Requirements Documentation

Intercollege CP Tournament Analysis Database

Data and Applications Project: Phase I

Amul Agrawal (2019101113)
Mihir Bani (2019113003)
Snehal Kumar (2019101003)

---

1. Description of the Mini World

Competitive programming (CP) is one of the most exciting exercises for programmers to showcase their cognitive and coding skills. This has led to an increasing number of competitions especially among colleges. A generic CP contest contains a set of problems to be solved by teams of programmers within a set duration of time.This database models the general platform and rules used to conduct a CP tournament allowing the admin to efficiently monitor the tournament. Using this database, one can perform common queries and use them for statistical analysis to improve both the contest and the users skills based on their performance.
This helps the programmers to build and improve on their skills on specific problems and coding.

Note: Using (min,max) notation for constraints. **Bold** texts are entities, *Italic* texts are attributes, and CAPS text are relationships.

2. Entities and their Attributes

- **Programmer**
    - *Programmer ID* [Key Attribute(Integer)]: Stores the registration ID of the player, unique for each.
    - *Name* [Composite Attribute (First Name, Last Name)(String max 50 chars, String max 50 chars)]:The full name of the player.
    - *Nationality* [String max 50 char]: Country represented by programmer.
    - *Age* [Integer (18-99)]: Age of programmer.
    - *Email id* [Key attribute (String max 50 char)]: Unique email-id of each programmer.
    - *Is_graduate* [Defining attribute (Bit value type{0/1})]: Whether programmer is a college graduate.

  The Programmer is divided into two following subclasses on the basis of *Is_graduate*.

- **Problem-setter (Subclass of Programmer) [**Is_graduate == 1**]**

- *Experience* [Integer (1-99)]: Number of years he has been writing problems.
- *No of problems* [Integer (1-1000)]: Total number of problems set by the problem setter in the past.

- **Contestant (Subclass of Programmer)** **[**Is_graduate == 0**]**
  - *College Year* [Integer (1-6)]: Represents the year the contestant is currently in . Eg- 1st year Undergrad.

- **Team**
  - *T-No* [Key attribute (Integer)]: Uniquely identifiable team number of participating team
  - *T-Name* [String 25 char max]: Name of participating team
  - *College Name* [String 75 char max]: Name of representing college

- **Contest**
  - *Contest-ID* [Key attribute(Integer)]: Uniquely identifies the Contest.
  - Start date [Date Time object]: The start date and time for Contest.
  - *Allowed languages* [Multivalued attribute(String 100 char max)]: List of allowed Programming languages in the contest.
  - *Duration* [Integer(1-500)]: Length of the contest in minutes.

- **Problem (Weak entity)**
  - *Prob-Number* [Partial Key (Integer (1-10))]: Linked to Contest-ID of **Contest,** Number of the problem. eg: Problem 4.2 (Contest-ID:4, Prob-Number:2)
  - *Tags* [Multivalued attribute (String 20 char max)]: Contains the tags related to the problem
  - *I/O Format* [String 500 char max]: Formatting of I/O required for the problem
  - *P-Statement* [String 5000 char max]: Problem statement
  - *Limits* [Composite attribute (Memory,Run time) (Integer (1-10000), Integer (1-100)]: The constraints of the problem with respect to memory and runtime.
  - *Max points* [Integer (1-5000)]: Maximum points for the problem on successful submission
  - *Sample I/O* [String 500 char max]: Sample Input and Output for the problem
  - *Test I/O* [Multivalued attribute (String 200 char max)]: Input and Output of test cases for the problem.

- **Submission**
  - *Sub-ID* [Key attribute (Integer)]: Uniquely identifies a submission.
  - *Language* [String 50 char max]: Programming language in which that submission was made.
  - *Sub-Time* [Date-Time type]: When that submission took place.

- **Result (Weak entity)**

- *Stats* [Partial Composite Key (Memory,Runtime) (Integer(1-1000000), Integer(1-5000))]: Statistics of the result containing the memory in KB used and execution time taken in milliseconds.
- *Verdict* [(String max 10 chars), can only be from {AC/WA/RE/ME/TLE}]: Outputs whether the result of the submission is correct/wrong/error.
- *Points* [Integer(1-5000)]: Points given as per result.

3. Relations

In the following points **Entity(N,M)** implies that there is a constraint of (N,M) between Entity and Relation.

- A **Team** CONTAINS **Contestant**(s). [changed to HAS]
  - Binary Relationship between **Team(3,3)** and **Contestant(1,1).**

- A **Team** PARTICIPATES in a **Contest.**
  - Binary Relationship between **Team(0,N)** and **Contest(1,M).**
  - Attributes:
    - *Total Score* [Integer]: Total score of the team in a contest.
    - *Rank* [Integer](Derived Attribute of Total Scores): Gives the rank of a team in a contest.

- A **Contest** CONTAINS **Problem**(s).
  - Binary Relationship between **Contest(1,10)** and **Problem(1,1)**.
  - It is the identifying relationship for **Problem.**

- A **Problem-setter** WRITES **Problem**(s).
  - Binary Relationship between **Problem-setter(0,L)** and **Problem(1,P)**.

- A **Team** SUBMITS a **Submission** *for* a **Problem** and *gets* **Result**.
  - 4-Nary Relationship using **Team(0,1)**, **Result(1,1), Problem(0,1)** and **Submission(0,N).**
  - Result is unique for (Team, Problem, Submission).
  - It is the identifying relationship for **Result.**

- A **Contestant** LEADS a **Team**.
  - Binary Relationship between **Contestant(0,1)** and **Team(1,1).**
  - Role Name: that contestant is called as ‘team leader’.

- A **Problem** IS_SUBPART of another **Problem**.
  - Unary Relationship between **Problem(0,1)** and **Problem(0,1).**
  - Role Name: One side is ‘parent problem’ and the other is ‘sub-problem’

4. Functional Requirements

4.1. Modification:

- Insert Operation
    1. **Programmer** entry on registration with its attributes
    2. **Team** entry with its attributes
    3. **Contest** entry on registering a new contest with all attributes
    4. **Problem** entries after entering contest

- Update Operation
    1. **Submission** is updated after **Team** submits solution for **problem**
    2. **Result** attributes are updated after each **Submission**
    3. **Team** *Score* is updated after result
    4. *Rank* of a **Team** is determined by *Total Score* of PARTICIPATES.

- Delete Operation
    1. Delete **Team** entry if team unable to participate
    2. Delete **Contest** entry if contest is cancelled
    3. Delete **Problem** entry in case of error in problem

4.2. Retrieval:

- Selection Query
    1. **List all teams participating in a contest**
    (SELECT Tno,TName FROM PARTICIPATES NATURAL JOIN TEAM WHERE ContestID= $ContestID)
- Projection Query
    1. **List all successful submissions of a team in a contest (Using *Verdict* attribute of Result)**
    (SELECT * FROM SUBMISSION NATURAL JOIN SUBMITS WHERE VERDICT=”AC”)
    2. **List team with points > 500 (**Using *Total Score* attribute)
    CREATE VIEW TEAM_SUBMIT AS SELECT TNo,ContestID,SubID,Points FROM SUBMITS NATURAL JOIN SUBMISSION
    SELECT * FROM TEAM_SUBMIT WHERE Points>500
- Aggregation Query
    1. **Average time spent on each problem. (Using *Sub-Time* attribute of Submission and *Start Date* of Contest) Shows the correlation between the difficulty of the problem and time spent.**
    (inner join SUBMITS,SUBMISSION, group by problemname, avg(subtime)
    (SELECT A.ProblemName, AVG(SubTime) AS AverageTimeSpent FROM SUBMITS A INNER JOIN SUBMISSION B ON A.SubID = B.SubID WHERE A.Verdict=’AC’ GROUP BY A.ContestID,A.ProblemName)
- Search Query

1. **Search for contestant names and team names.**
   SELECT * FROM PROGRAMMER WHERE
   SELECT * FROM TEAM WHERE TName LIKE %$NAME%

- Analysis
  1. **List the first team to solve each problem. Gives incentive to programmers to think and code faster and efficiently.**
     SELECT A.TNo, A.ProblemName, MIN(SubTime) AS TimeSpent FROM SUBMITS A INNER JOIN SUBMISSION B ON A.SubID = B.SubID WHERE A.Verdict='AC' GROUP BY A.ContestID, A.TNo, A.ProblemName
  2. **Ratio of successful submissions to unsuccessful attempts. This reveals accuracy of team submission.**
     SELECT A.TNo, COUNT(SubID) AS Successful WHERE VERDICT='AC', COUNT(SubID) AS Unsuccessful WHERE VERDICT!='AC' FROM SUBMITS A INNER JOIN SUBMISSION B ON A.SubID = B.SubID GROUP BY A.ContestID, A.TNo
  3. **Toughest problem-setter (Join problem setter to his problems with lowest successful submissions)**

```
#standings
CREATE VIEW POINT_TABLE AS
        SELECT TNo, ContestID,ProblemName, SubID, Points FROM SUBMITS NATURAL
        JOIN SUBMISSION WHERE VERDICT="AC" AND ContestID= $ContestID;
SELECT TNo, SUM(Points) AS TOTALSCORE FROM POINT_TABLE GROUP BY TNo
ORDER BY TOTALSCORE  DESC;



CREATE VIEW COUNT_SUCCESS AS
(
        SELECT A.ContestID, A.TNo, COUNT(SubID) AS Successful FROM SUBMITS A
        NATURAL JOIN SUBMISSION B WHERE VERDICT='AC' GROUP BY A.ContestID,
        A.TNo
);
CREATE VIEW COUNT_FAIL AS
(
        SELECT A.ContestID, A.TNo, COUNT(SubID) AS Unsuccessful FROM SUBMITS A
        NATURAL JOIN SUBMISSION B WHERE VERDICT != 'AC' GROUP BY A.ContestID,
        A.TNo
);
```

```
SELECT A.ContestID, A.TNo, A.Successful , B.Unsuccessful ,
A.Successful/(B.Unsuccessful+A.Successful) AS ACCURACY FROM  COUNT_SUCCESS A
NATURAL JOIN COUNT_FAIL B ;
```

##

```
INSERT INTO CONTEST VALUES
(1, '2020-01-01 18:00:00', '02:00:00'),
(2,'2019-08-01 15:00:00','02:00:00'),
(3,'2020-01-06 16:00:00', '02:00:00') ;

INSERT INTO CONTESTLANGUAGES VALUES
(1, 'C++'), (1, 'C'), (1, 'Python'),
(2,'JAVA'), (2,'Python'), (2,'C++'), (2,'RUBY''),
(3,'C'), (3,'C#'), (3,'C++'), (3,'RUBY');


INSERT INTO TEAM VALUES
(1,'3NOOBS','IIIT Hyderabad',1),
(2,'HighHopes','IIIT Hyderabad',4),
(3,Manforce, 'IIT Hyderabad',7);

INSERT INTO PARTICIPATES VALUES
(1,1),(2,1),(3,1),(1,2),(2,2),(3,2),(2,3),(3,3);


INSERT INTO PROGRAMMER VALUES
(1,'numberdedopls@gmail.com', 'Bhavyajeet' ,Singh,'India','20','0'),
(2,'sirplsgivemarks@gmail.com', 'Jaidev', 'Sriram','India',20,'0'),
(3,'maampls@gmail.com', 'Jyoti' , 'Sunkara','India','20','0'),
(4,'Charaswati@gmail.com', 'Ashish' , 'Gupta','India','19','0'),
(5,'ArnabGoswami@gmail.com','Rutvij', 'Menavlikar','India','19','0'),
(6,'BONDOP@gmail.com', 'Anvay'' , 'Karmore','India','19','0'),
(7,'multiplexer@gmail.com', 'Tejas' , 'Chaudhari','India','19','0'),
(8,'sidthesloth@gmail.com', 'Jaywant' , 'Patel','India','12','0'),
(9,'randibaaz@gmail.com', 'Aditya' , 'Verma','India','19','0'),
(10,'tyrantmf@gmail.com', 'Xi' ,'Jinping','China','29','1'),
(11,'momowala@gmail.com', 'Kim','Jong-Un','North Korea','25','1'),
(12,'omshaantiom@jaishreeram.com', 'Dalai','Lama','Nepal','35','1'),
(13,'orangeblob@whodoesntliketheirdaughtersass.com', 'Donald','Trump','USA','69','1');

INSERT INTO PROBLEM VALUES
(1,'A',500,'Problem statement for problem A contest 1','sample test','IOFORMAT','sample
io',500,'00:30:00'),
```

(1,’B’,750,’Problem statement for problem B contest 1’,’sample test’,’IOFORMAT’,’sample io’,500,’00:40:00’),
(1,’C’,1000,’Problem statement for problem C contest 1’,’sample test’,’IOFORMAT’,’sample io’,500,’00:50:00’),
(1,’D’,2000,’Problem statement for problem D contest 1’,’sample test’,’IOFORMAT’,’sample io’,500,’00:10:00’),
(2,’A’,500,’Problem statement for problem A contest 2’,’sample test’,’IOFORMAT’,’sample io’,500,’00:30:00’),
(2,’B’,750,’Problem statement for problem B contest 2’,’sample test’,’IOFORMAT’,’sample io’,500,’00:40:00’);
(3,’A’,500,’Problem statement for problem A contest 3’,’sample test’,’IOFORMAT’,’sample io’,500,’00:10:00’),
(3,’B’,750,’Problem statement for problem B contest 3’,’sample test’,’IOFORMAT’,’sample io’,500,’00:20:00’);