Datenbankschema: Design

NvG, FH, SR

1 relationales Schema

```
See /target/classes/hibernate3/sql/create.sql
```

2 Stored procedure

```
1
2  --
3  -- Erstellt einen Turnierplan
4  --
5  DROP FUNCTION createChampionship(int, text);
6  CREATE OR REPLACE FUNCTION createChampionship(int, text) RETURNS VOID AS
7  $$
8  DECLARE
9      yearParam ALIAS FOR $1;
10     nameParam ALIAS FOR $2;
11     host country%ROWTYPE;
12     currentStadium stadium%ROWTYPE;
13     groupStage groupstage%ROWTYPE;
14     finalId bigint;
15 BEGIN
16     RAISE NOTICE 'Creating a new tournament';
17
18   --Generiert die K.O.-Phase
19     finalId := getNextSequence();
20     INSERT INTO match(id, name, played, dtype)
21     VALUES (finalId, 'Finale', false, 'KnockoutMatch');
22     PERFORM generateKnockoutTree(1, finalId);
23
24     --Generiert die Gruppenphase
25     groupStage := generateGroupStage();
26
27     --Speichert das Turnier ab
28     INSERT INTO tournament (year, name, finalmatch_id, groupstage_id)
29     VALUES (yearParam, nameParam, finalId, groupStage.id);
30
31     -- Set a random host country
32     host := getCountry();
33     INSERT INTO "tournament_country" VALUES (yearParam, host.id);
34
35     -- Set 8 random stadiums
36     FOR currentStadium IN SELECT * FROM getStadiumsForCountry(host.id) LOOP
37         INSERT INTO tournament_stadium VALUES(yearParam,currentStadium.stadiumid);
38     END LOOP;
39   RETURN;
40 END;
41 $$
42 LANGUAGE 'plpgsql';
43
```

```
44
45
46 --
47 -- Generiert rekursiv alle Finalspiele
48 --
49 CREATE OR REPLACE FUNCTION generateKnockoutTree(int, bigint) RETURNS VOID AS
50 $$
51 DECLARE
52     height ALIAS FOR $1;
53     nodeId ALIAS FOR $2;
54     matchId1 bigint;
55     matchId2 bigint;
56     newHeight int;
57     knockoutMatchType varchar;
58 BEGIN
59   --Rekursionsanker
60     IF(height > 3) THEN
61         RETURN;
62     ELSIF (height = 1) THEN
63         knockoutMatchType := 'Halbfinale';
64     ELSIF (height = 2) THEN
65         knockoutMatchType := 'Viertelfinale';
66     ELSIF (height = 3) THEN
67         knockoutMatchType := 'Achtelfinale';
68     END IF;
69
70
71   -- Erstellen zweier Kindspiele
72     matchId1 := getNextSequence();
73     INSERT INTO match(id, name, played, dtype)
74     VALUES (matchId1, knockoutMatchType, false, 'KnockoutMatch');
75
76     matchId2 := getNextSequence();
77     INSERT INTO match(id, name, played, dtype)
78     VALUES (matchId2, knockoutMatchType, false, 'KnockoutMatch');
79
80   -- Hinzufuegen zum Baum
81     INSERT INTO match_match(match_id, childs_id) VALUES (nodeId, matchId1);
82     INSERT INTO match_match(match_id, childs_id) VALUES (nodeId, matchId2);
83
84   -- rekursiver Aufruf
85     newHeight := height + 1;
86     PERFORM generateKnockoutTree(newHeight, matchId1);
87     PERFORM generateKnockoutTree(newHeight, matchId2);
88
89     RETURN;
90 END;
91 $$
92 LANGUAGE 'plpgsql';
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107 --
108 -- Erstellt ein DummyLand.
109 -- Falls es schon vorhanden ist wird nur dieses zurueckgegeben.
110 --
111 CREATE OR REPLACE FUNCTION getCountry() RETURNS SETOF Country AS
```

```
$$
DECLARE
    selectedRow Country%ROWTYPE;
    n int := 0;
BEGIN
    SELECT COUNT(*) INTO n FROM Country;
    IF(n < 1) THEN
        INSERT INTO Country VALUES (getNextSequence(), 'DummyLand');
    END IF;

    SELECT * INTO selectedRow FROM Country ORDER BY RANDOM() LIMIT 1;
    RETURN NEXT selectedRow;
END
$$ LANGUAGE 'plpgsql';









--
-- String Concatination Helper
--
CREATE OR REPLACE FUNCTION concat(VARCHAR, INT) RETURNS VARCHAR AS
$$
BEGIN
    return $1 || ' ' || chr(49 + ($2%119));
END
$$ language 'plpgsql';

--
-- String Concatination Helper
--
CREATE OR REPLACE FUNCTION concat(VARCHAR, BIGINT) RETURNS VARCHAR AS
$$
BEGIN
    return $1 || ' ' || chr(CAST(49 + ($2%119) AS INT));
END
$$ language 'plpgsql';









--
-- Gibt 8 Stadien (aus dem gegebenen Land) zurueck.
-- Falls nicht genuegend existieren werden welche erstellt.
--
CREATE OR REPLACE FUNCTION getStadiumsForCountry(bigint) RETURNS SETOF stadium AS
$$
DECLARE
    countryId ALIAS FOR $1;
    selectedRow Stadium%ROWTYPE;
    n int := 0;
    i int;
BEGIN
```

```
180     SELECT COUNT(*) INTO n FROM Stadium WHERE country_id = countryId;
181     IF(n < 8) THEN
182         FOR i IN 1..(8-n) LOOP
183             INSERT INTO Stadium VALUES (getNextSequence(), 500, concat('Dummystadt',i) ,
                    concat('Dummystadion',i), countryId);
184         END LOOP;
185     END IF;
186
187     FOR selectedRow IN SELECT * FROM stadium ORDER BY RANDOM() LIMIT 8 LOOP
188         return next selectedRow;
189     END LOOP;
190
191     return;
192 END
193 $$ LANGUAGE 'plpgsql';
194
195
196
197
198
199
200 --
201 -- Hilfsfunktion um den Primaerschluessel fuer die Relationen zu ermitteln
202 --
203 CREATE OR REPLACE FUNCTION getNextSequence() RETURNS bigint AS
204 $$
205     SELECT nextval('hibernate_sequence') FROM hibernate_sequence;
206 $$ LANGUAGE 'sql';
207
208
209
210
211
212
213
214
215 --
216 -- Erstellt ein Team mit 23 Spielern
217 --
218 CREATE OR REPLACE FUNCTION generateTeam() RETURNS team AS
219 $$
220 DECLARE
221     i int;
222     j int;
223     sequenceValue int;
224     playerId int;
225     selectedTeam Team%ROWTYPE;
226 BEGIN
227     SELECT id INTO i FROM getCountry();
228
229     sequenceValue := getNextSequence();
230
231     INSERT INTO team VALUES (sequenceValue, concat('Musterteam ', sequenceValue), i);
232     SELECT * INTO selectedTeam FROM team WHERE id = sequenceValue;
233
234     FOR j IN 1..23 LOOP
235         SELECT id INTO playerId FROM getPlayer();
236         INSERT INTO team_player VALUES (selectedTeam.id, playerId);
237         INSERT INTO person_team VALUES (playerId, selectedTeam.id);
238     END LOOP;
239
240     return selectedTeam;
241 END
242 $$ LANGUAGE 'plpgsql';
243
244
245
246
```

```
247
248
249
250
251
252 --
253 -- Erstellt einen neuen Spieler
254 --
255 CREATE OR REPLACE FUNCTION getPlayer() RETURNS player AS
256 $$
257 DECLARE
258     createdPlayer Player%ROWTYPE;
259     sequenceValue bigint;
260 BEGIN
261     sequenceValue := getNextSequence();
262
263     INSERT INTO person (id, firstname, lastname)
264     VALUES (sequenceValue, concat('Vorname', sequenceValue), concat('Nachname',
            sequenceValue));
265
266     INSERT INTO player (id, nickname, club)
267     VALUES (sequenceValue, concat('Nick', sequenceValue), 'FC Seehaeusl');
268
269     SELECT * INTO createdPlayer FROM player WHERE id = sequenceValue;
270
271     return createdPlayer;
272 END
273 $$ LANGUAGE 'plpgsql';
274
275
276
277
278
279 --
280 -- Erstellt alle Gruppenspiele fuer eine gegeben Gruppe
281 --
282 CREATE OR REPLACE FUNCTION generateGroupMatches(BIGINT) RETURNS VOID AS
283 $$
284 DECLARE
285     groupId ALIAS FOR $1;
286     numberOfTeams int;
287     currentTeam team%ROWTYPE;
288     teams team[];
289     i int;
290     j int;
291 BEGIN
292
293     SELECT COUNT(*) INTO numberOfTeams
294     FROM tournamentgroup_team
295     WHERE tournamentgroup_groupid = groupId;
296
297   -- Test ob genuegend Teams in der Gruppe sind
298     if(numberOfTeams < 4) THEN
299         RAISE EXCEPTION 'at least 4 teams have to be in a group';
300         RETURN;
301     END IF;
302
303
304     -- Erstellt ein Array aus dem Teams der Gruppe
305     teams := '{}';
306     FOR currentTeam IN
307         SELECT t.*
308         FROM team t
309         JOIN tournamentgroup_team g ON (g.teams_id = t.id)
310         WHERE tournamentgroup_groupid = groupId
311     LOOP
312         teams := array_append(teams, currentTeam);
313     END LOOP;
```

```
314
315   -- Laesst jede Mannschaft einmal gegen alle anderen Manschaften antreten
316     FOR i IN 1..4 LOOP
317         FOR j IN (i+1)..4 LOOP
318             PERFORM generateMatch(teams[i].id, teams[j].id, groupId);
319         END LOOP;
320     END LOOP;
321
322     return;
323
324 END
325 $$ LANGUAGE 'plpgsql';
326
327
328
329 --
330 -- Erstellt ein noch nicht gespieltes Gruppenspiel für zwei Mannschaften
331 --
332 CREATE OR REPLACE FUNCTION generateMatch(bigint, bigint, bigint) RETURNS VOID AS
333 $$
334 DECLARE
335     hostTeam ALIAS FOR $1;
336     guestTeam ALIAS FOR $2;
337     groupId ALIAS FOR $3;
338     matchId bigint;
339     i int;
340 BEGIN
341     matchId := getNextSequence();
342
343     INSERT INTO match(id, hostteam_id, guestteam_id, played, dtype, group_groupid)
344     VALUES (matchId, hostTeam, guestTeam, false,'GroupMatch', groupId);
345
346     INSERT INTO tournamentgroup_match
347     VALUES (groupId, matchId);
348 END
349 $$ LANGUAGE 'plpgsql';
350
351
352 --
353 -- Generiert die Gruppenphase
354 -- Es werden 8 Gruppen mit jeweils 4 Mannschaften erstellt
355 --
356 CREATE OR REPLACE FUNCTION generateGroupStage() RETURNS GroupStage AS
357 $$
358 DECLARE
359     stageId int;
360     stage groupstage;
361     currentTeam team;
362     currentGroup tournamentgroup;
363     currentGroupId bigint;
364     i int;
365     j int;
366 BEGIN
367     stageId := getNextSequence();
368
369     INSERT INTO groupstage VALUES (stageId);
370
371     -- Fuer alle 8 Gruppen
372     FOR i IN 1..8 LOOP
373         currentGroupId := getNextSequence();
374
375         INSERT INTO tournamentgroup (groupid, name)
376         VALUES (currentGroupId, concat('Gruppe', 10));
377
378         -- generiere 4 Mannschaften
379         FOR j IN 1..4 LOOP
380             currentTeam := generateTeam();
381
```

```
382             INSERT INTO tournamentgroup_team (tournamentgroup_groupid, teams_id)
383             VALUES (currentGroupId, currentTeam.id);
384         END LOOP;
385
386         INSERT INTO groupstage_tournamentgroup VALUES (stageId, currentGroupId);
387
388     -- und trage die Gruppenspiele ein
389         PERFORM generateGroupMatches(currentGroupId);
390
391     END LOOP;
392
393     SELECT * INTO stage FROM groupstage WHERE id = stageId;
394
395     return stage;
396
397 END
398 $$ LANGUAGE 'plpgsql';
```