REPORT

Our command line interface works based on the choices we enter the terminal. Each choice has a sub choice, which is explained below.

RETRIEVAL

This choice focuses on retrieving data from the database. The sub-choices provide a versatile set of options for obtaining specific information. Users can retrieve details about participants in a team, problems set by an author, teams that solved a particular problem, and more. The availability of options like finding teams with the most accepted submissions and calculating the average number of problems solved in each region enhances the utility of this choice. The sub-choices are described below.

1 Retrieve participants in a team.

```
SELECT * FROM participants
WHERE Team_Id = <team_id>
```

DESCRIPTION: Retrieve participants with Team_ID = <team_id> which exists in the participants table which has been populated beforehand.

2 Retrieve problems set by an author.

```
SELECT * FROM problem
WHERE Author_id = <author_id>
```

DESCRIPTION: Retrieve problems set by an author with Author_id=<author_id> existing in the problem relation.

3 Retrieve list of participants based on their university.

```
SELECT * FROM participants
WHERE University_ID = <university_id>
```

DESCRIPTION: Retrieve list of participants WITH their university_ID = <University_id> taken from the user input.

4 Retrieve list of teams who have solved a particular problem

```
SELECT DISTINCT(Team_id)
FROM submission
WHERE Problem_id = <problem_id>
```

DESCRIPTION: Retrieve list of teams who have solved a particular problem.

5 Retrieve all submissions in a coding language

```
SELECT * FROM submission
WHERE Programming_Language = '<programming_language>'
```

DESCRIPTION: Display all the submissions based in language

6 Find teams with the most accepted submissions.

```
SELECT Team_id, SUM(Acceptance) AS AC
FROM submission
GROUP BY Team_id
HAVING SUM(Acceptance) = (
    SELECT MAX(total_acceptance)
    FROM (
        SELECT Team_id, SUM(Acceptance) AS total_acceptance
        FROM submission
        GROUP BY Team_id
    ) AS max_acceptance
);
```

DESCRIPTION: Display teams with the most accepted submissions, we start by grouping the teams and then finding the number of accepted submissions per team.

7 Find average number of problems solved in each region.

```
SELECT AVG(total_problems_solved) AS avg_problems_solved
FROM (
SELECT t.Team_Id, COUNT(DISTINCT s.Problem_id) AS
total_problems_solved
FROM team t
LEFT JOIN submission s ON t.Team_Id = s.Team_id
WHERE t.Region_Id = {}
GROUP BY t.Team_Id
) AS team_problems;
```

DESCRIPTION: Average number of problems solved in each region (based on region_id). Obtained by joining the team and submission table and grouping based on team_id. The distinct records are then taken into consideration.

8 Find most solved problems:

```
SELECT Problem_id, COUNT(*) AS total_solutions
FROM submission
WHERE Acceptance = 1
GROUP BY Problem_id
ORDER BY total_solutions DESC
LIMIT 1;
```

DESCRIPTION: display the most solved problem by grouping based on problem_id and then sort the output based on the number of records per problem_id.

9 Count number of accepted submissions in a language

```
SELECT COUNT(*) AS total_ac
FROM submission
WHERE Programming_Language = '<language>' AND
Acceptance = 1;
```

DESCRIPTION: display the number of submissions in a language obtained by checking if the acceptance value is set and sorting the number of records from the submission table.

10 Search for team names containing a string

```
SELECT *
FROM team
WHERE Name LIKE '%<string>%';
```

DESCRIPTION: Display the teams which have <string> in their names using the ‘LIKE’ keyword.

11 Search for teams belonging to universities starting with string

```
SELECT *
FROM team
WHERE University_Id IN (
    SELECT University_Id
    FROM university
    WHERE name LIKE '<STRING>%'
);
```

DESCRIPTION: Display the universities with name staring with <STRING>

12 Search for mentors with their first name

```
SELECT DISTINCT Fname,Lname
FROM mentor
WHERE Fname LIKE '<fname>%';
```

DESCRIPTION: Display the name of mentors whose name is starting with <fname> using the 'LIKE' keyword.

MODIFICATION:

Choice 2 allows users to modify the database. Sub-choices include adding submissions and test cases, updating test cases, scores of a team, disqualifying a team, and removing low-rated teams. These functionalities empower users to actively manage and update the database content. The sub-choices are described below.

Sub-choices:

1 Add Submission

```
INSERT INTO submission (time_stamp, Team_id, Problem_id,
Programming_Language, Acceptance) VALUES
('<time_stamp>', <team_id>, <problem_id>, '<lang>',
<acceptance>);
```

DESCRIPTION: Add a new entry to the table with values ('<time_stamp>', <team_id>, <problem_id>, '<lang>', <acceptance>)

2 Add Test Case

```
INSERT INTO test_case (problem_id, testcase_id,
testcase) VALUES
(<problem_id>, <testcase_id>, '<testcase>');
```

DESCRIPTION: Add a new entry to the table with values(<problem_id>, <testcase_id>, '<testcase>') in the test_case table.

3 Update Test Case

```
UPDATE test_case
SET testcase = '<updated_testcase>'
WHERE problem_id = <problem_id> AND testcase_id =
<testcase_id>;
```

DESCRIPTION: Update the testcase with problem_id = <problem_id> AND testcase_id = <testcase_id> in the test_case table.

4 Update score of a team

```
UPDATE team
SET Score = (
SELECT COUNT(
  DISTINCT(Problem_id)
  FROM submission
  WHERE Team_id = <team_id> AND Acceptance = 1
)
WHERE Team_Id = <team_id>;
```

DESCRIPTION: Update the score of a team with new value where Team_id = <team_id> and having the acceptance value set, from the submission table.

5 Disqualify Team

```
DELETE FROM team
WHERE Team_Id = {team_id};
```

DESCRIPTION: Delete entry from the table having the team_id taken from input from the ‘team’ table.

6 Remove Low-Rated Team

```
DELETE FROM team
WHERE Avg_Rating < <avg_rating>;
```

DESCRIPTION: Delete entry from the table of the team having their average rating less than the overall average rating.

ANALYSIS:

This choice facilitates in-depth analysis of the database. Sub-choices include displaying the average penalty for every problem based on non-accepted submissions, generating rank lists of teams based on scores for each region, and retrieving the acceptance rate of problems created by an author. These analytical capabilities provide valuable insights into the performance and trends within the ICPC database. The sub-choices are described below.

Sub choices:

1 Display the average penalty for every problem based on their non-accepted submissions.

```
SELECT COALESCE(AVG(total_tries), 0) AS avg_penalty
 FROM (
     SELECT Team_id, COUNT(*) AS total_tries
     FROM submission
     WHERE Problem_id = <problem_id> AND Acceptance = 0
     GROUP BY Team_id
     HAVING SUM(Acceptance) > 0
 ) AS team_tries;
```

DESCRIPTION: get the average penalty for each team for the Problem_id = <problem_id> based on the submissions they have set which were not accepted per problem_id, after grouping we take their sum and display the output.

2 Generate a rank list of teams participating based on their scores (after penalty) for each region

```
SELECT ROW_NUMBER() OVER (
      ORDER BY COALESCE(SUM(s.Acceptance), 0)
      DESC
      ) AS team_rank,
t.Team_Id, t.Name,
COALESCE(SUM(s.Acceptance), 0) AS total_acceptance
FROM team t
LEFT JOIN submission s ON t.Team_Id = s.Team_id
WHERE t.Region_Id = <region_id> GROUP
BY t.Team_Id, t.Name
ORDER BY total_acceptance DESC;
```

DESCRIPTION: Get the rank lists of each team at the region <region_id> obtained based on accepted number of submissions.

3 Retrieve the acceptance rate of problems created by an author

```
SELECT p.Problem_id,
COUNT(s.Acceptance) AS total_submissions,
SUM(s.Acceptance) AS total_acceptances,
SUM(s.Acceptance)/COUNT(s.Acceptance) AS
acceptance_rate
FROM submission s
RIGHT JOIN problem p ON s.Problem_id = p.Problem_id
WHERE p.Author_Id = <author_id>
GROUP BY p.Problem_id
ORDER BY acceptance_rate DESC;
```

DESCRIPTION: Get the acceptance rate of the problem created by author = <author_id> obtained by joining submission on problems and then ordering on acceptance rate.

4. EXIT: A straightforward choice to exit the command-line interface, offering users a seamless way to conclude their interactions with our database management system.