROM table LEFT JOIN t2 ON (a BETWEEN 1 and 3) AND b = 2"))
30     .to eq("SELECT * FROM `table` LEFT JOIN `t2` ON ((`a` >= 1 AND `a` <= 3) AND `b` = 2)")
31   end
32 end
33
34 context "with a between query and two other type of query" do
35   it "returns the updated query" do
36     expect(subject.transform("SELECT * FROM table LEFT JOIN t2 ON a BETWEEN 1 and 3 AND b = 2 AND c = 3"))
37     .to eq("SELECT * FROM `table` LEFT JOIN `t2` ON (((`a` >= 1 AND `a` <= 3) AND `b` = 2) AND `c` =
38       ↳ 3)")
39   end
40 end
41 end

```

Source code for spec/sql\_assess/transformers/equivalent\_columns/order\_by\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::EquivalentColumns::OrderBy do
4   subject { described_class.new(connection).transform(sql) }
5
6   context "no equivalence" do
7     context "with no join clause" do
8       let(:sql) do
9         <<-SQL.squish
10          SELECT *
11          FROM
12            `table1`
13          ORDER BY `table1`.`id` ASC
14        SQL
15      end
16
17      it "returns the same query" do
18        expect(subject).to eq(sql)
19      end
20    end
21
22    context "with a join clause but no equivalence" do
23      let(:sql) do
24        <<-SQL.squish
25        SELECT *
26        FROM
27          `table1`
28          LEFT JOIN `table2` ON `table2`.`id` = `table1`.`id2`
29          LEFT JOIN `table3` ON `table3`.`id` = `table1`.`id3`
30          ORDER BY `table1`.`id` ASC
31        SQL
32      end
33
34      it "returns the same query" do
35        expect(subject).to eq(sql)
36      end
37    end
38  end
39
40  context "with an equivalence" do
41    context "with a left join" do
42      let(:sql) do
43        <<-SQL.squish
44        SELECT *
45        FROM

```



```

        `b`
        LEFT JOIN `a` ON `a`.`id` = `b`.`id`
        ORDER BY `b`.`id` ASC
    SQL
end

it "changes to the lowest string" do
  expect(subject).to eq(
    <<-SQL.squish
      SELECT *
      FROM
        `b`
        LEFT JOIN `a` ON `a`.`id` = `b`.`id`
        ORDER BY `a`.`id` ASC
    SQL
  )
end
end

context "with two left joins" do
  let(:sql) do
    <<-SQL.squish
      SELECT *
      FROM
        `c`
        LEFT JOIN `a` ON `a`.`id` = `c`.`id`
        LEFT JOIN `b` ON `b`.`id` = `c`.`id`
        ORDER BY `c`.`id` ASC
    SQL
  end

  it "changes to the lowest string" do
    expect(subject).to eq(
      <<-SQL.squish
        SELECT *
        FROM
          `c`
          LEFT JOIN `a` ON `a`.`id` = `c`.`id`
          LEFT JOIN `b` ON `b`.`id` = `c`.`id`
          ORDER BY `a`.`id` ASC
        SQL
      )
  end
end

context "with two joins" do
  let(:sql) do
    <<-SQL.squish
      SELECT *
      FROM
        `c`
        LEFT JOIN `a` ON `a`.`id` = `c`.`id`
        RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
        ORDER BY `c`.`id` ASC
    SQL
  end

  it "changes to the lowest string" do
    expect(subject).to eq(
      <<-SQL.squish
        SELECT *
        FROM
          `c`

```

```

09         LEFT JOIN `a` ON `a`.`id` = `c`.`id`
10         RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
11     ORDER BY `a`.`id` ASC
12     SQL
13 )
14 end
15 end
16 end
17 end

```

Source code for spec/sql\_assess/transformers/equivalent\_columns/where\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::Transformers::EquivalentColumns::Where do
4    subject { described_class.new(connection).transform(sql) }
5
6    context "no equivalence" do
7      context "with no join clause" do
8        let(:sql) do
9          <<-SQL.squish
10             SELECT *
11             FROM
12               `table1`
13             WHERE `table1`.`id` = 1
14             SQL
15         end
16
17         it "returns the same query" do
18           expect(subject).to eq(sql)
19         end
20       end
21
22       context "with a join clause but no equivalence" do
23         let(:sql) do
24           <<-SQL.squish
25             SELECT *
26             FROM
27               `table1`
28             LEFT JOIN `table2` ON `table2`.`id` = `table1`.`id2`
29             LEFT JOIN `table3` ON `table3`.`id` = `table1`.`id3`
30             WHERE `table1`.`id` = 1
31             SQL
32         end
33
34         it "returns the same query" do
35           expect(subject).to eq(sql)
36         end
37       end
38     end
39
40     context "with an equivalence" do
41       context "with a left join" do
42         let(:sql) do
43           <<-SQL.squish
44             SELECT *
45             FROM
46               `b`
47             LEFT JOIN `a` ON `a`.`id` = `b`.`id`
48             WHERE `b`.`id` = 1
49             SQL
50         end
51
52         it "changes to the lowest string" do

```

```

53 expect(subject).to eq(
54   <<-SQL.squish
55     SELECT *
56     FROM
57       `b`
58     LEFT JOIN `a` ON `a`.`id` = `b`.`id`
59     WHERE `a`.`id` = 1
60   SQL
61 )
62 end
63 end
64
65 context "with two left joins" do
66   let(:sql) do
67     <<-SQL.squish
68       SELECT *
69       FROM
70         `c`
71       LEFT JOIN `a` ON `a`.`id` = `c`.`id`
72       LEFT JOIN `b` ON `b`.`id` = `c`.`id`
73       WHERE `c`.`id` = 1
74     SQL
75   end
76
77   it "changes to the lowest string" do
78     expect(subject).to eq(
79       <<-SQL.squish
80         SELECT *
81         FROM
82           `c`
83         LEFT JOIN `a` ON `a`.`id` = `c`.`id`
84         LEFT JOIN `b` ON `b`.`id` = `c`.`id`
85         WHERE `a`.`id` = 1
86       SQL
87     )
88   end
89 end
90
91 context "with two joins" do
92   let(:sql) do
93     <<-SQL.squish
94       SELECT *
95       FROM
96         `c`
97       LEFT JOIN `a` ON `a`.`id` = `c`.`id`
98       RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
99       WHERE `c`.`id` = 1
100     SQL
101   end
102
103   it "changes to the lowest string" do
104     expect(subject).to eq(
105       <<-SQL.squish
106         SELECT *
107         FROM
108           `c`
109         LEFT JOIN `a` ON `a`.`id` = `c`.`id`
110         RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
111         WHERE `a`.`id` = 1
112       SQL
113     )
114   end
115 end

```

```

16   end
17 end

Source code for spec/sql_assess/transformers/equivalent_columns/group_by_spec.rb

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::EquivalentColumns::Group do
4   subject { described_class.new(connection).transform(sql) }
5
6   context "no equivalence" do
7     context "with no join clause" do
8       let(:sql) do
9         <<-SQL.squish
10          SELECT *
11          FROM
12            `table1`
13          GROUP BY `table1`.`id`
14        SQL
15      end
16
17      it "returns the same query" do
18        expect(subject).to eq(sql)
19      end
20    end
21
22    context "with a join clause but no equivalence" do
23      let(:sql) do
24        <<-SQL.squish
25        SELECT *
26        FROM
27          `table1`
28          LEFT JOIN `table2` ON `table2`.`id` = `table1`.`id2`
29          LEFT JOIN `table3` ON `table3`.`id` = `table1`.`id3`
30        GROUP BY `table1`.`id`
31      SQL
32    end
33
34    it "returns the same query" do
35      expect(subject).to eq(sql)
36    end
37  end
38 end
39
40 context "with an equivalence" do
41   context "with a left join" do
42     let(:sql) do
43       <<-SQL.squish
44       SELECT *
45       FROM
46         `b`
47         LEFT JOIN `a` ON `a`.`id` = `b`.`id`
48       GROUP BY `b`.`id`
49     SQL
50   end
51
52   it "changes to the lowest string" do
53     expect(subject).to eq(
54       <<-SQL.squish
55       SELECT *
56       FROM
57         `b`
58         LEFT JOIN `a` ON `a`.`id` = `b`.`id`
59       GROUP BY `a`.`id`

```

```

60     SQL
61   )
62   end
63 end
64
65 context "with two left joins" do
66   let(:sql) do
67     <<-SQL.squish
68     SELECT *
69     FROM
70       `c`
71     LEFT JOIN `a` ON `a`.`id` = `c`.`id`
72     LEFT JOIN `b` ON `b`.`id` = `c`.`id`
73     GROUP BY `c`.`id`
74     SQL
75   end
76
77   it "changes to the lowest string" do
78     expect(subject).to eq(
79       <<-SQL.squish
80       SELECT *
81       FROM
82         `c`
83       LEFT JOIN `a` ON `a`.`id` = `c`.`id`
84       LEFT JOIN `b` ON `b`.`id` = `c`.`id`
85       GROUP BY `a`.`id`
86       SQL
87     )
88   end
89 end
90
91 context "with two joins" do
92   let(:sql) do
93     <<-SQL.squish
94     SELECT *
95     FROM
96       `c`
97     LEFT JOIN `a` ON `a`.`id` = `c`.`id`
98     RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
99     GROUP BY `c`.`id`
100    SQL
101  end
102
103  it "changes to the lowest string" do
104    expect(subject).to eq(
105      <<-SQL.squish
106      SELECT *
107      FROM
108        `c`
109      LEFT JOIN `a` ON `a`.`id` = `c`.`id`
110      RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
111      GROUP BY `a`.`id`
112      SQL
113    )
114  end
115 end
116 end
117 end

```

Source code for spec/sql\_assess/transformers/equivalent\_columns/having\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::EquivalentColumns::Having do

```

```

4 subject { described_class.new(connection).transform(sql) }
5
6 context "no equivalence" do
7   context "with no join clause" do
8     let(:sql) do
9       <<-SQL.squish
10        SELECT *
11        FROM
12          `table1`
13        HAVING `table1`.`id` = 1
14      SQL
15    end
16
17    it "returns the same query" do
18      expect(subject).to eq(sql)
19    end
20  end
21
22  context "with a join clause but no equivalence" do
23    let(:sql) do
24      <<-SQL.squish
25      SELECT *
26      FROM
27        `table1`
28      LEFT JOIN `table2` ON `table2`.`id` = `table1`.`id2`
29      LEFT JOIN `table3` ON `table3`.`id` = `table1`.`id3`
30      HAVING `table1`.`id` = 1
31    SQL
32    end
33
34    it "returns the same query" do
35      expect(subject).to eq(sql)
36    end
37  end
38 end
39
40 context "with an equivalence" do
41   context "with a left join" do
42     let(:sql) do
43       <<-SQL.squish
44       SELECT *
45       FROM
46         `b`
47       LEFT JOIN `a` ON `a`.`id` = `b`.`id`
48       HAVING `b`.`id` = 1
49     SQL
50   end
51
52   it "changes to the lowest string" do
53     expect(subject).to eq(
54       <<-SQL.squish
55       SELECT *
56       FROM
57         `b`
58       LEFT JOIN `a` ON `a`.`id` = `b`.`id`
59       HAVING `a`.`id` = 1
60     SQL
61   )
62 end
63
64 context "with two left joins" do
65   let(:sql) do

```

```

67 <<-SQL.squish
68 SELECT *
69 FROM
70   `c`
71   LEFT JOIN `a` ON `a`.`id` = `c`.`id`
72   LEFT JOIN `b` ON `b`.`id` = `c`.`id`
73   HAVING `c`.`id` = 1
74 SQL
75 end
76
77 it "changes to the lowest string" do
78   expect(subject).to eq(
79     <<-SQL.squish
80     SELECT *
81     FROM
82       `c`
83       LEFT JOIN `a` ON `a`.`id` = `c`.`id`
84       LEFT JOIN `b` ON `b`.`id` = `c`.`id`
85       HAVING `a`.`id` = 1
86     SQL
87   )
88 end
89 end
90
91 context "with two joins" do
92   let(:sql) do
93     <<-SQL.squish
94     SELECT *
95     FROM
96       `c`
97       LEFT JOIN `a` ON `a`.`id` = `c`.`id`
98       RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
99       HAVING `c`.`id` = 1
100    SQL
101  end
102
103  it "changes to the lowest string" do
104    expect(subject).to eq(
105      <<-SQL.squish
106      SELECT *
107      FROM
108        `c`
109        LEFT JOIN `a` ON `a`.`id` = `c`.`id`
110        RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
111        HAVING `a`.`id` = 1
112      SQL
113    )
114  end
115 end
116 end
117 end

```

Source code for spec/sql\_assess/transformers/equivalent\_columns/select\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::EquivalentColumns::Select do
4   subject { described_class.new(connection).transform(sql) }
5
6   context "no equivalence" do
7     context "with no join clause" do
8       let(:sql) do
9         <<-SQL.squish
10        SELECT `table1`.`id`

```

```

11         FROM
12         `table1`
13     SQL
14 end
15
16 it "returns the same query" do
17     expect(subject).to eq(sql)
18 end
19 end
20
21 context "with a join clause but no equivalence" do
22     let(:sql) do
23         <<-SQL.squish
24         SELECT `table1`.`id`
25         FROM
26             `table1`
27         LEFT JOIN `table2` ON `table2`.`id` = `table1`.`id2`
28         LEFT JOIN `table3` ON `table3`.`id` = `table1`.`id3`
29     SQL
30 end
31
32 it "returns the same query" do
33     expect(subject).to eq(sql)
34 end
35 end
36
37 context "with a join clause but no equivalence" do
38     let(:sql) do
39         <<-SQL.squish
40         SELECT `a`.`id`
41         FROM
42             `b`
43         LEFT JOIN `a` ON (`a`.`id` = `b`.`id` OR `a`.`id` = `b`.`id2`)
44     SQL
45 end
46
47 it "returns the same query" do
48     expect(subject).to eq(sql)
49 end
50 end
51 end
52
53 context "with an equivalence" do
54     context "with a left join" do
55         let(:sql) do
56             <<-SQL.squish
57             SELECT `b`.`id`
58             FROM
59                 `b`
60             LEFT JOIN `a` ON (`a`.`id` = `b`.`id` AND `a`.`id2` = `b`.`id2`)
61         SQL
62     end
63
64     it "changes to the lowest string" do
65         expect(subject).to eq(
66             <<-SQL.squish
67             SELECT `a`.`id`
68             FROM
69                 `b`
70             LEFT JOIN `a` ON (`a`.`id` = `b`.`id` AND `a`.`id2` = `b`.`id2`)
71             SQL
72         )
73     end

```



```

74 end
75
76 context "with two left joins" do
77   let(:sql) do
78     <<-SQL.squish
79     SELECT `c`.`id`
80     FROM
81       `c`
82     LEFT JOIN `a` ON `a`.`id` = `c`.`id`
83     LEFT JOIN `b` ON `b`.`id` = `c`.`id`
84   SQL
85 end
86
87 it "changes to the lowest string" do
88   expect(subject).to eq(
89     <<-SQL.squish
90     SELECT `a`.`id`
91     FROM
92       `c`
93     LEFT JOIN `a` ON `a`.`id` = `c`.`id`
94     LEFT JOIN `b` ON `b`.`id` = `c`.`id`
95   SQL
96 )
97 end
98 end
99
100 context "with two left joins" do
101   let(:sql) do
102     <<-SQL.squish
103     SELECT `c`.`id`
104     FROM
105       `c`
106     LEFT JOIN `a` ON `a`.`id` = `c`.`id`
107     RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
108   SQL
109 end
110
111 it "changes to the lowest string" do
112   expect(subject).to eq(
113     <<-SQL.squish
114     SELECT `a`.`id`
115     FROM
116       `c`
117     LEFT JOIN `a` ON `a`.`id` = `c`.`id`
118     RIGHT JOIN `b` ON `b`.`id` = `c`.`id`
119   SQL
120 )
121 end
122 end
123 end
124 end

```

Source code for spec/sql\_assess/transformers/base\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::Transformers::Base do
4   subject { described_class.new(@connection) }
5
6   context "#transform" do
7     it "throws an error" do
8       expect { subject.transform }.to raise_error('Implement this method in subclass')
9     end
10  end

```

```

11 context "#tables" do
12   context "one table" do
13     let(:query) { "SELECT * from t1" }
14
15     it "returns the table" do
16       expect(subject.tables(query)).to eq(["t1"])
17     end
18   end
19 end
20
21 context "two tables" do
22   let(:query) { "SELECT * from t1, t2" }
23
24   it "returns the table" do
25     expect(subject.tables(query)).to eq(["t1", "t2"])
26   end
27 end
28
29 context "three tables" do
30   let(:query) { "SELECT * from t1, t2 LEFT JOIN t3 on t1.id = t3.id" }
31
32   it "returns the table" do
33     expect(subject.tables(query)).to eq(["t1", "t2", "t3"])
34   end
35 end
36 end
37 end

```

Source code for spec/sql\_assess/database\_connection\_spec.rb

```

1 require "spec_helper"
2
3 RSpec.describe SqlAssess::DatabaseConnection do
4   let(:do_not_delete_database) { false }
5   around do |example|
6     Timecop.freeze(Time.local(1990)) do
7       example.run
8       subject.delete_database unless do_not_delete_database
9     end
10  end
11
12  describe "#initialize" do
13    context "when the user is invalid" do
14      let(:do_not_delete_database) { true }
15      it "throws an error" do
16        expect { described_class.new(username: "test") }.to raise_error(SqlAssess::DatabaseConnectionError)
17      end
18    end
19
20    context "when everything is valid" do
21      it "doesn't throw an error" do
22        expect { subject }.to_not raise_error
23      end
24    end
25  end
26
27  describe "#database_name" do
28    context "with no existing database" do
29      it "uses the correct name" do
30        subject
31        expect(subject.query("SELECT DATABASE();").first["DATABASE()"]).to eq("000000")
32      end
33    end
34  end

```

```

35 context "with an existing database" do
36   it "creates a database with attempt in it" do
37     existing_connection = described_class.new
38     expect(existing_connection.query("SELECT DATABASE();").first["DATABASE()"]).to eq("000000")
39
40     subject
41
42     expect(subject.query("SELECT DATABASE();").first["DATABASE()"]).to eq("000000_1")
43
44     existing_connection.delete_database
45   end
46 end
47
48 describe "#query" do
49   it "runs the query" do
50     expect(subject.query("SHOW tables").count).to eq(0)
51   end
52
53   context "when trying to create another database" do
54     it "throws an error" do
55       expect { subject.query("CREATE DATABASE TEST") }.to raise_error(MySql2::Error)
56     end
57   end
58 end
59
60 describe "#multi_query" do
61   it "runs the query" do
62     result = subject.multiple_query("SELECT 1; SELECT 2; SELECT 3")
63     expect(result.count).to eq(3)
64     expect(result.map(&:first)).to eq([{"1" => 1 }, {"2" => 2 }, {"3" => 3 }])
65   end
66
67   context "when trying to create another database" do
68     it "throws an error" do
69       expect { subject.multiple_query("CREATE DATABASE TEST") }.to raise_error(MySql2::Error)
70     end
71   end
72 end
73
74 describe "#delete_database" do
75   context "when using default table name" do
76     let(:do_not_delete_database) { true }
77
78     it "deletes the database" do
79       subject
80
81       expect(subject.query("SELECT DATABASE();").first["DATABASE()"]).to eq("000000")
82
83       subject.delete_database
84
85       expect(subject.query("SHOW DATABASES;").map { |r| r["Database"] }).to_not include("000000")
86     end
87   end
88
89   context "when passing default database" do
90     subject { described_class.new(database: "local_db") }
91     it "leaves FOREIGN_KEY_CHECKS set to ON" do
92       subject.delete_database
93
94       expect(subject.query("SHOW Variables WHERE Variable_name='foreign_key_checks';").first["Value"])
95         .to eq("ON")
96     end
97   end

```

```

98 context "when there are no existing tables" do
99   it "doesn't throw an error" do
100     expect { subject.delete_database }.to_not raise_error
101
102     tables = connection.query('SHOW tables;')
103
104     expect(tables.count).to eq(0)
105   end
106 end
107
108 context "when there are existing tables with data" do
109   context "without foreign keys" do
110     before do
111       subject.query('CREATE TABLE table1 (id integer);')
112       subject.query('CREATE TABLE table2 (id integer);')
113       subject.query('INSERT INTO table1 (id) values(1);')
114       subject.query('INSERT INTO table2 (id) values(1);')
115
116       tables = subject.query("SHOW tables;")
117     end
118
119     it "drops all tables" do
120       subject.delete_database
121
122       tables = subject.query('SHOW tables;')
123
124       expect(tables.count).to eq(0)
125     end
126
127     it "leaves FOREIGN_KEY_CHECKS set to ON" do
128       subject.delete_database
129
130       expect(subject.query("SHOW Variables WHERE Variable_name='foreign_key_checks';").first["Value"])
131         .to eq("ON")
132     end
133   end
134 end
135
136 context "with foreign keys" do
137   before do
138     subject.query('
139       CREATE TABLE table1 (
140         id integer,
141         PRIMARY KEY (id)
142       );
143   ')
144     subject.query('
145       CREATE TABLE table2 (
146         table1_id integer,
147         FOREIGN KEY(table1_id) REFERENCES table1(id)
148       );
149   ')
150
151     subject.query('INSERT INTO table1 (id) values(1);')
152     subject.query('INSERT INTO table2 (table1_id) values(1);')
153
154     tables = subject.query("SHOW tables;")
155   end
156
157   it "drops all tables" do
158     subject.delete_database
159
160     tables = subject.query('SHOW tables;')

```

```

61         expect(tables.count).to eq(0)
62     end
63
64     it "leaves FOREIGN_KEY_CHECKS set to ON" do
65         subject.delete_database
66
67         expect(subject.query("SHOW Variables WHERE Variable_name='foreign_key_checks';").first["Value"])
68             .to eq("ON")
69     end
70 end
71 end
72 end
73 end
74 end
75 end

```

Source code for spec/sql\_assess/data\_extractor\_spec.rb

```

1  require "spec_helper"
2
3  RSpec.describe SqlAssess::DataExtractor do
4      subject { described_class.new(connection) }
5
6      context "with a single table" do
7          before do
8              connection.query('CREATE TABLE table1 (id integer);')
9              connection.query('CREATE TABLE table2 (id integer);')
10             connection.query('INSERT INTO table1 (id) values(1);')
11             connection.query('INSERT INTO table1 (id) values(2);')
12         end
13
14         it "returns the correct answer" do
15             expect(subject.run).to eq([
16                 {
17                     name: "table1",
18                     columns: [{ name: "id", type: "int(11)" }],
19                     data: [{ "id" => 1 }, { "id" => 2 }]
20                 },
21                 {
22                     name: "table2",
23                     columns: [{ name: "id", type: "int(11)" }],
24                     data: []
25                 },
26             ])
27         end
28     end
29 end

```

Source code for lib/sql\_assess.rb

```

1  # frozen_string_literal: true
2
3  require 'sql_assess/version'
4  require 'sql_assess/error'
5  require 'sql_assess/assessor'
6  require 'active_support/all'
7
8  # The namespace of the library. The public interface is provided by #{Assessor}
9  module SqlAssess; end

```

Source code for lib/sql\_assess/grader/base.rb

```

1  # frozen_string_literal: true
2
3  require 'rubygems/text'

```

```

4 module SqlAssess
5   # Namespace that handles the grading part of the library
6   module Grader
7     # Base class for the grader
8     # @author Vlad Stoica
9     class Base
10      # Returns the grade for a certain attribute given a list of attributes
11      # @param [String] attribute component name (e.g. columns)
12      # @param [Hash] student_attributes student's attributes for that component
13      # @param [Hash] instructor_attributes instructor's attributes for that component
14      def self.grade_for(attribute:, student_attributes:, instructor_attributes:)
15        "SqlAssess::Grader::#{attribute.to_s.camelize}".constantize.new(
16          student_attributes: student_attributes,
17          instructor_attributes: instructor_attributes
18        ).rounded_grade
19      end
20    end
21
22    def initialize(student_attributes:, instructor_attributes:)
23      @student_attributes = student_attributes
24      @instructor_attributes = instructor_attributes
25    end
26
27    # The levenshtein distance between two strings
28    # @param [String] string1
29    # @param [String] string2
30    # @return [Integer] the distance
31    def levenshtein_distance(string1, string2)
32      ld = Class.new.extend(Gem::Text).method(:levenshtein_distance)
33
34      ld.call(string1, string2)
35    end
36
37    # Rounds the grade to two decimals. The subclasses must implement the
38    # grade method.
39    # @return [Double] rounded grade to two decimals
40    def rounded_grade
41      grade.round(2)
42    end
43
44    private
45
46    def grade_for_array(instructor_attributes = @instructor_attributes, student_attributes =
47      ↳ @student_attributes)
48      max_grade = (student_attributes.length + instructor_attributes.length).to_d
49      return 1 if max_grade.zero?
50
51      instructor_unmatched_attributes = instructor_attributes.dup
52      student_unmatched_attributes = student_attributes.dup
53
54      student_unmatched_attributes = student_unmatched_attributes.keep_if do |student_unmatched_attribute|
55        next 0 if instructor_unmatched_attributes.empty?
56
57        match_score = instructor_unmatched_attributes.map do |instructor_unmatched_attribute|
58          match_score(student_unmatched_attribute, instructor_unmatched_attribute)
59        end
60
61        best_match_score = match_score.each_with_index.max
62
63        if best_match_score[0] == 1
64          instructor_unmatched_attributes.delete_at(best_match_score[1])
65          false
66        else

```

```

66     true
67   end
68 end
69
70 matched_attributes = array_difference(
71   student_attributes,
72   student_unmatched_attributes
73 )
74
75 matched_grade = matched_attributes.length * 2.0
76
77 unmatched_grade = student_unmatched_attributes.sum do |student_unmatched_attribute|
78   next 0 if instructor_unmatched_attributes.empty?
79
80   match_score = instructor_unmatched_attributes.map do |instructor_unmatched_attribute|
81     match_score(student_unmatched_attribute, instructor_unmatched_attribute)
82   end
83
84   best_match_score = match_score.each_with_index.max
85
86   if best_match_score[0].positive?
87     instructor_unmatched_attributes.delete_at(best_match_score[1])
88   end
89
90   best_match_score[0]
91 end
92
93 (matched_grade + unmatched_grade) / max_grade
94 end
95
96 # Difference with removing only once solution obtained from
97 # https://stackoverflow.com/questions/30429659/ruby-difference-in-array-including-duplicates
98 def array_difference(array1, array2)
99   array1 = array1.dup
100   array2.each { |del| array1.slice!(array1.index(del)) if array1.include?(del) }
101   array1
102 end
103 end
104 end
105 end
106
107 require_relative 'columns'
108 require_relative 'order_by'
109 require_relative 'where'
110 require_relative 'distinct_filter'
111 require_relative 'limit'
112 require_relative 'tables'
113 require_relative 'group'
114 require_relative 'having'

```

Source code for lib/sql\_assess/grader/having.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Grader
5     # Grader for HAVING clause
6     # @author Vlad Stoica
7     class Having < Base
8       def initialize(student_attributes:, instructor_attributes:)
9         @student_having = student_attributes
10        @instructor_having = instructor_attributes
11      end
12

```

```

13 private
14
15 def grade
16     return 1 if @student_having == @instructor_having
17
18     return 0 if @student_having == {} || @instructor_having == {}
19
20     # Partial grading
21
22     student_leaves = [get_leaves(@student_having)].flatten
23     instructor_leaves = [get_leaves(@instructor_having)].flatten
24
25     conditions_grade = grade_for_array(student_leaves, instructor_leaves)
26
27     internal_nodes = internal_count(@student_having) + internal_count(@instructor_having)
28
29     if internal_nodes.positive?
30         tree_grade = grade_for_tree(@student_having, @instructor_having).to_d / internal_nodes
31         (conditions_grade + tree_grade) / 2
32     else
33         conditions_grade
34     end
35 end
36
37 def grade_for_tree(student_tree, instructor_tree)
38     if student_tree && student_tree[:is_inner] && instructor_tree && instructor_tree[:is_inner]
39         current_grade = grade_for_node(student_tree, instructor_tree)
40
41         child_node_grade_as_normal = grade_for_tree(student_tree[:left_clause],
42             ↪ instructor_tree[:left_clause]) +
43             grade_for_tree(student_tree[:right_clause],
44             ↪ instructor_tree[:right_clause])
45
46         child_node_grade_as_reversed = grade_for_tree(student_tree[:left_clause],
47             ↪ instructor_tree[:right_clause]) +
48             grade_for_tree(student_tree[:right_clause],
49             ↪ instructor_tree[:left_clause])
50
51         child_grade = [
52             child_node_grade_as_normal,
53             child_node_grade_as_reversed,
54         ].max
55
56         current_grade + child_grade
57     else
58         0
59     end
60 end
61
62 def internal_count(having_clause)
63     if having_clause && having_clause[:is_inner]
64         1 + internal_count(having_clause[:left_clause]) + internal_count(having_clause[:right_clause])
65     else
66         0
67     end
68 end
69
70 def grade_for_node(student_tree, instructor_tree)
71     if student_tree[:type] == instructor_tree[:type]
72         2
73     else
74         0
75     end
76 end

```



```

72 end
73
74 def get_leaves(having_clause)
75   if having_clause.nil?
76     nil
77   elsif having_clause[:is_inner] == false
78     having_clause
79   else
80     [
81       get_leaves(having_clause[:left_clause]),
82       get_leaves(having_clause[:right_clause]),
83     ].flatten
84   end
85 end
86
87 def match_score(having_clause1, having_clause2)
88   if having_clause1 == having_clause2
89     2
90   else
91     0
92   end
93 end
94 end
95 end
96 end

```

Source code for lib/sql\_assess/grader/distinct\_filter.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Grader
5      # Grader for distinct filter
6      # @author Vlad Stoica
7      class DistinctFilter < Base
8        def initialize(student_attributes:, instructor_attributes:)
9          @student_distinct = student_attributes
10         @instructor_distinct = instructor_attributes
11       end
12
13       private
14
15       def grade
16         if @student_distinct == @instructor_distinct
17           1.0
18         elsif @student_distinct == 'DISTINCT' && @instructor_distinct == 'DISTINCTROW'
19           0.5
20         elsif @student_distinct == 'DISTINCTROW' && @instructor_distinct == 'DISTINCT'
21           0.5
22         else
23           0
24         end
25       end
26     end
27   end
28 end

```

Source code for lib/sql\_assess/grader/tables.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Grader
5      # Grader for FROM clause
6      # @author Vlad Stoica

```

```

7 class Tables < Base
8   private
9
10  def grade
11    if @instructor_attributes.length == 1 && @student_attributes.length == 1
12      compare_base_grade
13    else
14      joins_grade = grade_for_array(
15        @instructor_attributes.drop(1),
16        @student_attributes.drop(1)
17      )
18
19      (joins_grade + compare_base_grade) / 2
20    end
21  end
22
23  def compare_base_grade
24    instructor_condition = @instructor_attributes.first
25    student_condition = @student_attributes.first
26
27    if instructor_condition == student_condition
28      1
29    else
30      0
31    end
32  end
33
34  def match_score(instructor_join, student_expressions)
35    if instructor_join == student_expressions
36      1
37    elsif instructor_join[:table] == student_expressions[:table]
38      if instructor_join[:join_type] == student_expressions[:join_type]
39        0.75
40      elsif instructor_join[:condition] == student_expressions[:condition]
41        0.75
42      else
43        0.5
44      end
45    else
46      0
47    end
48  end
49 end
50 end
51 end

```

Source code for lib/sql\_assess/grader/group.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Grader
5     # Grader for GROUP clause
6     # @author Vlad Stoica
7     class Group < Base
8       private
9
10      def grade
11        grade_for_array
12      end
13
14      def match_score(column1, column2)
15        table_name1, column_name1 = column1.split('.')
16        table_name2, column_name2 = column2.split('.')

```

```

17         if table_name1 == table_name2
18             1.0 / (levenshtein_distance(column_name1, column_name2) + 1)
19         else
20             0
21         end
22     end
23 end
24 end
25 end
26 end

```

Source code for lib/sql\_assess/grader/columns.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4     module Grader
5         # Grader for columns
6         # @author Vlad Stoica
7         class Columns < Base
8             private
9
10            def grade
11                grade_for_array
12            end
13
14            def match_score(column1, column2)
15                table_name1, column_name1 = column1.split('.')
16                table_name2, column_name2 = column2.split('.')
17
18                if table_name1 == table_name2
19                    1.0 / (levenshtein_distance(column_name1, column_name2) + 1)
20                else
21                    0
22                end
23            end
24        end
25    end
26 end

```

Source code for lib/sql\_assess/grader/order\_by.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4     module Grader
5         # Grader for ORDER BY clause
6         # @author Vlad Stoica
7         class OrderBy < Base
8             private
9
10            def grade
11                grade_for_array
12            end
13
14            def match_score(order_by1, order_by2)
15                column1, order1 = order_by1[:column].split(' ')
16                column2, order2 = order_by2[:column].split(' ')
17                position_difference = (order_by1[:position] - order_by2[:position]).abs + 1
18
19                if column1 == column2
20                    if order1 == order2
21                        1.0 / position_difference
22                    else
23                        0.5 / position_difference

```

```

24     end
25   else
26     0
27   end
28 end
29 end
30 end
31 end

```

Source code for lib/sql\_assess/grader/limit.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Grader
5     # Grader for LIMIT clause
6     # @author Vlad Stoica
7     class Limit < Base
8       def initialize(student_attributes:, instructor_attributes:)
9         @student_limit = student_attributes
10        @instructor_limit = instructor_attributes
11      end
12
13      private
14
15      def grade
16        grade = 0
17
18        grade += 0.5 if @student_limit[:limit] == @instructor_limit[:limit]
19
20        grade += 0.5 if @student_limit[:offset] == @instructor_limit[:offset]
21
22        grade
23      end
24    end
25  end
26 end

```

Source code for lib/sql\_assess/grader/where.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Grader
5     # Grader for WHERE clause
6     # @author Vlad Stoica
7     class Where < Base
8       def initialize(student_attributes:, instructor_attributes:)
9         @student_where = student_attributes
10        @instructor_where = instructor_attributes
11      end
12
13      private
14
15      def grade
16        return 1 if @student_where == @instructor_where
17
18        return 0 if @student_where == {} || @instructor_where == {}
19
20        # Partial grading
21
22        student_leaves = [get_leaves(@student_where)].flatten
23        instructor_leaves = [get_leaves(@instructor_where)].flatten
24
25        conditions_grade = grade_for_array(student_leaves, instructor_leaves)

```

```

26 internal_nodes = internal_count(@student_where) + internal_count(@instructor_where)
27
28
29 if internal_nodes.positive?
30   tree_grade = grade_for_tree(@student_where, @instructor_where).to_d / internal_nodes
31   (conditions_grade + tree_grade) / 2
32 else
33   conditions_grade
34 end
35 end
36
37 def grade_for_tree(student_tree, instructor_tree)
38   if student_tree && student_tree[:is_inner] && instructor_tree && instructor_tree[:is_inner]
39     current_grade = grade_for_node(student_tree, instructor_tree)
40
41     child_node_grade_as_normal = grade_for_tree(student_tree[:left_clause],
42   ↪   instructor_tree[:left_clause]) +
43       grade_for_tree(student_tree[:right_clause],
44   ↪   instructor_tree[:right_clause])
45
46     child_node_grade_as_reversed = grade_for_tree(student_tree[:left_clause],
47   ↪   instructor_tree[:right_clause]) +
48       grade_for_tree(student_tree[:right_clause],
49   ↪   instructor_tree[:left_clause])
50
51     child_grade = [
52       child_node_grade_as_normal,
53       child_node_grade_as_reversed,
54     ].max
55
56     current_grade + child_grade
57   else
58     0
59   end
60 end
61
62 def internal_count(where_clause)
63   if where_clause && where_clause[:is_inner]
64     1 + internal_count(where_clause[:left_clause]) + internal_count(where_clause[:right_clause])
65   else
66     0
67   end
68 end
69
70 def grade_for_node(student_tree, instructor_tree)
71   if student_tree[:type] == instructor_tree[:type]
72     2
73   else
74     0
75   end
76 end
77
78 def get_leaves(node)
79   if node.nil?
80     nil
81   elsif node[:is_inner] == false
82     node
83   else
84     [
85       get_leaves(node[:left_clause]),
86       get_leaves(node[:right_clause]),
87     ].flatten
88   end
89 end

```

```

85     end
86
87     def match_score(where_clause1, where_clause2)
88         if where_clause1 == where_clause2
89             2
90         else
91             0
92         end
93     end
94 end
95 end
96 end

```

Source code for lib/sql\_assess/query\_attribute\_extractor.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      # Class for handling the attribute extraction process
5      # @author Vlad Stoica
6      class QueryAttributeExtractor
7          # Extract the attributes of both the instructor's and student's queries
8          #
9          # @param [String] instructor_sql_query
10         # @param [String] student_sql_query
11         # @return [Hash] with two keys student and instructor. Each value has the format
12         #   returned by {#extract_query}
13         def extract(instructor_sql_query, student_sql_query)
14             {
15                 student: extract_query(student_sql_query),
16                 instructor: extract_query(instructor_sql_query),
17             }
18         end
19
20         # Extract the attributes of a query
21         # @param [String] query
22         # @return [Hash] that contains all attributes of a query.
23         def extract_query(query)
24             {
25                 columns: Parsers::Columns.new(query).columns,
26                 order_by: Parsers::OrderBy.new(query).order,
27                 where: Parsers::Where.new(query).where,
28                 where_tree: Parsers::Where.new(query).where_tree,
29                 tables: Parsers::Tables.new(query).tables,
30                 distinct_filter: Parsers::DistinctFilter.new(query).distinct_filter,
31                 limit: Parsers::Limit.new(query).limit,
32                 group: Parsers::Group.new(query).group,
33                 having: Parsers::Having.new(query).having,
34                 having_tree: Parsers::Having.new(query).having_tree,
35             }
36         end
37     end
38 end

```

Source code for lib/sql\_assess/parsers/base.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      # Namespace that handles the components extraction
5      module Parsers
6          # Base class for the parsers
7          # @author Vlad Stoica
8          class Base
9              def initialize(query)

```

```

10     @parsed_query = SQLParser::Parser.new.scan_str(query)
11   end
12 end
13 end
14 end

```

```

16 require_relative 'columns'
17 require_relative 'order_by'
18 require_relative 'where'
19 require_relative 'tables'
20 require_relative 'distinct_filter'
21 require_relative 'limit'
22 require_relative 'group'
23 require_relative 'having'

```

Source code for lib/sql\_assess/parsers/having.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Parsers
5      # @author Vlad Stoica
6      # Parser for the HAVING clause
7      class Having < Base
8        # @return [Hash] the binary expression tree of the HAVING clause
9        def having
10          if @parsed_query.query_expression.table_expression.having_clause.nil?
11            {}
12          else
13            self.class.transform(@parsed_query.query_expression.table_expression.having_clause.search_condition)
14          end
15        end
16
17        # @return [Hash] the expression tree (not binary tree) of the HAVING clause
18        def having_tree
19          if @parsed_query.query_expression.table_expression.having_clause.nil?
20            {}
21          else
22            transform_tree(
23              @parsed_query.query_expression.table_expression.having_clause.search_condition
24            )
25          end
26        end
27
28        # Transform a clause to a tree
29        # @param [SQLParser::Statement] clause current node
30        # @return [Hash] tree version the clause
31        def self.transform(clause)
32          if clause.is_a?(SQLParser::Statement::ComparisonPredicate)
33            {
34              type: clause.class.name.split('::').last.underscore.humanize.upcase,
35              left: clause.left.to_sql,
36              right: clause.right.to_sql,
37              sql: clause.to_sql,
38            }
39          elsif clause.is_a?(SQLParser::Statement::SearchCondition)
40            type = clause.class.name.split('::').last.underscore.humanize.upcase
41            transform_left = merge(type, transform(clause.left))
42            transform_right = merge(type, transform(clause.right))
43            {
44              type: type,
45              clauses: [
46                transform_left,
47                transform_right,

```

```

48     ].flatten,
49   }
50 end
51 end
52
53 def self.merge(type, clause)
54   if clause[:type] == type
55     clause[:clauses]
56   else
57     clause
58   end
59 end
60 private_class_method :merge
61
62 private
63
64 def transform_tree(clause)
65   if clause.is_a?(SQLParser::Statement::ComparisonPredicate)
66     {
67       is_inner: false,
68       type: clause.class.name.split('::').last.underscore.humanize.upcase,
69       left: clause.left.to_sql,
70       right: clause.right.to_sql,
71       sql: clause.to_sql,
72     }
73   elsif clause.is_a?(SQLParser::Statement::SearchCondition)
74     type = clause.class.name.split('::').last.underscore.humanize.upcase
75
76     {
77       is_inner: true,
78       type: type,
79       left_clause: transform_tree(clause.left),
80       right_clause: transform_tree(clause.right),
81     }
82   end
83 end
84 end
85 end
86 end

```

Source code for lib/sql\_assess/parsers/distinct\_filter.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Parsers
5      # Parser for the distinct filter
6      # @author Vlad Stoica
7      class DistinctFilter < Base
8        # @return [String] distinct filter or ALL if no distinct filter is mentioned
9        def distinct_filter
10         @parsed_query.query_expression.filter || 'ALL'
11       end
12     end
13 end
14 end

```

Source code for lib/sql\_assess/parsers/tables.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Parsers
5      # Parser for the FROM clause
6      # @author Vlad Stoica

```



```

7 class Tables < Base
8   # @return [Array<Hash{type:, table:, sql:}, Hash{join_type:, table: Hash{type:, table:, sql:}, sql:>}]
9   def tables
10    if @parsed_query.query_expression&.table_expression&.from_clause.nil?
11      []
12    else
13      @parsed_query.query_expression.table_expression.from_clause.tables.map do |expression|
14        transform(expression)
15      end.flatten
16    end
17  end
18
19  private
20
21  def transform(query)
22    if query.is_a?(SQLParser::Statement::Table)
23      {
24        type: 'table',
25        table: query.to_sql,
26        sql: query.to_sql,
27      }
28    elsif query.is_a?(SQLParser::Statement::JoinedTable)
29      hash = {
30        join_type: query.class.name.split('::').last.underscore.humanize.upcase,
31        table: transform(query.right),
32        sql: "#{query.class.name.split('::').last.underscore.humanize.upcase} #{query.right.to_sql}",
33      }
34
35      if query.is_a?(SQLParser::Statement::QualifiedJoin)
36        hash[:condition] = Where.transform(
37          query.search_condition.search_condition
38        )
39        hash[:sql] = "#{query.class.name.split('::').last.underscore.humanize.upcase}
40          ↳ #{query.right.to_sql} #{query.search_condition.to_sql}"
41      end
42
43      [transform(query.left), hash].flatten
44    elsif query.is_a?(SQLParser::Statement::Subquery)
45      {
46        type: 'Subquery',
47        sql: query.to_sql,
48        attributes:
49          ↳ SqlAssess::QueryAttributeExtractor.new.extract_query(query.query_specification.to_sql),
50      }
51    end
52  end
53 end

```

Source code for lib/sql\_assess/parsers/group.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Parsers
5     # Parser for the GROUP clause
6     # @author Vlad Stoica
7     class Group < Base
8       # @return [Array<String>] the list of columns in the group clause
9       def group
10        if @parsed_query.query_expression.table_expression.group_by_clause.nil?
11          []
12        else

```

```

13         @parsed_query.query_expression.table_expression.group_by_clause.columns.map(&:to_sql)
14     end
15 end
16 end
17 end
18 end

```

Source code for lib/sql\_assess/parsers/columns.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Parsers
5     # @author Vlad Stoica
6     # Parser for the columns
7     class Columns < Base
8       # @return [Array<String>] the list of columns selected
9       def columns
10         @parsed_query.query_expression.list.columns.map(&:to_sql)
11       end
12     end
13 end
14 end

```

Source code for lib/sql\_assess/parsers/order\_by.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Parsers
5     # Parser for the Order BY clause
6     # @author Vlad Stoica
7     class OrderBy < Base
8       # @return [Array<Hash{column:, position:}>]
9       def order
10         if @parsed_query.order_by.nil?
11           []
12         else
13           @parsed_query.order_by.sort_specification.each_with_index.map do |column, i|
14             {
15               column: column.to_sql,
16               position: i,
17             }
18           end
19         end
20       end
21     end
22 end
23 end

```

Source code for lib/sql\_assess/parsers/limit.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Parsers
5     # Parser for the limit clause
6     # @author Vlad Stoica
7     class Limit < Base
8       # @return [Hash{limit:, offset:}]. If offset is not present then return 0,
9       # if limit is not present then return inf.
10       def limit
11         if @parsed_query.query_expression&.table_expression&.limit_clause.present?
12           {
13             limit: @parsed_query.query_expression&.table_expression&.limit_clause&.limit,

```

```

14         offset: @parsed_query.query_expression&.table_expression&.limit_clause&.offset || 0,
15     }
16   else
17     {
18       limit: 'inf',
19       offset: 0,
20     }
21   end
22 end
23 end
24 end
25 end

```

Source code for lib/sql\_assess/parsers/where.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Parsers
5      # @author Vlad Stoica
6      # Parser for the WHERE clause
7      class Where < Base
8        # @return [Hash] the binary expression tree of the WHERE clause
9        def where
10          if @parsed_query.query_expression.table_expression.where_clause.nil?
11            {}
12          else
13            self.class.transform(@parsed_query.query_expression.table_expression.where_clause.search_condition)
14          end
15        end
16
17        # @return [Hash] the expression tree (not binary tree) of the WHERE clause
18        def where_tree
19          if @parsed_query.query_expression.table_expression.where_clause.nil?
20            {}
21          else
22            transform_tree(
23              @parsed_query.query_expression.table_expression.where_clause.search_condition
24            )
25          end
26        end
27
28        # Transform a clause to a tree
29        # @param [SQLParser::Statement] clause current node
30        # @return [Hash] tree version the clause
31        def self.transform(clause)
32          if clause.is_a?(SQLParser::Statement::ComparisonPredicate)
33            {
34              type: clause.class.name.split('::').last.underscore.humanize.upcase,
35              left: clause.left.to_sql,
36              right: clause.right.to_sql,
37              sql: clause.to_sql,
38            }
39          elsif clause.is_a?(SQLParser::Statement::SearchCondition)
40            type = clause.class.name.split('::').last.underscore.humanize.upcase
41            transform_left = merge(type, transform(clause.left))
42            transform_right = merge(type, transform(clause.right))
43            {
44              type: type,
45              clauses: [
46                transform_left,
47                transform_right,
48              ].flatten,
49            }

```

```

50     end
51 end
52
53 def self.merge(type, clause)
54   if clause[:type] == type
55     clause[:clauses]
56   else
57     clause
58   end
59 end
60 private_class_method :merge
61
62 private
63
64 def transform_tree(clause)
65   if clause.is_a?(SQLParser::Statement::ComparisonPredicate)
66     {
67       is_inner: false,
68       type: clause.class.name.split('::').last.underscore.humanize.upcase,
69       left: clause.left.to_sql,
70       right: clause.right.to_sql,
71       sql: clause.to_sql,
72     }
73   elsif clause.is_a?(SQLParser::Statement::SearchCondition)
74     type = clause.class.name.split('::').last.underscore.humanize.upcase
75
76     {
77       is_inner: true,
78       type: type,
79       left_clause: transform_tree(clause.left),
80       right_clause: transform_tree(clause.right),
81     }
82   end
83 end
84 end
85 end
86 end

```

Source code for lib/sql\_assess/database\_connection.rb

```

1  # frozen_string_literal: true
2
3  require 'mysql2'
4
5  module SqlAssess
6    # Class for handling database connection and securely executing queries
7    # @author Vlad Stoica
8    class DatabaseConnection
9      def initialize(host: '127.0.0.1', port: '3306', username: 'root', database: nil, password: '')
10        @client = Mysql2::Client.new(
11          host: host,
12          port: port,
13          username: username,
14          password: password,
15          flags: Mysql2::Client::MULTI_STATEMENTS
16        )
17
18        if database.present?
19          @parent_database = true
20          @database = database
21          @client.query("CREATE DATABASE IF NOT EXISTS `#{@database}`")
22        else
23          success = false
24          attempt = 0

```

```

25 until success
26   if attempt.positive?
27     @database = "#{Time.now.strftime('%H%M%S')}_#{attempt}"
28   else
29     @database = Time.now.strftime('%H%M%S').to_s
30   end
31
32   begin
33     @client.query("CREATE DATABASE `#{@database}`")
34     success = true
35   rescue Mysql2::Error => exception
36     raise exception unless exception.message.include?('database exists')
37     success = false
38     attempt += 1
39   end
40 end
41
42
43 @client.query("CREATE USER IF NOT EXISTS `#{@database}`;")
44 @client.query("GRANT ALL PRIVILEGES ON `#{@database}`.* TO `#{@database}` WITH GRANT OPTION;")
45
46 @restricted_client = Mysql2::Client.new(
47   host: host,
48   port: port,
49   username: @database,
50   flags: Mysql2::Client::MULTI_STATEMENTS
51 )
52
53 @restricted_client.select_db(@database)
54 @client.select_db(@database)
55 rescue Mysql2::Error => exception
56   raise DatabaseConnectionError, exception.message
57 end
58
59 # Execute queries as restricted user
60 # @param [String] query
61 # @return [Hash] the results of the query
62 def query(query)
63   @restricted_client.query(query)
64 end
65
66 # Drop the temporary database and the temporary user
67 def delete_database
68   if @parent_database
69     # disable foreign key checks before dropping the database
70     @client.query('SET FOREIGN_KEY_CHECKS = 0')
71
72     tables = query('SHOW tables')
73
74     tables.each do |table|
75       table_name = table['Tables_in_local_db']
76       @client.query("DROP table `#{table_name}`")
77     end
78
79     @client.query('SET FOREIGN_KEY_CHECKS = 1')
80   else
81     @client.query("DROP DATABASE `#{@database}`")
82     @client.query("DROP USER IF EXISTS `#{@database}`")
83   end
84 end
85
86 # Execute a multi statement query as restricted user
87 # @param [String] query

```

```

88 # @return [Array<Hash>] the results of each statement
89 def multiple_query(query)
90   result = []
91
92   result << @restricted_client.query(query)
93
94   while @restricted_client.next_result
95     result << @restricted_client.store_result
96   end
97
98   result
99 end
100 end
101 end

```

Source code for lib/sql\_assess/query\_comparator.rb

```

1 # frozen_string_literal: true
2
3 require 'sql_assess/query_comparison_result'
4 require 'sql_assess/parsers/base'
5
6 module SqlAssess
7   # Class for handling the comparison of results between two queries
8   # @author Vlad Stoica
9   class QueryComparator
10     def initialize(connection)
11       @connection = connection
12     end
13
14     # Compares the results of two queries
15     #
16     # @param [String] instructor_sql_query
17     # @param [String] student_sql_query
18     # @return [Boolean] whether the result matches
19     def compare(instructor_sql_query, student_sql_query)
20       instructor_result = @connection.query(instructor_sql_query).to_a
21       student_result = @connection.query(student_sql_query).to_a
22
23       success?(instructor_result, student_result)
24     end
25
26     private
27
28     def success?(instructor_result, student_result)
29       return false if instructor_result.count != student_result.count
30
31       (0..instructor_result.count).all? do |i|
32         instructor_result[i] == student_result[i]
33       end
34     end
35   end
36 end

```

Source code for lib/sql\_assess/query\_comparison\_result.rb

```

1 # frozen_string_literal: true
2
3 require 'sql_assess/grader/base'
4
5 module SqlAssess
6   # @author Vlad Stoica
7   # The final result of an assesment
8   # @!attribute [r] success
9   #   @return [Boolean] whether the query returned the same results

```

```

10 # @!attribute [r] attributes
11 #   @return [Hash] The extracted attributes of the two queries. See
12 #     {QueryAttributeExtractor}
13 # @!attribute [r] attributes_grade
14 #   @return [Hash] The grade for each component
15 # @!attribute [r] grade
16 #   @return [Double] The overall grade
17 # @!attribute [r] message
18 #   @return [String] Hint
19 class QueryComparisonResult
20   attr_reader :success, :attributes, :grade, :message
21
22   def initialize(success:, attributes:)
23     @success = success
24     @attributes = attributes
25
26     attributes_grade
27
28     if @success == true
29       @grade = calculate_grade * 100.00
30     else
31       @grade = calculate_grade * 90.00
32     end
33
34     @message = determine_hints
35   end
36
37   def attributes_grade
38     @attributes_grade ||= grade_components_percentages.keys.map do |key|
39       key_hash = key == :where ? :where_tree : key
40       [
41         key,
42         SqlAssess::Grader::Base.grade_for(
43           attribute: key,
44           student_attributes: attributes[:student][key_hash],
45           instructor_attributes: attributes[:instructor][key_hash]
46         ).to_d,
47       ]
48     end.to_h
49   end
50
51   private
52
53   def calculate_grade
54     attributes_grade.sum do |attribute, grade|
55       grade * grade_components_percentages[attribute]
56     end
57   end
58
59   def grade_components_percentages
60     {
61       tables: 1 / 8.0,
62       columns: 1 / 8.0,
63       group: 1 / 8.0,
64       where: 1 / 8.0,
65       distinct_filter: 1 / 8.0,
66       limit: 1 / 8.0,
67       order_by: 1 / 8.0,
68       having: 1 / 8.0,
69     }
70   end
71
72   def determine_hints

```

```

73 if @grade == 100.00
74   'Congratulations! Your solution is correct'
75 else
76   "Your query is not correct. #{message_for_attribute(first_wrong_component)}"
77 end
78 end
79
80 def first_wrong_component
81   comp = grade_components_percentages.detect do |component, _|
82     attributes_grade[component].to_d != 1
83   end
84
85   comp.present? ? comp.first : nil
86 end
87
88 def message_for_attribute(attribute)
89   case attribute
90   when :columns then 'Check what columns you are selecting.'
91   when :tables then 'Are you sure you are selecting the right tables?'
92   when :order_by then 'Are you ordering the rows correctly?'
93   when :where then 'Looks like you are selecting the right columns, but you are not selecting only the
94     ↪ correct rows.'
95   when :distinct_filter then 'What about duplicates? What does the exercise say?'
96   when :limit then 'Are you selecting the correct number of rows?'
97   when :group then 'Are you grouping by the correct columns?'
98   end
99 end
100 end

```

Source code for lib/sql\_assess/transformers/base.rb

```

1  # frozen_string_literal: true
2
3  require 'sql-parser'
4
5  module SqlAssess
6    # Module for canonicalization transformers
7    module Transformers
8      # Base transformer. Provides implementation for traversing from, for
9      # getting the list of tables.
10     # @abstract
11     # @author Vlad Stoica
12     class Base
13       def initialize(connection)
14         @connection = connection
15         @parser = SQLParser::Parser.new
16       end
17
18       # Transform method that must be implemented in subclasses
19       def transform
20         raise 'Implement this method in subclass'
21       end
22
23       # Gets the full list of tables from a query. It assumes that there are
24       # no sub-queries involved
25       # @return [Array<String>] the list of tables
26       def tables(query)
27         SqlAssess::Parsers::Tables.new(query).tables.map do |table|
28           if table.key?(:join_type)
29             table[:table][:table].remove('`)
30           else
31             table[:table].remove('`)
32           end

```



```

33     end
34 end
35
36 private
37
38 def traverse_from(node)
39   if node.is_a?(SQLParser::Statement::QualifiedJoin)
40     node.class.new(
41       traverse_from(node.left),
42       traverse_from(node.right),
43       SQLParser::Statement::On.new(
44         transform_tree(node.search_condition.search_condition)
45       )
46     )
47   elsif node.is_a?(SQLParser::Statement::JoinedTable)
48     node.class.new(
49       traverse_from(node.left),
50       traverse_from(node.right)
51     )
52   else
53     node
54   end
55 end
56 end
57 end
58 end
59
60 require_relative 'all_columns'
61 require_relative 'from_subquery'
62
63 require_relative 'not/base'
64 require_relative 'ambiguous_columns/base'
65 require_relative 'between_predicate/base'
66 require_relative 'comparison_predicate/base'
67 require_relative 'equivalent_columns/base'

```

Source code for lib/sql\_assess/transformers/between\_predicate/base.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Transformers
5      # Transformers for the between predicate
6      module BetweenPredicate
7        # @author Vlad Stoica
8        # Base class for transformers for between predicate to two >= and <=
9        class Base < SqlAssess::Transformers::Base
10          # The list of between predicate transformers
11          def self.transformers
12            [From, Where, Having]
13          end
14
15          private
16
17          def transform_between(between)
18            SQLParser::Statement::And.new(
19              SQLParser::Statement::GreaterOrEquals.new(between.left, between.min),
20              SQLParser::Statement::LessOrEquals.new(between.left, between.max)
21            )
22          end
23
24          def transform_tree(node)
25            if node.is_a?(SQLParser::Statement::SearchCondition)
26              node.class.new(

```

```

27         transform_tree(node.left),
28         transform_tree(node.right)
29     )
30     elsif node.is_a?(SQLParser::Statement::Between)
31         transform_between(node)
32     else
33         node
34     end
35 end
36 end
37 end
38 end
39 end

```

```

41 require_relative 'from'
42 require_relative 'where'
43 require_relative 'having'

```

Source code for lib/sql\_assess/transformers/between\_predicate/having.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      module Transformers
5          module BetweenPredicate
6              # Between transformer for Having clause
7              # @author Vlad Stoica
8              class Having < Base
9                  # Transforms the query
10                 #
11                 # @param [String] query the initial query
12                 # @return [String] the transformed query
13                 # @example
14                 #   With tables: t1(id1), t2(id3);
15                 #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING SUM(`id3`) BETWEEN 1 AND 3 GROUP BY 1
16                 #   is transformed to
17                 #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING SUM(`id3`) >=1 AND HAVING SUM(`id3`) <= 3 GROUP BY
18                 ↪ 1
19                 def transform(query)
20                     parsed_query = @parser.scan_str(query)
21
22                     having_clause = parsed_query.query_expression.table_expression.having_clause
23
24                     return query if having_clause.nil?
25
26                     transformed_having_clause = transform_tree(having_clause.search_condition)
27
28                     parsed_query.query_expression.table_expression.having_clause.instance_variable_set(
29                         '@search_condition', transformed_having_clause
30                     )
31
32                     parsed_query.to_sql
33                 end
34             end
35         end
36     end

```

Source code for lib/sql\_assess/transformers/between\_predicate/from.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      module Transformers
5          module BetweenPredicate

```

```

6 # Between transformer for FROM clause
7 # @author Vlad Stoica
8 class From < Base
9   # Transforms the query
10  #
11  # @param [String] query the initial query
12  # @return [String] the transformed query
13  #
14  # @example
15  #   With tables: t1(id1), t2(id3);
16  #   SELECT * FROM `t1` LEFT JOIN `t2` ON id1 BETWEEN id2 and 3
17  #   is transformed
18  #   SELECT * FROM `t1` LEFT JOIN `t2` ON id1 >= id2 AND id1 <= 3
19  def transform(query)
20    parsed_query = @parser.scan_str(query)
21
22    join_clause = parsed_query.query_expression&.table_expression&.from_clause
23
24    return query if join_clause.nil?
25
26    new_tables = join_clause.tables.map do |table|
27      traverse_from(table)
28    end
29
30    parsed_query.query_expression.table_expression.from_clause.instance_variable_set(
31      '@tables', new_tables
32    )
33
34    parsed_query.to_sql
35  end
36 end
37 end
38 end
39 end

```

Source code for lib/sql\_assess/transformers/between\_predicate/where.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module BetweenPredicate
6       # Between transformer for WHERE clause
7       # @author Vlad Stoica
8       class Where < Base
9         # Transforms the query
10        #
11        # @param [String] query the initial query
12        # @return [String] the transformed query
13        #
14        # @example
15        #   With tables: t1(id1), t2(id3);
16        #   SELECT `id1` FROM `t1`, `t2` WHERE `id3` BETWEEN 1 AND 3
17        #   is transformed to
18        #   SELECT `id1` FROM `t1`, `t2` WHERE `id3` >=1 AND `id3` <= 3
19        def transform(query)
20          parsed_query = @parser.scan_str(query)
21
22          where_clause = parsed_query.query_expression.table_expression.where_clause
23
24          return query if where_clause.nil?
25
26          transformed_where_clause = transform_tree(where_clause.search_condition)
27

```



```

50         transform_tree(node.right)
51     )
52     else
53         transform_column(node)
54     end
55 end
56
57 def find_table_for(column_name)
58     table_list = tables(@parsed_query.to_sql)
59
60     table_list.detect do |table|
61         columns_query = "SHOW COLUMNS from #{table}"
62         columns = @connection.query(columns_query).map { |k| k['Field'] }
63         columns.map(&:downcase).include?(column_name.downcase)
64     end
65 end
66
67 end
68
69 end
70
71 require_relative 'from'
72 require_relative 'group'
73 require_relative 'order_by'
74 require_relative 'select'
75 require_relative 'where'
76 require_relative 'having'

```

Source code for lib/sql\_assess/transformers/ambiguous\_columns/having.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      module Transformers
5          module AmbiguousColumns
6              # @author Vlad Stoica
7              # Ambiguous columns transformer for the Having clause
8              class Having < Base
9                  # Transforms the query
10
11                  #
12                  # @param [String] query the initial query
13                  # @return [String] the transformed query
14                  #
15                  # @example
16                  #   With tables: t1(id1), t2(id3);
17                  #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING SUM(`id3`) > 3 GROUP BY 1
18                  #   is transformed to
19                  #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING SUM(`t2`.`id3`) > 3 GROUP BY 1
20              def transform(query)
21                  @parsed_query = @parser.scan_str(query)
22
23
24                  having_clause = @parsed_query.query_expression.table_expression.having_clause
25
26                  return query if having_clause.nil?
27
28                  transformed_having_clause = transform_tree(having_clause.search_condition)
29
30                  @parsed_query.query_expression.table_expression.having_clause.instance_variable_set(
31                      '@search_condition', transformed_having_clause
32                  )
33
34                  @parsed_query.to_sql
35              end
36          end
37      end
38  end

```

```
35 end
36 end
37 end
```

Source code for lib/sql\_assess/transformers/ambiguous\_columns/select.rb

```
1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module AmbiguousColumns
6       # @author Vlad Stoica
7       # Ambiguous columns transformer for the Select clause
8       class Select < Base
9         # Transforms the query
10        #
11        # @param [String] query the initial query
12        # @return [String] the transformed query
13        # @example
14        #   With tables: t1(id1), t2(id3);
15        #   SELECT `id1` FROM `t1`, `t2`
16        #   is transformed to
17        #   SELECT `t1`.`id1` FROM `t1`, `t2`
18        def transform(query)
19          @query = query
20          @parsed_query = @parser.scan_str(query)
21
22          columns = @parsed_query.query_expression.list.columns.map do |column|
23            transform_column(column)
24          end
25
26          @parsed_query.query_expression.list.instance_variable_set(
27            '@columns',
28            columns
29          )
30
31          @parsed_query.to_sql
32        end
33      end
34    end
35  end
36 end
```

Source code for lib/sql\_assess/transformers/ambiguous\_columns/from.rb

```
1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module AmbiguousColumns
6       # @author Vlad Stoica
7       # Ambiguous columns transformer for the FROM clause
8       class From < Base
9         # Transforms the query
10        #
11        # @param [String] query the initial query
12        # @return [String] the transformed query
13        #
14        # @example
15        #   With tables: t1(id1), t2(id3);
16        #   SELECT `id1` FROM `t1` LEFT JOIN `t2` on `id1` = `id3`
17        #   is transformed to
18        #   SELECT `id1` FROM `t1` LEFT JOIN `t2` on `t1`.`id1` = `t2`.`id3`
19        def transform(query)
20          @parsed_query = @parser.scan_str(query)
```

```

21 join_clause = @parsed_query.query_expression.table_expression.from_clause
22
23
24 return query if join_clause.nil?
25
26 new_tables = join_clause.tables.map do |table|
27   traverse_from(table)
28 end
29
30 @parsed_query.query_expression.table_expression.from_clause.instance_variable_set(
31   '@tables', new_tables
32 )
33
34 @parsed_query.to_sql
35 end
36 end
37 end
38 end
39 end

```

Source code for lib/sql\_assess/transformers/ambiguous\_columns/group.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module AmbiguousColumns
6       # @author Vlad Stoica
7       # Ambiguous columns transformer for the GROUP clause
8       class Group < Base
9         # Transforms the query
10
11         # @param [String] query the initial query
12         # @return [String] the transformed query
13
14         # @example
15         #   With tables: t1(id1), t2(id3);
16         #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` GROUP BY `id1`
17         #   is transformed
18         #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` GROUP BY `t1`.`id1`
19
20         # @example
21         #   With tables: t1(id1), t2(id3);
22         #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` GROUP BY 1
23         #   is transformed to
24         #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` GROUP BY `t1`.`id1`
25         def transform(query)
26           @query = query
27
28           @parsed_query = @parser.scan_str(query)
29
30           if @parsed_query.query_expression.table_expression.group_by_clause.nil?
31             return @parsed_query.to_sql
32           end
33
34           columns = @parsed_query.query_expression.table_expression.group_by_clause.columns.map do |column|
35             transform_column_integer(column)
36           end
37
38           @parsed_query.query_expression.table_expression.group_by_clause.instance_variable_set(
39             '@columns',
40             columns
41           )
42

```

```

43         @parsed_query.to_sql
44     end
45 end
46 end
47 end
48 end

```

Source code for lib/sql\_assess/transformers/ambiguous\_columns/order\_by.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Transformers
5      module AmbiguousColumns
6        # @author Vlad Stoica
7        # Ambiguous columns transformer for the ORDER BY clause
8        class OrderBy < Base
9          # Transforms the query
10         #
11         # @param [String] query the initial query
12         # @return [String] the transformed query
13         #
14         # @example
15         #   With tables: t1(id1), t2(id3);
16         #   SELECT `id1` FROM `t1`, `t2` ORDER BY 1
17         #   is transformed to
18         #   SELECT `id1` FROM `t1`, `t2` ORDER BY `t1`.`id1`
19         def transform(query)
20           @parsed_query = @parser.scan_str(query)
21
22           return @parsed_query.to_sql if @parsed_query.order_by.nil?
23
24           sort_specification = @parsed_query.order_by.sort_specification.map do |specification|
25             specification.class.new(
26               transform_column_integer(specification.column)
27             )
28           end
29
30           @parsed_query.order_by.instance_variable_set(
31             '@sort_specification',
32             sort_specification
33           )
34
35           @parsed_query.to_sql
36         end
37       end
38     end
39   end
40 end

```

Source code for lib/sql\_assess/transformers/ambiguous\_columns/where.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Transformers
5      module AmbiguousColumns
6        # @author Vlad Stoica
7        # Ambiguous columns transformer for the WHERE clause
8        class Where < Base
9          # Transforms the query
10         #
11         # @param [String] query the initial query
12         # @return [String] the transformed query
13         #

```



```

14 # @example
15 # With tables: t1(id1), t2(id3);
16 # SELECT `id1` FROM `t1`, `t2` WHERE `id3` > 3
17 # is transformed to
18 # SELECT `id1` FROM `t1`, `t2` WHERE `t2`.`id3` > 3
19 def transform(query)
20   @parsed_query = @parser.scan_str(query)
21
22   where_clause = @parsed_query.query_expression.table_expression.where_clause
23
24   return query if where_clause.nil?
25
26   transformed_where_clause = transform_tree(where_clause.search_condition)
27
28   @parsed_query.query_expression.table_expression.where_clause.instance_variable_set(
29     '@search_condition', transformed_where_clause
30   )
31
32   @parsed_query.to_sql
33 end
34 end
35 end
36 end
37 end

```

Source code for lib/sql\_assess/transformers/all\_columns.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     # @author Vlad Stoica
6     # Transformer for transforming * to the full list of qualified columns.
7     class AllColumns < Base
8       # Transforms the query
9       #
10      # @param [String] query the initial query
11      # @return [String] the transformed query
12      #
13      # @example
14      # With tables: t1(id1), t2(id3);
15      # "SELECT * FROM `t1`, `t2`"
16      # is transformed
17      # to "SELECT `t1`.`id1`, `t2`.`id3` FROM `t1`, `t2`"
18      #
19      # @example
20      # With tables: t1(id1), t2(id3);
21      # "SELECT `t1`.`id1` FROM `t1`, `t2`"
22      # is transformed
23      # to "SELECT `t1`.`id1` FROM `t1`, `t2`"
24      def transform(query)
25        @parsed_query = @parser.scan_str(query)
26
27        if @parsed_query.query_expression.list.is_a?(SQLParser::Statement::All)
28          transform_star_select
29        end
30
31        @parsed_query.to_sql
32      end
33
34      private
35
36      def transform_star_select
37        table_list = tables(@parsed_query.to_sql)

```

```

38 new_columns = table_list.map do |table|
39   columns_query = "SHOW COLUMNS from #{table}"
40   columns = @connection.query(columns_query).map { |k| k['Field'] }
41
42   columns.map do |column_name|
43     SQLParser::Statement::QualifiedColumn.new(
44       SQLParser::Statement::Table.new(table),
45       SQLParser::Statement::Column.new(column_name)
46     )
47   end
48 end.flatten
49
50 @parsed_query.query_expression.instance_variable_set(
51   '@list',
52   SQLParser::Statement::SelectList.new(new_columns)
53 )
54 end
55 end
56 end
57 end
58 end

```

Source code for lib/sql\_assess/transformers/not/base.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Transformers
5      # Namespace for NOT transformers
6      module Not
7        # @author Vlad Stoica
8        # Base class for transformers for not
9        class Base < SqlAssess::Transformers::Base
10          # The list not columns transformers
11          def self.transformers
12            [From, Where, Having]
13          end
14
15          private
16
17          def transform_not(not_statement)
18            # Greater
19            if not_statement.value.is_a?(SQLParser::Statement::Greater)
20              SQLParser::Statement::LessOrEquals.new(not_statement.value.left, not_statement.value.right)
21            elsif not_statement.value.is_a?(SQLParser::Statement::GreaterOrEquals)
22              SQLParser::Statement::Less.new(not_statement.value.left, not_statement.value.right)
23            # Less
24            elsif not_statement.value.is_a?(SQLParser::Statement::Less)
25              SQLParser::Statement::GreaterOrEquals.new(not_statement.value.left, not_statement.value.right)
26            elsif not_statement.value.is_a?(SQLParser::Statement::LessOrEquals)
27              SQLParser::Statement::Greater.new(not_statement.value.left, not_statement.value.right)
28            else
29              not_statement
30            end
31          end
32
33          def transform_tree(node)
34            if node.is_a?(SQLParser::Statement::SearchCondition)
35              node.class.new(
36                transform_tree(node.left),
37                transform_tree(node.right)
38              )
39            elsif node.is_a?(SQLParser::Statement::Not)
40              transform_not(node)

```

```

41         else
42             node
43         end
44     end
45 end
46 end
47 end
48 end
49

```

```

50 require_relative 'from'
51 require_relative 'where'
52 require_relative 'having'

```

Source code for lib/sql\_assess/transformers/not/having.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      module Transformers
5          module Not
6              # NOT transformer for the HAVING clause
7              class Having < Base
8                  # Transforms the query
9                  #
10                 # @param [String] query the initial query
11                 # @return [String] the transformed query
12                 # @example
13                 #   With tables: t1(id1), t2(id3);
14                 #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING NOT SUM(`id3`) > 3 GROUP BY 1
15                 #   is transformed to
16                 #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING SUM(`t2`.`id3`) <= 3 GROUP BY 1
17                 def transform(query)
18                     parsed_query = @parser.scan_str(query)
19
20                     having_clause = parsed_query.query_expression.table_expression.having_clause
21
22                     return query if having_clause.nil?
23
24                     transformed_having_clause = transform_tree(having_clause.search_condition)
25
26                     parsed_query.query_expression.table_expression.having_clause.instance_variable_set(
27                         '@search_condition', transformed_having_clause
28                     )
29
30                     parsed_query.to_sql
31                 end
32             end
33         end
34     end
35 end

```

Source code for lib/sql\_assess/transformers/not/from.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      module Transformers
5          module Not
6              # @author Vlad Stoica
7              # NOT transformer for the FROM clause
8              class From < Base
9                  # Transforms the query
10                 #
11                 # @param [String] query the initial query
12                 # @return [String] the transformed query

```

```

13 #
14 # @example
15 #   With tables: t1(id1), t2(id3);
16 #   SELECT * FROM `t1` LEFT JOIN `t2` ON NOT id1 > id2
17 #   is transformed
18 #   SELECT * FROM `t1` LEFT JOIN `t2` ON id1 <= id2
19 def transform(query)
20   parsed_query = @parser.scan_str(query)
21
22   join_clause = parsed_query.query_expression&.table_expression&.from_clause
23
24   return query if join_clause.nil?
25
26   new_tables = join_clause.tables.map do |table|
27     traverse_from(table)
28   end
29
30   parsed_query.query_expression.table_expression.from_clause.instance_variable_set(
31     '@tables', new_tables
32   )
33
34   parsed_query.to_sql
35 end
36 end
37 end
38 end
39 end

```

Source code for lib/sql\_assess/transformers/not/where.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module Not
6       # NOT transformer for the WHERE clause
7       class Where < Base
8         # Transforms the query
9         #
10        # @param [String] query the initial query
11        # @return [String] the transformed query
12        #
13        # @example
14        #   With tables: t1(id1), t2(id3);
15        #   SELECT * FROM `t1`, `t2` WHERE NOT id1 > id2
16        #   is transformed
17        #   SELECT * FROM `t1`, `t2` WHERE id1 <= id2
18        def transform(query)
19          parsed_query = @parser.scan_str(query)
20
21          where_clause = parsed_query.query_expression.table_expression.where_clause
22
23          return query if where_clause.nil?
24
25          transformed_where_clause = transform_tree(where_clause.search_condition)
26
27          parsed_query.query_expression.table_expression.where_clause.instance_variable_set(
28            '@search_condition', transformed_where_clause
29          )
30
31          parsed_query.to_sql
32        end
33      end
34    end
35  end
36 end

```

```

35 end
36 end

Source code for lib/sql_assess/transformers/comparison_predicate/base.rb

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     # Transformers for comparison predicate
6     module ComparisonPredicate
7       # @author Vlad Stoica
8       # Base class for transformers for comparison predicate
9       class Base < SqlAssess::Transformers::Base
10        # The list of comparison predicate transformers
11        def self.transformers
12          [From, Where, Having]
13        end
14
15        private
16
17        def transform_comparison_predicate(predicate)
18          if predicate.is_a?(SQLParser::Statement::Greater)
19            SQLParser::Statement::Less.new(
20              predicate.right,
21              predicate.left
22            )
23          elsif predicate.is_a?(SQLParser::Statement::GreaterOrEquals)
24            SQLParser::Statement::LessOrEquals.new(
25              predicate.right,
26              predicate.left
27            )
28          else
29            predicate
30          end
31        end
32
33        def transform_tree(node)
34          if node.is_a?(SQLParser::Statement::SearchCondition)
35            node.class.new(
36              transform_tree(node.left),
37              transform_tree(node.right)
38            )
39          else
40            transform_comparison_predicate(node)
41          end
42        end
43      end
44    end
45  end
46 end
47
48 require_relative 'from'
49 require_relative 'where'
50 require_relative 'having'

```

Source code for lib/sql\_assess/transformers/comparison\_predicate/having.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module ComparisonPredicate
6       # Comparison predicate transformer for HAVING clause
7       # @author Vlad Stoica

```

```

8 class Having < Base
9   # Transforms the query
10  #
11  # @param [String] query the initial query
12  # @return [String] the transformed query
13  #
14  # @example
15  #   With tables: t1(id1), t2(id3);
16  #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` HAVING SUM(`id3`) > 1 GROUP BY 1
17  #   is transformed to
18  #   SELECT `id1`, SUM(`id3`) FROM `t1`, `t2` 1 < HAVING SUM(`id3`) GROUP BY 1
19  def transform(query)
20    parsed_query = @parser.scan_str(query)
21
22    having_clause = parsed_query.query_expression.table_expression.having_clause
23
24    return query if having_clause.nil?
25
26    transformed_having_clause = transform_tree(having_clause.search_condition)
27
28    parsed_query.query_expression.table_expression.having_clause.instance_variable_set(
29      '@search_condition', transformed_having_clause
30    )
31
32    parsed_query.to_sql
33  end
34 end
35 end
36 end
37 end

```

Source code for lib/sql\_assess/transformers/comparison\_predicate/from.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module ComparisonPredicate
6       # Comparison predicate transformer for FROM clause
7       # @author Vlad Stoica
8       class From < Base
9         # Transforms the query
10        #
11        # @param [String] query the initial query
12        # @return [String] the transformed query
13        #
14        # @example
15        #   With tables: t1(id1), t2(id3);
16        #   SELECT * FROM `t1` LEFT JOIN `t2` ON id1 > id2
17        #   is transformed
18        #   SELECT * FROM `t1` LEFT JOIN `t2` ON id2 < id1
19        def transform(query)
20          parsed_query = @parser.scan_str(query)
21
22          join_clause = parsed_query.query_expression&.table_expression&.from_clause
23
24          return query if join_clause.nil?
25
26          new_tables = join_clause.tables.map do |table|
27            traverse_from(table)
28          end
29
30          parsed_query.query_expression.table_expression.from_clause.instance_variable_set(
31            '@tables', new_tables

```

```

32     )
33
34     parsed_query.to_sql
35   end
36 end
37 end
38 end
39 end

```

Source code for lib/sql\_assess/transformers/comparison\_predicate/where.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Transformers
5      module ComparisonPredicate
6        # Comparison predicate transformer for WHERE clause
7        # @author Vlad Stoica
8        class Where < Base
9          # Transforms the query
10         #
11         # @param [String] query the initial query
12         # @return [String] the transformed query
13         #
14         # @example
15         #   With tables: t1(id1), t2(id3);
16         #   SELECT `id1` FROM `t1`, `t2` WHERE `id3` > 1
17         #   is transformed to
18         #   SELECT `id1` FROM `t1`, `t2` WHERE 1 < `id3`
19         def transform(query)
20           parsed_query = @parser.scan_str(query)
21
22           where_clause = parsed_query.query_expression.table_expression.where_clause
23
24           return query if where_clause.nil?
25
26           transformed_where_clause = transform_tree(where_clause.search_condition)
27
28           parsed_query.query_expression.table_expression.where_clause.instance_variable_set(
29             '@search_condition', transformed_where_clause
30           )
31
32           parsed_query.to_sql
33         end
34       end
35     end
36   end
37 end

```

Source code for lib/sql\_assess/transformers/from\_subquery.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    module Transformers
5      # @author Vlad Stoica
6      # Equivalent columns transformer for subqueries in the FROM clause
7      # @deprecated Do not use
8      class FromSubquery < Base
9        # Transforms the query
10       #
11       # @param [String] query the initial query
12       # @return [String] the transformed query
13       def transform(query)
14         parsed_query = @parser.scan_str(query)

```

```

15     join_clause = parsed_query.query_expression&.table_expression&.from_clause
16
17
18     return query if join_clause.nil?
19
20     new_tables = join_clause.tables.map do |table|
21       transform_table(table)
22     end
23
24     parsed_query.query_expression.table_expression.from_clause.instance_variable_set(
25       '@tables', new_tables
26     )
27
28     parsed_query.to_sql
29   end
30
31   private
32
33   def transform_table(table)
34     if table.is_a?(SQLParser::Statement::QualifiedJoin)
35       table.class.new(
36         transform_table(table.left),
37         transform_table(table.right),
38         SQLParser::Statement::On.new(
39           table.search_condition.search_condition
40         )
41       )
42     elsif table.is_a?(SQLParser::Statement::Subquery)
43       SQLParser::Statement::Subquery.new(
44         @parser.scan_str(
45           SqlAssess::QueryTransformer.new(@connection).transform(
46             table.query_specification.to_sql
47           )
48         )
49       )
50     else
51       table
52     end
53   end
54 end
55 end
56 end

```

Source code for lib/sql\_assess/transformers/equivalent\_columns/base.rb

```

1  # frozen_string_literal: true
2
3  require 'rgl/adjacency'
4  require 'rgl/condensation.rb'
5
6  module SqlAssess
7    module Transformers
8      # Transformers for equivalent columns
9      module EquivalentColumns
10       # @author Vlad Stoica
11       # Base class for transformers for equivalent column
12       class Base < SqlAssess::Transformers::Base
13         # The list of equivalent columns transformers
14         def self.transformers
15           [Select, Where, Group, OrderBy, Having]
16         end
17
18         private
19

```



```

20 def transform_column(column)
21   if column.is_a?(SQLParser::Statement::QualifiedColumn)
22     equivalence = equivalences_list.detect do |equivalence|
23       equivalences.include?(column.to_sql)
24     end
25
26     if equivalence.present?
27       table_name, column_name = equivalence.sort.first.remove('').split('.')
28
29       SQLParser::Statement::QualifiedColumn.new(
30         SQLParser::Statement::Table.new(table_name),
31         SQLParser::Statement::Column.new(column_name)
32       )
33     else
34       column
35     end
36   elsif column.is_a?(SQLParser::Statement::Aggregate)
37     column.class.new(transform_column(column.column))
38   elsif column.is_a?(SQLParser::Statement::Arithmetic) ||
39     ↪ column.is_a?(SQLParser::Statement::ComparisonPredicate)
40     column.class.new(
41       transform_column(column.left),
42       transform_column(column.right)
43     )
44   else
45     column
46   end
47 end
48
49 def transform_tree(node)
50   if node.is_a?(SQLParser::Statement::SearchCondition)
51     node.class.new(
52       transform_tree(node.left),
53       transform_tree(node.right)
54     )
55   else
56     transform_column(node)
57   end
58 end
59
60 def equivalences_list
61   @equivalences_list ||= build_equivalence_graph.map(&:to_a)
62 end
63
64 def build_equivalence_graph
65   graph = RGL::DirectedAdjacencyGraph.new
66
67   join_conditions = @parsed_query.query_expression.table_expression.from_clause.tables.first
68
69   equivalences = find_equivalences(join_conditions)
70
71   equivalences.each do |equivalence|
72     graph.add_edge(equivalence[:equivalence_left].to_sql, equivalence[:equivalence_right].to_sql)
73     graph.add_edge(equivalence[:equivalence_right].to_sql, equivalence[:equivalence_left].to_sql)
74   end
75
76   graph.condensation_graph.vertices
77 end
78
79 def find_equivalences(clause)
80   if clause.is_a?(SQLParser::Statement::QualifiedJoin)
81     [
82       find_equivalences_search_condition(

```

```

82         clause.search_condition.search_condition
83     ),
84     find_equivalences(clause.left),
85     find_equivalences(clause.right),
86 ].flatten
87 elsif clause.is_a?(SQLParser::Statement::JoinedTable)
88 [
89     find_equivalences(clause.left),
90     find_equivalences(clause.right),
91 ].flatten
92 else
93 []
94 end
95 end
96
97 def find_equivalences_search_condition(search_condition)
98     if search_condition.is_a?(SQLParser::Statement::And)
99         [
100             find_equivalences_search_condition(search_condition.left),
101             find_equivalences_search_condition(search_condition.right),
102         ].flatten
103     elsif search_condition.is_a?(SQLParser::Statement::Equals)
104         [
105             {
106                 equivalence_left: search_condition.left,
107                 equivalence_right: search_condition.right,
108             },
109         ]
110     else
111         []
112     end
113 end
114 end
115 end
116 end
117 end
118
119 require_relative 'group'
120 require_relative 'order_by'
121 require_relative 'select'
122 require_relative 'where'
123 require_relative 'having'

```

Source code for lib/sql\_assess/transformers/equivalent\_columns/having.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4      module Transformers
5          module EquivalentColumns
6              # @author Vlad Stoica
7              # Equivalent columns transformer for HAVING clause
8              class Having < Base
9                  # Transforms the query
10                 #
11                 # @param [String] query the initial query
12                 # @return [String] the transformed query
13                 # @example
14                 #     SELECT *
15                 #     FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
16                 #     HAVING SUM(`b`.`id`) > 3
17                 #
18                 #     is transformed to
19                 #

```

```

20 # SELECT *
21 # FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
22 # HAVING SUM(`a`.`id`) > 3
23 def transform(query)
24   @parsed_query = @parser.scan_str(query)
25
26   having_clause = @parsed_query.query_expression.table_expression.having_clause
27
28   return query if having_clause.nil?
29
30   transformed_having_clause = transform_tree(having_clause.search_condition)
31
32   @parsed_query.query_expression.table_expression.having_clause.instance_variable_set(
33     '@search_condition', transformed_having_clause
34   )
35
36   @parsed_query.to_sql
37 end
38 end
39 end
40 end
41 end

```

Source code for lib/sql\_assess/transformers/equivalent\_columns/select.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module EquivalentColumns
6       # @author Vlad Stoica
7       # Equivalent columns transformer for columns list
8       class Select < Base
9         # Transforms the query
10
11         # @param [String] query the initial query
12         # @return [String] the transformed query
13         # @example
14         #   SELECT `b`.`id`
15         #   FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
16         #
17         #   is transformed to
18         #
19         #   SELECT `a`.`id`
20         #   FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
21         def transform(query)
22           @query = query
23           @parsed_query = @parser.scan_str(query)
24
25           columns = @parsed_query.query_expression.list.columns.map do |column|
26             transform_column(column)
27           end
28
29           @parsed_query.query_expression.list.instance_variable_set(
30             '@columns',
31             columns
32           )
33
34           @parsed_query.to_sql
35         end
36       end
37     end
38   end
39 end

```

# Source code for lib/sql\_assess/transformers/equivalent\_columns/group.rb

```
1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module EquivalentColumns
6       # @author Vlad Stoica
7       # Equivalent columns transformer for GROUP clause
8       class Group < Base
9         # Transforms the query
10        #
11        # @param [String] query the initial query
12        # @return [String] the transformed query
13        # @example
14        #   SELECT *
15        #   FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
16        #   GROUP BY `b`.`id`
17        #
18        #   is transformed to
19        #
20        #   SELECT *
21        #   FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
22        #   GROUP BY `a`.`id`
23        def transform(query)
24          @query = query
25
26          @parsed_query = @parser.scan_str(query)
27
28          if @parsed_query.query_expression.table_expression.group_by_clause.nil?
29            return @parsed_query.to_sql
30          end
31
32          columns = @parsed_query.query_expression.table_expression.group_by_clause.columns.map do |column|
33            transform_column(column)
34          end
35
36          @parsed_query.query_expression.table_expression.group_by_clause.instance_variable_set(
37            '@columns',
38            columns
39          )
40
41          @parsed_query.to_sql
42        end
43      end
44    end
45  end
46 end
```

# Source code for lib/sql\_assess/transformers/equivalent\_columns/order\_by.rb

```
1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module EquivalentColumns
6       # @author Vlad Stoica
7       # Equivalent columns transformer for Order clause
8       class OrderBy < Base
9         # Transforms the query
10        #
11        # @param [String] query the initial query
12        # @return [String] the transformed query
13        # @example
```

```

14 # SELECT *
15 # FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
16 # ORDER BY `b`.`id`
17 #
18 # is transformed to
19 #
20 # SELECT *
21 # FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
22 # ORDER BY `a`.`id`
23 def transform(query)
24   @parsed_query = @parser.scan_str(query)
25
26   return @parsed_query.to_sql if @parsed_query.order_by.nil?
27
28   sort_specification = @parsed_query.order_by.sort_specification.map do |specification|
29     specification.class.new(
30       transform_column(specification.column)
31     )
32   end
33
34   @parsed_query.order_by.instance_variable_set(
35     '@sort_specification',
36     sort_specification
37   )
38
39   @parsed_query.to_sql
40 end
41 end
42 end
43 end
44 end

```

Source code for lib/sql\_assess/transformers/equivalent\_columns/where.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4   module Transformers
5     module EquivalentColumns
6       # @author Vlad Stoica
7       # Equivalent columns transformer for WHERE clause
8       class Where < Base
9         # Transforms the query
10         #
11         # @param [String] query the initial query
12         # @return [String] the transformed query
13         # @example
14         #   SELECT *
15         #   FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
16         #   WHERE `b`.`id` > 3
17         #
18         # is transformed to
19         #
20         #   SELECT *
21         #   FROM `b` LEFT JOIN `a` ON `a`.`id` = `b`.`id`
22         #   WHERE `a`.`id` > 3
23         def transform(query)
24           @parsed_query = @parser.scan_str(query)
25
26           where_clause = @parsed_query.query_expression.table_expression.where_clause
27
28           return query if where_clause.nil?
29
30           transformed_where_clause = transform_tree(where_clause.search_condition)

```

```

31
32         @parsed_query.query_expression.table_expression.where_clause.instance_variable_set(
33             '@search_condition', transformed_where_clause
34         )
35
36         @parsed_query.to_sql
37     end
38 end
39 end
40 end
41 end

```

#### Source code for lib/sql\_assess/version.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4     # Version of the gem
5     VERSION = '0.1.0'
6 end

```

#### Source code for lib/sql\_assess/error.rb

```

1 # frozen_string_literal: true
2
3 module SqlAssess
4     # Base class for errors from the library
5     # @author Vlad Stoica
6     class Error < StandardError
7     end
8
9     # Error thrown when the library can't connect to the database
10    # @author Vlad Stoica
11    class DatabaseConnectionError < SqlAssess::Error
12    end
13
14    # Error thrown when the library encounters an error while executing the schema query
15    # @author Vlad Stoica
16    class DatabaseSchemaError < SqlAssess::Error
17    end
18
19    # Error thrown when the library encounters an error while executing the seed query
20    # @author Vlad Stoica
21    class DatabaseSeedError < SqlAssess::Error
22    end
23
24    # Error thrown when the library encounters an error while executing the instructor's or student's query
25    # @author Vlad Stoica
26    class DatabaseQueryExecutionFailed < SqlAssess::Error
27    end
28
29    # Error thrown when the library cannot canonicalize a query
30    # @author Vlad Stoica
31    class CanonicalizationError < SqlAssess::Error
32    end
33 end

```

#### Source code for lib/sql\_assess/assessor.rb

```

1 # frozen_string_literal: true
2
3 require 'sql_assess/database_connection'
4 require 'sql_assess/runner'
5 require 'sql_assess/query_comparator'
6 require 'sql_assess/query_transformer'

```

```

7 require 'sql_assess/data_extractor'
8 require 'sql_assess/query_attribute_extractor'
9
10 module SqlAssess
11   # Public interface of the library
12   # @author
13   class Assesor
14     attr_reader :connection
15
16     # @raise [DatabaseSchemaError] if any MySQL errors are encountered
17     def initialize(database_host: '127.0.0.1', database_port: '3306', database_username: 'root',
18       ↪ database_password: '')
19       @connection = SqlAssess::DatabaseConnection.new(
20         host: database_host,
21         port: database_port,
22         username: database_username,
23         password: database_password
24       )
25     end
26
27     # Compile an assignment
28     # @param [String] create_schema_sql_query
29     # @param [String] instructor_sql_query
30     # @param [String] seed_sql_query
31     # @return [Hash] see {DataExtractor#run}
32     # @raise [DatabaseSeedError]
33     #   if any MySQL errors are encountered while seeding the database
34     # @raise [DatabaseSchemaError] if any MySQL errors are encountered
35     #   while creating the schema
36     # @raise [DatabaseQueryExecutionFailed] if any MySQL errors are
37     #   encountered while running the instructor query
38     def compile(create_schema_sql_query:, instructor_sql_query:, seed_sql_query:)
39       create_database(create_schema_sql_query, seed_sql_query)
40
41       Runner.new(@connection).execute_query(instructor_sql_query)
42
43       QueryTransformer.new(@connection).transform(instructor_sql_query)
44
45       DataExtractor.new(@connection).run
46     ensure
47       clear_database
48     end
49
50     # Assess an assignment
51     # @param [String] create_schema_sql_query
52     # @param [String] instructor_sql_query
53     # @param [String] seed_sql_query
54     # @return [QueryComparisonResult]
55     # @raise [DatabaseSeedError]
56     #   if any MySQL errors are encountered while seeding the database
57     # @raise [DatabaseSchemaError] if any MySQL errors are encountered
58     #   while creating the schema
59     # @raise [DatabaseQueryExecutionFailed] if any MySQL errors are
60     #   encountered while running the instructor query or student's query
61     def assess(create_schema_sql_query:, instructor_sql_query:, seed_sql_query:, student_sql_query:)
62       create_database(create_schema_sql_query, seed_sql_query)
63
64       # Try to compile
65       Runner.new(@connection).execute_query(student_sql_query)
66
67       query_result_match = QueryComparator.new(@connection)
68         .compare(instructor_sql_query, student_sql_query)

```

```

69 transformer = QueryTransformer.new(@connection)
70 instructor_sql_query = transformer.transform(instructor_sql_query)
71 student_sql_query = transformer.transform(student_sql_query)
72
73 attributes = QueryAttributeExtractor.new.extract(
74   instructor_sql_query, student_sql_query
75 )
76
77 QueryComparisonResult.new(
78   success: query_result_match,
79   attributes: attributes
80 )
81 ensure
82   clear_database
83 end
84
85 private
86
87 def create_database(create_schema_sql_query, seed_sql_query)
88   SqlAssess::Runner.new(@connection).create_schema(
89     create_schema_sql_query
90   )
91
92   SqlAssess::Runner.new(@connection).seed_initial_data(
93     seed_sql_query
94   )
95 end
96
97 def clear_database
98   @connection.delete_database
99 end
100 end
101 end

```

Source code for lib/sql\_assess/data\_extractor.rb

```

1  # frozen_string_literal: true
2
3  require 'mysql2'
4
5  module SqlAssess
6    # Class for handling the extraction of data and schema from a database
7    # @author Vlad Stoica
8    class DataExtractor
9      def initialize(connection)
10        @connection = connection
11      end
12
13      # Extract data from the current connection
14      # @return [Hash] data from the table. The format of the hash is { table_name: [rows] }
15      def run
16        result = []
17        tables = @connection.query('SHOW tables;')
18
19        tables.each do |table|
20          table_name = table.first.last
21
22          data = @connection.query("SELECT * from #{table_name}")
23          columns = @connection.query("SHOW columns from #{table_name}").to_a.map do |column|
24            {
25              name: column.fetch('Field'),
26              type: column.fetch('Type'),
27            }
28          end

```



```

29     result << {
30       name: table_name,
31       columns: columns,
32       data: data.to_a,
33     }
34   end
35 end
36
37 result
38 end
39 end
40 end

```

Source code for lib/sql\_assess/query\_transformer.rb

```

1  # frozen_string_literal: true
2
3  require 'sql_assess/transformers/base'
4
5  module SqlAssess
6    # Class for handling the canonicalization process
7    # @author Vlad Stoica
8    class QueryTransformer
9      # The ordered list of transformers applied
10     TRANSFORMERS = [
11       # Subquery
12       Transformers::FromSubquery,
13       # Predicate
14       Transformers::Not::Base.transformers,
15       Transformers::BetweenPredicate::Base.transformers,
16       Transformers::ComparisonPredicate::Base.transformers,
17       # Columns
18       Transformers::AllColumns,
19       Transformers::AmbiguousColumns::Base.transformers,
20       Transformers::EquivalentColumns::Base.transformers,
21     ].flatten.freeze
22
23     def initialize(connection)
24       @connection = connection
25     end
26
27     # Apply sequentially all transformations to a query
28     #
29     # @param [String] query input query
30     # @return [String] canonicalized query
31     # @raise [CanonicalizationError] if any parsing errors are encountered
32     def transform(query)
33       TRANSFORMERS.each do |transformer_class|
34         query = transformer_class.new(@connection).transform(query)
35       end
36
37       query
38     rescue SQLParser::Parser::ScanError, Racc::ParseError
39       raise CanonicalizationError
40     end
41   end
42 end

```

Source code for lib/sql\_assess/runner.rb

```

1  # frozen_string_literal: true
2
3  module SqlAssess
4    # @author Vlad Stoica
5    # A class for executing various types of queries. By providing a method

```

```

6 # for each type of query, an appropriate error can be returned.
7 class Runner
8   def initialize(connection)
9     @connection = connection
10  end
11
12  # Execute the create schema SQL query
13  #
14  # @param [String] create_schema_sql_query
15  # @return [Hash] the results of the query
16  # @raise [DatabaseSchemaError] if any MySQL errors are encountered
17  def create_schema(create_schema_sql_query)
18    @connection.execute(create_schema_sql_query)
19  rescue Mysql2::Error => exception
20    raise DatabaseSchemaError, exception.message
21  end
22
23  # Execute the seed SQL query
24  #
25  # @param [String] seed_sql_query
26  # @return [Hash] the results of the query
27  # @raise [DatabaseSeedError] if any MySQL errors are encountered
28  def seed_initial_data(seed_sql_query)
29    @connection.execute(seed_sql_query)
30  rescue Mysql2::Error => exception
31    raise DatabaseSeedError, exception.message
32  end
33
34  # Execute student's or instructors' query
35  #
36  # @param [String] sql_query
37  # @return [Hash] the results of the query
38  # @raise [DatabaseQueryExecutionFailed] if any MySQL errors are encountered
39  def execute_query(sql_query)
40    @connection.execute(sql_query)
41  rescue Mysql2::Error => exception
42    raise DatabaseQueryExecutionFailed, exception.message
43  end
44 end
45 end

```