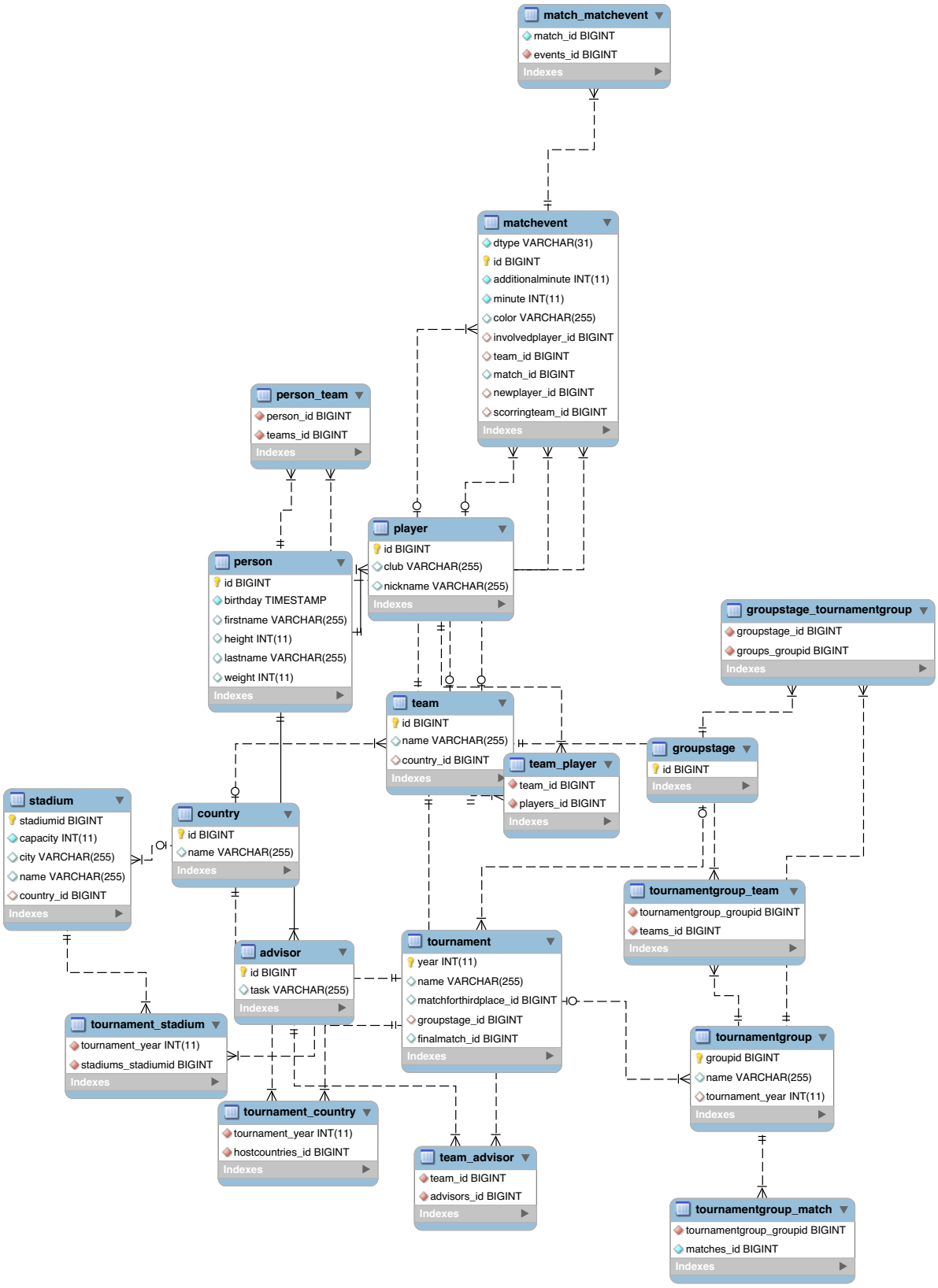
1 relationales Schema

1.1 Grafisches Model



2 SQL

```
1
2  --
3  -- Erstellt einen Turnierplan
4  --
5  CREATE OR REPLACE FUNCTION createChampionship(int, text) RETURNS VOID AS
6  $$
7  DECLARE
8      yearParam ALIAS FOR $1;
9      nameParam ALIAS FOR $2;
10     host country%ROWTYPE;
11     currentStadium stadium%ROWTYPE;
12     groupStage groupstage%ROWTYPE;
13     finalId bigint;
14 BEGIN
15     RAISE NOTICE 'Creating a new tournament';
16
17   --Generiert die K.O.-Phase
18     finalId := getNextSequence();
19     INSERT INTO match(id, name, played, dtype)
20     VALUES (finalId, 'Finale', false, 'KnockoutMatch');
21     PERFORM generateKnockoutTree(1, finalId);
22
23     --Generiert die Gruppenphase
24     groupStage := generateGroupStage();
25
26     --Speichert das Turnier ab
27     INSERT INTO tournament (year, name, finalmatch_id, groupstage_id)
28     VALUES (yearParam, nameParam, finalId, groupStage.id);
29
30     -- Set a random host country
31     host := getCountry();
32     INSERT INTO "tournament_country" VALUES (yearParam, host.id);
33
34     -- Set 8 random stadiums
35     FOR currentStadium IN SELECT * FROM getStadiumsForCountry(host.id) LOOP
36         INSERT INTO tournament_stadium VALUES(yearParam,currentStadium.stadiumid);
37     END LOOP;
38   RETURN;
39 END;
40 $$
41 LANGUAGE 'plpgsql';
42
43
44
45 --
46 -- Generiert rekursiv alle Finalspiele
47 --
48 CREATE OR REPLACE FUNCTION generateKnockoutTree(int, bigint) RETURNS VOID AS
49 $$
50 DECLARE
51     height ALIAS FOR $1;
52     nodeId ALIAS FOR $2;
53     matchId1 bigint;
54     matchId2 bigint;
55     newHeight int;
56     knockoutMatchType varchar;
57 BEGIN
58   --Rekursionsanker
59     IF(height > 3) THEN
60         RETURN;
61     ELSIF (height = 1) THEN
62         knockoutMatchType := 'Halbfinale';
63     ELSIF (height = 2) THEN
64         knockoutMatchType := 'Viertelfinale';
65     ELSIF (height = 3) THEN
```

```
66          knockoutMatchType := 'Achtelfinale';
67      END IF;
68
69
70      -- Erstellen zweier Kindspiele
71      matchId1 := getNextSequence();
72      INSERT INTO match(id, name, played, dtype)
73      VALUES (matchId1, knockoutMatchType, false, 'KnockoutMatch');
74
75      matchId2 := getNextSequence();
76      INSERT INTO match(id, name, played, dtype)
77      VALUES (matchId2, knockoutMatchType, false, 'KnockoutMatch');
78
79      -- Hinzufuegen zum Baum
80      INSERT INTO match_match(match_id, childs_id) VALUES (nodeId, matchId1);
81      INSERT INTO match_match(match_id, childs_id) VALUES (nodeId, matchId2);
82
83      -- rekursiver Aufruf
84      newHeight := height + 1;
85      PERFORM generateKnockoutTree(newHeight, matchId1);
86      PERFORM generateKnockoutTree(newHeight, matchId2);
87
88      RETURN;
89  END;
90  $$
91  LANGUAGE 'plpgsql';
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106 --
107 -- Erstellt ein DummyLand.
108 -- Falls es schon vorhanden ist wird nur dieses zurueckgegeben.
109 --
110 CREATE OR REPLACE FUNCTION getCountry() RETURNS SETOF Country AS
111 $$
112 DECLARE
113      selectedRow Country%ROWTYPE;
114      n int := 0;
115 BEGIN
116      SELECT COUNT(*) INTO n FROM Country;
117      IF(n < 1) THEN
118          INSERT INTO Country VALUES (getNextSequence(), 'DummyLand');
119      END IF;
120
121      SELECT * INTO selectedRow FROM Country ORDER BY RANDOM() LIMIT 1;
122      RETURN NEXT selectedRow;
123 END
124 $$ LANGUAGE 'plpgsql';
125
126
127
128
129
130
131
132
133
```

```
134
135
136 --
137 -- String Concatination Helper
138 --
139 CREATE OR REPLACE FUNCTION concat(VARCHAR, INT) RETURNS VARCHAR AS
140 $$
141 BEGIN
142     return $1 || ' ' || chr(49 + ($2%119));
143 END
144 $$ language 'plpgsql';
145
146 --
147 -- String Concatination Helper
148 --
149 CREATE OR REPLACE FUNCTION concat(VARCHAR, BIGINT) RETURNS VARCHAR AS
150 $$
151 BEGIN
152     return $1 || ' ' || chr(CAST(49 + ($2%119) AS INT));
153 END
154 $$ language 'plpgsql';
155
156
157
158
159
160
161
162
163
164
165
166
167 --
168 -- Gibt 8 Stadien (aus dem gegebenen Land) zurueck.
169 -- Falls nicht genuegend existieren werden welche erstellt.
170 --
171 CREATE OR REPLACE FUNCTION getStadiumsForCountry(bigint) RETURNS SETOF stadium AS
172 $$
173 DECLARE
174     countryId ALIAS FOR $1;
175     selectedRow Stadium%ROWTYPE;
176     n int := 0;
177     i int;
178 BEGIN
179     SELECT COUNT(*) INTO n FROM Stadium WHERE country_id = countryId;
180     IF(n < 8) THEN
181         FOR i IN 1..(8-n) LOOP
182             INSERT INTO Stadium VALUES (getNextSequence(), 500, concat('Dummystadt',i) ,
                    concat('Dummystadion',i), countryId);
183         END LOOP;
184     END IF;
185
186     FOR selectedRow IN SELECT * FROM stadium ORDER BY RANDOM() LIMIT 8 LOOP
187         return next selectedRow;
188     END LOOP;
189
190     return;
191 END
192 $$ LANGUAGE 'plpgsql';
193
194
195
196
197
198
199 --
200 -- Hilfsfunktion um den Primaerschluessel fuer die Relationen zu ermitteln
```

```
201 --
202 CREATE OR REPLACE FUNCTION getNextSequence() RETURNS bigint AS
203 $$
204     SELECT nextval('hibernate_sequence') FROM hibernate_sequence;
205 $$ LANGUAGE 'sql';
206
207
208
209
210
211
212
213
214 --
215 -- Erstellt ein Team mit 23 Spielern
216 --
217 CREATE OR REPLACE FUNCTION generateTeam() RETURNS team AS
218 $$
219 DECLARE
220     i int;
221     j int;
222     sequenceValue int;
223     playerId int;
224     selectedTeam Team%ROWTYPE;
225 BEGIN
226     SELECT id INTO i FROM getCountry();
227
228     sequenceValue := getNextSequence();
229
230     INSERT INTO team VALUES (sequenceValue, concat('Musterteam ', sequenceValue), i);
231     SELECT * INTO selectedTeam FROM team WHERE id = sequenceValue;
232
233     FOR j IN 1..23 LOOP
234         SELECT id INTO playerId FROM getPlayer();
235         INSERT INTO team_player VALUES (selectedTeam.id, playerId);
236         INSERT INTO person_team VALUES (playerId, selectedTeam.id);
237     END LOOP;
238
239     return selectedTeam;
240 END
241 $$ LANGUAGE 'plpgsql';
242
243
244
245
246
247
248
249
250
251 --
252 -- Erstellt einen neuen Spieler
253 --
254 CREATE OR REPLACE FUNCTION getPlayer() RETURNS player AS
255 $$
256 DECLARE
257     createdPlayer Player%ROWTYPE;
258     sequenceValue bigint;
259 BEGIN
260     sequenceValue := getNextSequence();
261
262     INSERT INTO person (id, firstname, lastname)
263     VALUES (sequenceValue, concat('Vorname', sequenceValue), concat('Nachname',
            sequenceValue));
264
265     INSERT INTO player (id, nickname, club)
266     VALUES (sequenceValue, concat('Nick', sequenceValue), 'FC Seehaeusl');
267
```

```
268     SELECT * INTO createdPlayer FROM player WHERE id = sequenceValue;
269
270     return createdPlayer;
271 END
272 $$ LANGUAGE 'plpgsql';
273
274
275
276
277
278 --
279 -- Erstellt alle Gruppenspiele fuer eine gegeben Gruppe
280 --
281 CREATE OR REPLACE FUNCTION generateGroupMatches(BIGINT) RETURNS VOID AS
282 $$
283 DECLARE
284     groupId ALIAS FOR $1;
285     numberOfTeams int;
286     currentTeam team%ROWTYPE;
287     teams team[];
288     i int;
289     j int;
290 BEGIN
291
292     SELECT COUNT(*) INTO numberOfTeams
293     FROM tournamentgroup_team
294     WHERE tournamentgroup_groupid = groupId;
295
296   -- Test ob genuegend Teams in der Gruppe sind
297     if(numberOfTeams < 4) THEN
298         RAISE EXCEPTION 'at least 4 teams have to be in a group';
299         RETURN;
300     END IF;
301
302
303     -- Erstellt ein Array aus dem Teams der Gruppe
304     teams := '{}';
305     FOR currentTeam IN
306         SELECT t.*
307         FROM team t
308         JOIN tournamentgroup_team g ON (g.teams_id = t.id)
309         WHERE tournamentgroup_groupid = groupId
310     LOOP
311         teams := array_append(teams, currentTeam);
312     END LOOP;
313
314   -- Laesst jede Mannschaft einmal gegen alle anderen Manschaften antreten
315     FOR i IN 1..4 LOOP
316         FOR j IN (i+1)..4 LOOP
317             PERFORM generateMatch(teams[i].id, teams[j].id, groupId);
318         END LOOP;
319     END LOOP;
320
321     return;
322
323 END
324 $$ LANGUAGE 'plpgsql';
325
326
327
328 --
329 -- Erstellt ein noch nicht gespieltes Gruppenspiel für zwei Mannschaften
330 --
331 CREATE OR REPLACE FUNCTION generateMatch(bigint, bigint, bigint) RETURNS VOID AS
332 $$
333 DECLARE
334     hostTeam ALIAS FOR $1;
335     guestTeam ALIAS FOR $2;
```

```
336     groupId ALIAS FOR $3;
337     matchId bigint;
338     i int;
339 BEGIN
340     matchId := getNextSequence();
341 
342     INSERT INTO match(id, hostteam_id, guestteam_id, played, dtype,  group_groupid)
343     VALUES (matchId, hostTeam, guestTeam, false,'GroupMatch', groupId);
344 
345     INSERT INTO tournamentgroup_match
346     VALUES (groupId, matchId);
347 END
348 $$ LANGUAGE 'plpgsql';
349 
350 
351 --
352 -- Generiert die Gruppenphase
353 -- Es werden 8 Gruppen mit jeweils 4 Mannschaften erstellt
354 --
355 CREATE OR REPLACE FUNCTION generateGroupStage() RETURNS GroupStage AS
356 $$
357 DECLARE
358     stageId int;
359     stage groupstage;
360     currentTeam team;
361     currentGroup tournamentgroup;
362     currentGroupId bigint;
363     i int;
364     j int;
365 BEGIN
366     stageId := getNextSequence();
367 
368     INSERT INTO groupstage VALUES (stageId);
369 
370     -- Fuer alle 8 Gruppen
371     FOR i IN 1..8 LOOP
372         currentGroupId := getNextSequence();
373 
374         INSERT INTO tournamentgroup (groupid, name)
375         VALUES (currentGroupId, concat('Gruppe', 10));
376 
377         -- generiere 4 Mannschaften
378         FOR j IN 1..4 LOOP
379             currentTeam := generateTeam();
380 
381             INSERT INTO tournamentgroup_team (tournamentgroup_groupid, teams_id)
382             VALUES (currentGroupId, currentTeam.id);
383         END LOOP;
384 
385         INSERT INTO groupstage_tournamentgroup VALUES (stageId, currentGroupId);
386 
387     -- und trage die Gruppenspiele ein
388         PERFORM generateGroupMatches(currentGroupId);
389 
390     END LOOP;
391 
392     SELECT * INTO stage FROM groupstage WHERE id = stageId;
393 
394     return stage;
395 
396 END
397 $$ LANGUAGE 'plpgsql';
```