PROJECT CHECKTPOINT 3

Architecture and management of database

Antoni Forzpanczyk a2019156557
Jędrzej Szor a2019156734

Exceptions

We provided additional package called EXCEPTIONS which stores all codes of required exceptions to be thrown. In addition package has implemented method RAISE_EXCEPRION which invokes built-In RAISE_APPLIACTION_ERROR with passed code and message obtained by GET_MESSAGE local function.

```
1. create PACKAGE EXCEPTIONS AS
2.     no_team number := -20501;
3.     no_game number := -20502;
4.     no_bettor number := -20503;
5.     no_bet number := -20504;
6.     no_competition number := -20505;
7.     invalid_bet_type number := -20506;
8.     invalid_bet_type_game number := -20507;
9.     no_money number := -20508;
10.      game_closed number := -20509;
11.      invalid_year number := -20510;
12.      invalid_phase_game number := -20511;
13.      invalid_phase_date number := -20512;
14.      duplicate_game number := -20513;
15.      duplicate_odds number := -20514;
16.      invalid_event_type number := -20515;
17.      negative_bet number := -20516;
18.      game_not_over number := -20517;
19.      prizes_paid number := -20518;
20.
21.  --          own
22.      invalid_charge number := -20519;
23.
24.      procedure raise_exception(p_exception_code number);
25.
26.  END EXCEPTIONS;
```

```
1.
2. create PACKAGE BODY EXCEPTIONS AS
3.     --
4.     function get_message(p_exception_code number) return varchar2 is
5.     begin
6.         if p_exception_code = -20501 then
7.             return 'Team does not exist.';
8.         elsif p_exception_code = -20502 then
9.             return 'Game does not exist.';
10.          elsif p_exception_code = -20503 then
11.              return 'Bettor does not exist.';
12.  ...
13.          else
14.              return 'message not found';
15.          end if;
16.      end get_message;
17.
18.      procedure raise_exception(p_exception_code number) is
19.      begin
20.          RAISE_APPLICATION_ERROR(p_exception_code,
21.                                  get_message(p_exception_code));
22.      end;
23.
24.  END EXCEPTIONS;
```

Tasks

a) Create the nGoals function, which takes the code of a game and the name of a team as an argument and returns the number of goals scored by that team in that game. The function can throw the following exceptions: -20502 and -20501.

```
1. create function a_nGoals(p_game_id games.game_id%type,
2.                          p_team_id teams.team_id%type)
3.     return number is
4.
5.     v_goals_a history_games.a_goals%type := 0;
6.     v_goals_b history_games.b_goals%type := 0;
7.     v_team_a  teams.team_id%type;
8.     v_team_b  teams.team_id%type;
9.     v_count   number;
10.
11.  begin
12.      select count(*)
13.      into v_count
14.      from history_games
15.      where game_id = p_game_id;
16.
17.      if v_count = 0 then
18.          exceptions.RAISE_EXCEPTION(EXCEPTIONS.no_game);
19.      end if;
20.
21.      select h.a_goals, h.b_goals, h.a_team_id, h.b_team_id
22.      into v_goals_a, v_goals_b, v_team_a, v_team_b
23.      from history_games h
24.      where h.game_id = p_game_id;
25.
26.      if v_team_a = p_team_id then
27.          return v_goals_a;
28.      elsif v_team_b = p_team_id then
29.          return v_goals_b;
30.      else
31.          exceptions.RAISE_EXCEPTION(EXCEPTIONS.no_team);
32.          return 0;
33.      end if;
34.  end;
35.  /
36.
```

b) Create the nGamesBetweenTeams function, which takes as an argument the name of two teams (the visited team and the visiting team) and optionally the number of years (lastNYears), by default it must consider 5 years, and which returns the number of games that occurred between those 2 teams in the last lastNYears years. The function can throw the following exceptions: -20501 and -20510.

```
1. create function b_nGames_Between_Teams(teamA varchar,
2.                                        teamB varchar,
3.                                        lastNYears number default 5)
4.     return number as
5.     v_final number;
6.     v_count number;
7.
8. begin
9.     select count(*)
10.      into v_count
11.      from teams
12.      where TEAM_ID = teamA
13.         or TEAM_ID = teamB;
14.
15.      if v_count < 2 then
16.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_team);
17.      end if;
18.      if lastNYears < 0 then
19.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.invalid_year);
20.      end if;
21.
22.      select count(*)
23.      into v_final
24.      from (select game_id
25.            from HISTORY_GAMES
26.            where (A_TEAM_ID = teamA and B_TEAM_ID = teamB)
27.               or (A_TEAM_ID = teamB and B_TEAM_ID = teamA)
28.            and MATCH_DATE > add_months(sysdate, -lastNYears * 12));
29.
30.      return v_final;
31.  end;
32.  /
```

c) Create the gameDiffGoals function, which takes the identifier of a game as an argument and returns the difference between the goals scored by the visited team and the visiting team. The function can throw the following exceptions: -20502.

```
d) create function c_game_Diff_Goals(vIdGame Number)
e)     return number as
f)
g)     v_final number;
h)     v_count number;
i)
j) begin
k)     select count(*)
l)     into v_count
m)     from HISTORY_GAMES
n)     where GAME_ID = vIdGame;
o)
p)     if v_count = 0 then
q)         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_game);
r)     end if;
s)
t)     select A_GOALS - B_GOALS
u)     into v_final
v)     from HISTORY_GAMES
w)     where GAME_ID = vIdGame;
x)
y)     return v_final;
z) end;
aa)  /
```

d) Create the function lastday, which takes the name of a competition as an argument and returns the identifier of the last day that took place (held) of that competition, that is, whose date of the day is less than the current one. The function can throw the following exceptions: -20505 and -20511.

```
a) create function d_last_phase(vNameCompetition COMPETITIONS.NAME%type)
b)     return phases.phase_id%type as
c)
d)     v_final phases.phase_id%type;
e)     v_count number;
f)
g) begin
h)     select count(*)
i)     into v_count
j)     from COMPETITIONS c
k)     where c.NAME = vNameCompetition;
l)
m)     if v_count = 0 then
n)         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_competition);
o)     end if;
p)
q)     select *
r)     into v_final
s)     from (select p.PHASE_ID
t)           from PHASES p,
u)                COMPETITIONS c
v)           where c.COMPETITION_ID = p.COMPETITION_ID
w)             and c.NAME = vNameCompetition
x)           order by p.END_DATE desc)
y)     where rownum = 1;
z)
aa)      return v_final;
bb)  end;
cc)  /
```

e) Create the new_game procedure, which receives the name of the two teams (the visited team and the visiting team), the time of the game (date and time) and the competition and records that game. The game must be associated with the respective matchday (determined by the date). The procedure can throw the following exceptions: -20501, -20505, -20512 and -20513

```
1. create procedure e_new_Game(teamA varchar,
2.                             teamB varchar,
3.                             dataHora date,
4.                             vCompeticao varchar)
5. as
6.     v_stadium  games.stadium%type;
7.     v_id       games.game_id%type;
8.     v_phase_id games.phase_id%type;
9.     v_count    number;
10.
11.  begin
12.     select count(*)
13.     into v_count
14.     from TEAMS
15.     where TEAM_ID = teamA
16.        or TEAM_ID = teamB;
17.     if v_count < 2 then
18.         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_team);
19.     end if;
20.
21.     select count(*)
22.     into v_count
23.     from COMPETITIONS
24.     where COMPETITION_ID = vCompeticao;
25.     if v_count = 0 then
26.         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_competition);
27.     end if;
28.
29.     select coalesce(max(game_id), 0)
30.     into v_id
31.     from games;
32.     v_id := v_id + 1;
33.
34.     select *
35.     into v_phase_id
36.     from (select phase_id
37.           from phases
38.           where START_DATE < dataHora
39.             and END_DATE > dataHora
40.             and COMPETITION_ID = vCompeticao
41.           order by START_DATE)
42.     where rownum = 1;
43.
44.     select STADIUM
45.     into v_stadium
46.     from TEAMS
47.     where TEAM_ID = teamA;
48.
49.     insert into games
50.     values (v_id, v_phase_id, teamA, teamB, dataHora, v_stadium);
51.  end;
52.  /
```

f) Create the procedure define_odds_iniadas_21a which, receives the name of the two teams (the visited team and the visiting team), and the time of the game (date and time) and defines the initial odds of that game according to the historical information (defined in section 2.1.a). The procedure can throw the following exceptions: -20501, -20502 and -20514.

```
1. create procedure f_define_odds_iniciais_21a(teamA varchar,
2.                                             teamB varchar,
3.                                             dataHora date) as
4.
5.     v_count        number;
6.     v_id           number := 1;
7. --         prevention flags
8.     v_margin       float  := 0.3;
9.     v_min_chance   float  := 50;
10.      -- sometimes chance from probA is 0, to avoid calculation problems we
   assume
11.     minimal chance to 50%
12.     v_min_odd      float  := 1.01;
13. --             final odds
14.     v_A_win        float  := 0;
15.     v_draw         float  := 0;
16.     v_B_win        float  := 0;
17.     --             chances
18.     v_A_win_chance probability_B.A_WIN_CHANCE%type;
19.     v_draw_chance  probability_B.DRAW_CHANCE%type;
20.     v_B_win_chance probability_B.B_WIN_CHANCE%type;
21.     cursor c_game is select *
22.                      from games
23.                      where (A_TEAM_ID = teamA and B_TEAM_ID = teamB)
24.                         or (A_TEAM_ID = teamB and B_TEAM_ID = teamA);
25. begin
26.     for record in c_game
27.         loop
28.             select count(*)
29.             into v_count
30.             from PROBABILITY_A
31.             where A_TEAM_ID = record.A_TEAM_ID and
32.                   B_TEAM_ID = record.B_TEAM_ID
33.                or A_TEAM_ID = record.B_TEAM_ID and
34.                   B_TEAM_ID = record.A_TEAM_ID;
35.
36. --         if pair of team does not have calculated probability
37.             if v_count = 0 then
38.                 v_A_win_chance := 0;
39.                 v_draw_chance := 0;
40.                 v_B_win_chance := 0;
41.
42.             else
43. --         get probability A record of pair of teams playing in following game
44.                 select A_WIN_CHANCE, DRAW_CHANCE, B_WIN_CHANCE
45.                 into v_A_win_chance, v_draw_chance, v_B_win_chance
46.                 from probability_B
47.                 where A_TEAM_ID = record.A_TEAM_ID and
48.                       B_TEAM_ID = record.B_TEAM_ID
49.                    or A_TEAM_ID = record.B_TEAM_ID and
50.                       B_TEAM_ID = record.A_TEAM_ID;
51.             end if;
52.
53. --         calculate win A odd according to formula
54.             if v_A_win_chance = 0 then
55.                 v_A_win := (1 / (v_min_chance / 100)) * (1 - v_margin);
56.             else
57.                 v_A_win := (1 / (v_A_win_chance / 100)) * (1 - v_margin);
58.             end if;
59.
60.
```

```
61.
62.  --          calculate draw odd according to formula
63.              if v_draw_chance = 0 then
64.                  v_draw := (1 / (v_min_chance / 100)) * (1 - v_margin);
65.              else
66.                  v_draw := (1 / (v_draw_chance / 100)) * (1 - v_margin);
67.              end if;
68.
69.
70.  --          calculate win B odd according to formula
71.              if v_B_win_chance = 0 then
72.                  v_B_win := (1 / (v_min_chance / 100)) * (1 - v_margin);
73.              else
74.                  v_B_win := (1 / (v_B_win_chance / 100)) * (1 - v_margin);
75.              end if;
76.
77.  --          prevent odds not to be below 1
78.              if v_A_win <= 1 then
79.                  v_A_win := v_min_odd;
80.              end if;
81.
82.              if v_draw <= 1 then
83.                  v_draw := v_min_odd;
84.              end if;
85.
86.              if v_B_win <= 1 then
87.                  v_B_win := v_min_odd;
88.              end if;
89.
90.  --          find max id of odds
91.              select coalesce(max(odd_id), 0)
92.              into v_id
93.              from odds;
94.
95.              v_id := v_id + 1;
96.
97.  --          win A odd
98.              insert into odds
99.              values (v_id, record.game_id, 1, v_A_win, dataHora);
100.             v_id := v_id + 1;
101.
102. --          draw odd
103.             insert into odds
104.             values (v_id, record.game_id, 2, v_draw, dataHora);
105.             v_id := v_id + 1;
106.
107. --          win B odd
108.             insert into odds
109.             values (v_id, record.game_id, 3, v_B_win, dataHora);
110.         end loop;
111. end;
112. /
```

g) Create the payBets procedure, which receives the identifier of a game, and records the payment of all winning bets for that game. The procedure can throw the following exceptions: -20502, -205017 and -20518.

```
1. create procedure G_PAY_BETS(vIdGame number) is
2.
3.     v_count            number;
4.     v_id               number := 0;
5.     v_final            float  := 0;
6.     v_game             history_games%rowtype;
7.     v_odd_type_id      odds.odd_type_id%type;
8.     v_goals_difference number := 0;
9.
10.  --         get all bets placed on that game
11.     cursor c_bets is
12.         select b.*
13.         from bets b,
14.              history_odds o
15.         where b.odd_id = o.odd_id
16.           and o.game_id = vIdGame;
17.  begin
18.
19.     --     count whether there is any game of that game_id
20.     select count(GAME_ID)
21.     into v_count
22.     from (select GAME_ID
23.           from GAMES
24.           union
25.           select GAME_ID
26.           from history_games)
27.     where GAME_ID = vIdGame;
28.
29.     if v_count = 0 then
30.         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_game);
31.     end if;
32.
33.     --         count whether there are any prized paid out for that game
34.     select count(*)
35.     into v_count
36.     from PAYOUTS p
37.              join BETS b
38.                   on b.BET_ID = p.BET_ID
39.              join HISTORY_ODDS HO on b.ODD_ID = HO.ODD_ID
40.     where HO.GAME_ID = vIdGame;
41.
42.     if v_count > 0 then
43.         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.prizes_paid);
44.     end if;
45.
46.     --     check if game has finished - it is moved to history_games table
47.     select count(*)
48.     into v_count
49.     from games
50.     where GAME_ID = vIdGame;
51.
52.     if v_count > 0 then
53.         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.game_not_over);
54.     else
55.         select * into v_game from HISTORY_GAMES where GAME_ID = vIdGame;
56.         v_goals_difference := v_game.A_goals - v_game.B_goals;
57.     end if;
58.
59.
60.
61.
62.
63.
```

```
64.
65.  --      pay for winning bets
66.      for record in c_bets
67.          loop
68.              select o.ODD_TYPE_ID
69.              into v_odd_type_id
70.              from history_odds o
71.              where o.ODD_ID = record.ODD_ID;
72.
73.              if
74.                      (v_goals_difference > 0 and v_odd_type_id = 1)
75.                      or
76.                      (v_goals_difference = 0 and v_odd_type_id = 2)
77.                      or
78.                      (v_goals_difference < 0 and v_odd_type_id = 3)
79.              then
80.                  select coalesce(max(payout_id), 0)
81.                  into v_id
82.                  from payouts;
83.                  v_id := v_id + 1;
84.
85.                  select o.value
86.                  into v_final
87.                  from history_odds o
88.                  where o.odd_id = record.odd_id;
89.
90.                  v_final := v_final * record.money_placed;
91.                  insert into payouts
92.                  values (v_id, v_final, sysdate, record.client_id,
93.                          record.bet_id);
94.              end if;
95.          end loop;
96.  end;
97.  /
```

h) Create the nGameEvents function, which receives the code for a game, the code for a team and the type of event that can happen in a game, and returns the number of events of that type that happened in that game by that team. Eg the number of goals the team scored in that game. The function can throw the following exceptions: -20501, -20502 and -20515.

```
1. create function H_nGAME_EVENTS(vIDgame NUMBER,
2.                                vIDTeam teams.team_id%type,
3.                                vIdEventType NUMBER) return number as
4.     v_count number;
5.     v_total number := 0;
6. begin
7.     -- check if game exists in history games
8.     select count(*)
9.     into v_count
10.      from HISTORY_GAMES
11.      where GAME_ID = vIDgame;
12.
13.      if v_count = 0 then
14.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_game);
15.          return 0;
16.      end if;
17.
18.      -- check if team exists in that game
19.      select count(*)
20.      into v_count
21.      from HISTORY_GAMES
22.      where GAME_ID = vIDgame
23.        and (A_TEAM_ID = vIDTeam or B_TEAM_ID = vIDTeam);
24.
25.      if v_count = 0 then
26.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_team);
27.          return 0;
28.      end if;
29.
30.      -- check if team exists in that game
31.      select count(*)
32.      into v_count
33.      from EVENT_TYPE
34.      where EVENT_TYPE_ID = vIdEventType;
35.
36.      if v_count = 0 then
37.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.invalid_event_type);
38.          return 0;
39.      end if;
40.
41.      select count(*)
42.      into v_total
43.      from EVENTS
44.      where GAME_ID = vIDgame
45.        and TEAM_ID = vIDTeam
46.        and EVENT_TYPE_ID = vIdEventType;
47.
48.      return v_total;
49.  end;
50.  /
```

i) Create the placeBet procedure, which receives the code of a user / bettor, the identifier of a game, the type of bet and the value of the bet and records that bet. The minimum stake is 1. The procedure may raise the following exceptions: -20503, -20502, -20506, -20507, -20508, -20509 and -20516.

```
1. create procedure I_PLACE_BET(vIdUser CLIENTS.CLIENT_ID%type,
2.                              vIdGame GAMES.GAME_ID%type,
3.                              vIdOddType ODDS.ODD_ID%type,
4.                              value BETS.MONEY_PLACED%TYPE) as
5.
6.     v_match_date games.match_date%type;
7.     v_count      number;
8.     v_interval   number;
9.     v_max_bet_id bets.bet_id%type;
10.      v_odd_id     bets.odd_id%type;
11.  begin
12.
13.      --     check if user exists in database
14.      select count(*) into v_count from CLIENTS where CLIENT_ID = vIdUser;
15.      if v_count = 0 then
16.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_bettor);
17.      end if;
18.
19.  -- check if game has not finished, is in games table
20.      select count(*) into v_count from GAMES where GAME_ID = vIdGame;
21.      if v_count = 0 then
22.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_game);
23.      end if;
24.
25.  --     check type of odd
26.      select count(*) into v_count from ODD_TYPE where ODD_TYPE_ID = vIdOddType;
27.      if v_count = 0 then
28.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.invalid_bet_type);
29.      end if;
30.
31.  --     check minimal bet (1 euro)
32.      if value < 1 and value >= 0 then
33.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_money);
34.      end if;
35.
36.      --     check negative bet
37.      if value < 0 then
38.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.negative_bet);
39.      end if;
40.
41.  --     check date whether bet time fits 15 min before start of the game
42.      select MATCH_DATE
43.      into v_match_date
44.      from GAMES
45.      where GAME_ID = vIdGame;
46.
47.      select extract(minute from diff) minutes
48.      into v_interval
49.      from (select systimestamp - v_match_date diff
50.            from dual);
51.
52.  --     shift match date -15 min to check if bet date does not exceed it
53.      if v_interval >= -15 then
54.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.game_closed);
55.      end if;
56.
57.
58.
59.
60.
61.
62.
```

```
63.      --     INSERTING BET
64.  --     get odd id by game_id and odd_type_id
65.      select ODD_ID
66.      into v_odd_id
67.      from ODDS
68.      where GAME_ID = vIdGame
69.        and ODD_TYPE_ID = vIdOddType;
70.
71.  --     get max bet id
72.      select coalesce(max(BET_ID), 0)
73.      into v_max_bet_id
74.      from bets;
75.      v_max_bet_id := v_max_bet_id + 1;
76.
77.      --     insert into BETS(BET_ID, CLIENT_ID, ODD_ID, MONEY_PLACED, BET_DATE)
78.  --     values (v_max_bet_id, vIdUser, v_odd_id, value, sysdate);
79.
80.  --     lack of data according to L_FILL_BET trigger requirements
81.      insert into BETS(BET_ID, CLIENT_ID, ODD_ID, MONEY_PLACED, BET_DATE)
82.      values (v_max_bet_id, vIdUser, v_odd_id, value, null);
83.  end;
84.  /
```

j) Create a trigger actualiza_saldo_premios that when the payment of the prize of a winning bet is registered, updates (increases) that player's account balance with the prize amount.

```
1. create trigger J_UPDATE_PAYMENTS_BALANCE
2.     after insert
3.     on PAYOUTS
4.     for each row
5. declare
6. begin
7.     update CLIENTS set BALANCE = BALANCE + :new.money
8.     where CLIENT_ID = :new.client_id;
9. end;
10.  /
```

k) Create a update_saldo_bet trigger that when a bet is registered, updates (decreases) the player account balance with the bet amount.

```
1. create trigger K_UPDATE_BET_BALANCE
2.     after insert
3.     on BETS
4.     for each row
5. declare
6. begin
7.     update CLIENTS set BALANCE = BALANCE - :new.money_placed
8.     where CLIENT_ID = :new.client_id;
9. end;
10.  /
```

I) Create a fillBet trigger that when a bettor registers a bet, indicating the game, the type of bet and the value of that bet, it fills in the missing information, namely the date of registration of that bet, the current value of the odd and the potential win of the bet.

```
1. create trigger L_FILL_BET
2.     before insert
3.     on BETS
4.     for each row
5. declare
6.     v_bet_date    timestamp;
7.     v_odd_value   odds.value%type;
8.     v_odd_type    odds.odd_type_id% type;
9.     v_probability number;
10.      v_A_team_id   teams.team_id%type;
11.      v_B_team_id   teams.team_id%type;
12.  begin
13.      --     assign sysdate to new inserted row
14.      v_bet_date := sysdate;
15.      :new.BET_DATE := v_bet_date;
16.
17.  -- get value and type of odd by odd_id
18.      select VALUE, ODD_TYPE_ID
19.      into v_odd_value, v_odd_type
20.      from ODDS
21.      where ODD_ID = :new.ODD_ID;
22.
23.  --     get teams ids of the game with particular odd
24.      select A_TEAM_ID, B_TEAM_ID
25.      into v_A_team_id, v_B_team_id
26.      from games g
27.               join ODDS O on O.GAME_ID = g.GAME_ID
28.      where O.ODD_ID = :new.ODD_ID;
29.
30.  --     select probability to win depending on odd type
31.      select case v_odd_type
32.                 when 1 then A_WIN_CHANCE
33.                 when 2 then DRAW_CHANCE
34.                 when 3 then B_WIN_CHANCE end
35.      into v_probability
36.      from PROBABILITY_B
37.      where A_TEAM_ID = v_A_team_id and B_TEAM_ID = v_B_team_id
38.         or B_TEAM_ID = v_A_team_id and A_TEAM_ID = v_B_team_id;
39.  end;
40.  /
```

m) Create the update_odds trigger that after registering a bet, checks and, if necessary, recalculates the odds for that game (see section 2.2).

```
1. create trigger M_UPDATE_ODDS
2.     after insert
3.     on BETS
4.     for each row
5. declare
6.     v_game_id           games.game_id%type;
7.     v_sum_on_one_game   CALC_TOTAL.PLACED_TOTAL%type;
8.     v_odd_type_id       odd_type.odd_type_id%type;
9.     v_total_prize       calc_type_game.result_prize%type;
10.      v_total_match_prize calc_type_game.placed%type;
11.      v_max_prize         CALC_TOTAL.MAX_PRIZE%type;
12.  begin
13.
14.      --  get game_id and odd_type_id of bet odd
15.      select game_id, odd_type_id
16.      into v_game_id, v_odd_type_id
17.      from odds
18.      where odd_id = :new.odd_id;
19.      --
20.  --  get total prize for the game_id nad odd_type
21.      select PLACED_TOTAL
22.      into v_sum_on_one_game
23.      from CALC_TOTAL
24.      where GAME_ID = v_game_id;
25.      --
26.  --  get total sum placed on the game on the same result
27.      select result_prize
28.      into v_total_prize
29.      from CALC_TYPE_GAME
30.      where game_id = v_game_id
31.        and odd_type_id = v_odd_type_id;
32.      --
33.  --      get max prize on the game
34.      select MAX_PRIZE
35.      into v_max_prize
36.      from CALC_TOTAL
37.      where GAME_ID = v_game_id;
38.      --
39.  --  the amount of a bet on a game result exceeds € 100
40.      if :new.money_placed > 100 or
41.
42.          --the amount of a bet on a result of the game is
43.          --greater than 2% of the total amount bet on that result.
44.         :new.money_placed > v_sum_on_one_game * 0.02 or
45.
46.          --total Prize on match result > Total Max Prize Match
47.        v_total_prize >= v_total_match_prize then
48.
49.          ODD_CTRL.RECALCULATE_ODD(v_game_id);
50.
51.      end if;
52.  end;
53.  /
```

n) Each member of the group must create a function, with the format func_n_aluno, which they consider relevant, justifying its relevance. Relevance and level of complexity will strongly influence your assessment.

N1_funct_a2019156557
Function which gets in parameters client_id and amount of money to charge. Function updates balance of given value and returns new amount of money in client's account. Possible exceptions: -20503 and -20519 (own - invalid charge value).

```
1. create function N1_FUNCT_A2019156557(p_client_id clients.client_id%type,
2.                                      p_charge clients.balance%type)
3.     return number as
4.     v_count   number;
5.     v_balance clients.balance%type;
6. begin
7.
8.     --     check if client exists in database
9.     select count(*) into v_count from clients where client_id = p_client_id;
10.      if v_count = 0 then
11.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_bettor);
12.      end if;
13.
14.  --     check if charge is correct
15.      if p_charge <= 0 then
16.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.invalid_charge);
17.      end if;
18.
19.      update CLIENTS
20.      set balance = balance + p_charge
21.      where CLIENT_ID = p_client_id;
22.
23.      select BALANCE
24.      into v_balance
25.      from CLIENTS
26.      where client_id = p_client_id;
27.
28.      return v_balance;
29.  end;
30.  /
```

N1_funct_a2019156734

Returns the total amount of bets placed by a bettor for all games. Possible exceptions: -20503

```
1. create function n2_funct_a2019156734(p_client_id clients.client_id%type)
2.     return number as
3.     v_final1 number;
4.     v_final2 number;
5.     v_count number;
6. begin
7.     select count(*) into v_count from CLIENTS where CLIENT_ID = p_client_id;
8.     if v_count = 0 then
9.         EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_bettor);
10.      end if;
11.
12.      select count(CLIENT_ID)
13.      into v_final1
14.      from BETS
15.      where CLIENT_ID = p_client_id;
16.
17.      select count(CLIENT_ID)
18.      into v_final2
19.      from HISTORY_BETS
20.      where CLIENT_ID = p_client_id;
21.
22.      return v_final1 + v_final2;
23.  end;
24.  /
25.
```

o) Each member of the group must create a trigger, with the format trig_ n_student, which they consider relevant, justifying its relevance. Relevance and level of complexity will strongly influence your evaluation.

O1_proc_a2019156557
Procedure which updates PROBABILITY_A table with given team A id and team B id. It calculates new chances according to provided formulas. It can be used to calculate odds in further database operations. Possible exceptions: -20501.

```
1. create procedure O1_PROC_A2019156557(p_A_team_id
2.                                      history_comparison.A_team_id%type,
3.                                      p_B_team_id
4.                                      history_comparison.B_team_id%type) as
5.
6.     v_history_comparison history_comparison%rowtype;
7.     v_id                 number := 0;
8.     v_count              number := 0;
9.     v_A_win_prob         float  := 0;
10.    v_draw_prob          float  := 0;
11.    v_B_win_prob         float  := 0;
12.
13.  begin
14.
15.      --     check if those teams exists in database
16.      select count(*)
17.      into v_count
18.      from TEAMS
19.      where TEAM_ID = p_A_team_id
20.         or TEAM_ID = p_B_team_id;
21.
22.      if v_count != 2 then
23.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_team);
24.      end if;
25.
26.      --         get history comparison record concerning pair on team ids
27.      select *
28.      into v_history_comparison
29.      from HISTORY_COMPARISON
30.      where A_TEAM_ID = p_A_team_id
31.        and B_TEAM_ID = p_B_team_id;
32.
33.  --         calculate probabilities
34.      v_A_win_prob := (v_history_comparison.A_won /
35.                       (v_history_comparison.matches_amount)) * 100;
36.
37.      v_draw_prob := (v_history_comparison.draw /
38.                      (v_history_comparison.matches_amount)) * 100;
39.
40.      v_B_win_prob := (v_history_comparison.B_won /
41.                       (v_history_comparison.matches_amount)) * 100;
42.
43.  --         check if record for probabilities exists
44.      select count(*)
45.      into v_count
46.      from probability_A
47.      where A_team_id = v_history_comparison.A_team_id
48.        and B_team_id = v_history_comparison.B_team_id;
49.
50.      if v_count = 0 then
51.
52.          --         get max id in table
53.          select max(PROB_A_ID)
54.          into v_id
55.          from PROBABILITY_A;
56.
57.
```

```
58.
59.
60.
61.
62.          if v_id is null then
63.              v_id := 0;
64.          else
65.              v_id := v_id + 1;
66.          end if;
67.
68.          insert into probability_A
69.          values (v_id, v_history_comparison.A_team_id,
70.                  v_history_comparison.B_team_id,
71.                  v_A_win_prob, v_draw_prob, v_B_win_prob);
72.
73.      else
74.          update probability_A
75.          set A_win_chance = v_A_win_prob,
76.              draw_chance  = v_draw_prob,
77.              B_win_chance = v_B_win_prob
78.          where A_team_id = v_history_comparison.A_team_id
79.            and B_team_id = v_history_comparison.B_team_id;
80.      end if;
81.  end;
82.  /
```

O2_proc_a2019156734

The procedure calculates the values in table PROBABILITY_B concerning particular game, using values from PROBABILITY_A and TEAM_STATISTICS tables. The values can be later used to calculate the initial odds.

```
1. create procedure o2_proc_a2019156734(p_A_team_id
2.                                      history_comparison.A_team_id%type,
3.                                      p_B_team_id
4.                                      history_comparison.B_team_id%type) is
5.
6.     v_probability_A probability_A%rowtype;
7.     v_id            number := 0;
8.     v_count         number := 0;
9.     v_A_win         float  := 0;
10.      v_A_draw        float  := 0;
11.      v_A_lose        float  := 0;
12.      v_B_win         float  := 0;
13.      v_B_draw        float  := 0;
14.      v_B_lose        float  := 0;
15.      v_final_A_win   float  := 0;
16.      v_final_draw    float  := 0;
17.      v_final_B_win   float  := 0;
18.  --
19.      v_A_won_num     number;
20.      v_A_draw_num    number;
21.      v_A_lost_num    number;
22.      v_A_played      number;
23.  --
24.      v_B_won_num     number;
25.      v_B_draw_num    number;
26.      v_B_lost_num    number;
27.      v_B_played      number;
28.
29.  begin
30.
31.      select count(*)
32.      into v_count
33.      from teams
34.      where TEAM_ID = p_A_team_id
35.         or TEAM_ID = p_B_team_id;
36.      if v_count < 2 then
37.          EXCEPTIONS.RAISE_EXCEPTION(EXCEPTIONS.no_team);
38.      end if;
39.
40.      --          get team statistics record of A team
41.      select sum(WON), sum(DRAW), sum(LOST), sum(PLAYED)
42.      into v_A_won_num, v_A_draw_num, v_A_lost_num, v_A_played
43.      from TEAM_STATISTICS
44.      where team_id = p_A_team_id;
45.
46.      --          get team statistics record of B team
47.      select sum(WON), sum(DRAW), sum(LOST), sum(PLAYED)
48.      into v_B_won_num, v_B_draw_num, v_B_lost_num, v_B_played
49.      from TEAM_STATISTICS
50.      where team_id = p_B_team_id;
51.
52.      --          get probability A record concerning pair on team ids
53.      select *
54.      into v_probability_A
55.      from PROBABILITY_A
56.      where A_TEAM_ID = p_A_team_id
57.        and B_TEAM_ID = p_B_team_id;
58.
59.  --          calculate team A ratios
60.      v_A_win := (v_A_won_num / v_A_played) * 100;
61.      v_A_draw := (v_A_draw_num / v_A_played) * 100;
62.      v_A_lose := (v_A_lost_num / v_A_played) * 100;
```

```
63.
64.  --          calculate team B ratios
65.      v_B_win := (v_B_won_num / v_B_played) * 100;
66.      v_B_draw := (v_B_draw_num / v_B_played) * 100;
67.      v_B_lose := (v_B_lost_num / v_B_played) * 100;
68.
69.
70.  --          calculate average of ratio A with probabilityA of A team
71.      v_A_win := (v_A_win + v_probability_A.A_win_chance) / 2;
72.      v_A_draw := (v_A_draw + v_probability_A.draw_chance) / 2;
73.      v_A_lose := (v_A_lose + v_probability_A.B_win_chance) / 2;
74.
75.  --          calculate average of ratio B with probabilityA of B team
76.      v_B_win := (v_B_win + v_probability_A.B_win_chance) / 2;
77.      v_B_draw := (v_B_draw + v_probability_A.draw_chance) / 2;
78.      v_B_lose := (v_B_lose + v_probability_A.A_win_chance) / 2;
79.
80.  --          calculate final average
81.      v_final_A_win := (v_A_win + v_B_lose) / 2;
82.      v_final_draw := (v_A_draw + v_B_draw) / 2;
83.      v_final_B_win := (v_B_win + v_A_lose) / 2;
84.
85.  --          check if record for probabilities exists
86.      select count(*)
87.      into v_count
88.      from probability_B
89.      where A_team_id = p_A_team_id
90.        and B_team_id = p_B_team_id;
91.
92.      if v_count = 0 then
93.
94.          --          get max id in table
95.          select max(PROB_A_ID)
96.          into v_id
97.          from PROBABILITY_A;
98.
99.          if v_id is null then
100.             v_id := 0;
101.         else
102.             v_id := v_id + 1;
103.         end if;
104.
105.         insert into probability_B
106.         values (p_A_team_id, p_B_team_id,
107.                 v_final_A_win, v_final_draw, v_final_B_win,
108.                 v_probability_A.PROB_A_ID);
109.     else
110.         update probability_B
111.         set A_win_chance = v_final_A_win,
112.             draw_chance  = v_final_draw,
113.             B_win_chance = v_final_B_win
114.         where A_team_id = p_A_team_id
115.           and B_team_id = p_B_team_id;
116.     end if;
117.
118. end;
119. /
```

p) Each member of the group must create a trigger, with the format trig_ n_student, which they consider relevant, justifying its relevance. Relevance and level of complexity will strongly influence your evaluation.

P1_trig_a2019156557
Trigger which before insert on table BETS updates total bets placed on particular odd type for a betted game.

```
1. create trigger P1_TRIG_A2019156557
2.     before insert
3.     on BETS
4.     for each row
5. declare
6.     v_game_id     games.GAME_ID%type;
7.     v_odd_type_id odds.odd_type_id%type;
8.     v_calc_id     number := 0;
9.     v_count       number := 0;
10.    v_odd_value   number;
11.
12.  begin
13.      --         get game_id of bet and odd value
14.      select GAMES.GAME_ID, O2.VALUE, O2.ODD_TYPE_ID
15.      into v_game_id, v_odd_value, v_odd_type_id
16.      from GAMES
17.               join ODDS O2 on GAMES.GAME_ID = O2.GAME_ID
18.      where O2.ODD_ID = :new.odd_id;
19.
20.      --         check if record of current bet game exists
21.      select count(*)
22.      into v_count
23.      from CALC_TYPE_GAME
24.               join GAMES G on CALC_TYPE_GAME.GAME_ID = G.GAME_ID
25.               join ODDS O on G.GAME_ID = O.GAME_ID
26.      where O.ODD_ID = :new.odd_id
27.        and CALC_TYPE_GAME.ODD_TYPE_ID = v_odd_type_id;
28.      --
29.  --          if not, create new record
30.      if v_count = 0 then
31.
32.          select coalesce(max(calc_id), 0)
33.          into v_calc_id
34.          from calc_type_game;
35.          v_calc_id := v_calc_id + 1;
36.
37.          insert into CALC_TYPE_GAME
38.          values (v_calc_id, v_game_id, v_odd_type_id, 0, 0);
39.      end if;
40.
41.      update CALC_TYPE_GAME
42.      set PLACED       = PLACED + :new.money_placed,
43.          RESULT_PRIZE = RESULT_PRIZE + (:new.money_placed * v_odd_value),
44.
45.          --         max prize is total money placed on odd * this odd_value
46.          ODD_TYPE_ID  = v_odd_type_id
47.      where GAME_ID = v_game_id
48.        and ODD_TYPE_ID = v_odd_type_id;
49.
50.  end;
51.  /
```

P2_trig_a2019156734

The trigger uses the value of the inserted bet to recalculate values in table CALC_TOTAL concerning particular game, which is later used in recalculation of odd

```
1. create trigger P2_TRIG_A2019156734
2.     after insert
3.     on BETS
4.     for each row
5. declare
6.
7.     v_game_id   games.GAME_ID%type ;
8.     v_count     number := 0;
9.     v_odd_value number;
10.  begin
11.      --          get game_id of bet and odd value
12.      select GAMES.GAME_ID, O2.VALUE
13.      into v_game_id, v_odd_value
14.      from GAMES
15.               join ODDS O2 on GAMES.GAME_ID = O2.GAME_ID
16.      where O2.ODD_ID = :new.odd_id;
17.
18.  --          check if record of current bet game exists
19.      select count(*)
20.      into v_count
21.      from CALC_TOTAL
22.               join GAMES G on CALC_TOTAL.GAME_ID = G.GAME_ID
23.               join ODDS O on G.GAME_ID = O.GAME_ID
24.      where O.ODD_ID = :new.odd_id;
25.
26.  --          if not, create new record
27.      if v_count = 0 then
28.          insert into CALC_TOTAL values (v_game_id, 0, 0);
29.      end if;
30.
31.      update CALC_TOTAL
32.      set PLACED_TOTAL = PLACED_TOTAL + :new.money_placed,
33.          MAX_PRIZE    = MAX_PRIZE + (:new.money_placed * 0.7);
34.      --          max prize is 70% of total money placed on odd
35.
36.  end;
37.  /
```

Data integrity

Here we present the security measures preserving the data integrity. For each table there are shown both security measures we have implemented and the ones that could be implemented but we have not yet done so.

BETS table

Type	Name	Purpose
Foreign key	FK_BETS_CLIENT_ID	To ensure that bet has its client in database
Unique constraint	PK_BETS	To avoid repetition of PK bet_id
Trigger	K_UPDATE_BET_BALANCE	Reduces the amount of money on client's account after placing a bet.
Trigger	L_FILL_BET	Updates missing data such as date before inserting new bet
Trigger	M_UPDATE_ODDS	Check whether bet satisfies any condition to invoke recalculation of the bet by provided formula in task description.
Trigger	P1_TRIG_A2019156557	Updates table CALC_TOTAL which stores total money placed on particular game with recalculated other values like maximum prize able to be paid out.
Trigger	P2_TRIG_A2019156734	Updates table CALC_TYPE_GAME that consists of money staked on the game yet on particular result of the game (1,x,2)

Propositions

Type	Purpose
Trigger	Check whether odd exists of particular odd type connected to betted game

CALC_TOTAL table

Type	Name	Purpose
Unique constraint	PK_CALC_TOTAL	To avoid repetition of PK game_id

CALC_TYPE_GAME table

Type	Name	Purpose
Foreign key	FK_CALC_TYPE_GAME_ODD_TYPE_ID	To ensure that calculation for specific betted result has it's representative in ODDS table by certain odd_id
Unique constraint	PK_CALC_TYPE_GAME	To avoid repetition of calc_id

CLIENTS table

Type	Name	Purpose
Unique constraint	PK_CLIENTS	To avoid repetition of PK client_id

Propositions

Type	Purpose
Trigger	Check if identification number from document fits proper regex
Trigger	Check if phone number fits proper format

COMPETITIONS table

Type	Name	Purpose
Unique constraint	PK_COMPETITIONS	To avoid repetition of PK competition_id

Propositions

Type	Purpose
Trigger	Check if end_date is lower than start_date

EVENT_TYPE table

Type	Name	Purpose
Unique constraint	PK_EVENT_TYPE	To avoid repetition of PK event_type_id

EVENTS table

Type	Name	Purpose
Foreign key	FK_EVENTS_EVENT_TYPE_ID	To ensure that there exists such event_type
Foreign key	FK_EVENTS_GAME_ID	To ensure that there exists such team
Foreign key	FK_EVENTS_TEAM_ID	To ensure that there exists such team
Unique constraint	PK_EVENTS	To avoid repetition of PK event_id

Propositions

Type	Purpose
Trigger	Check if minute of the event does not exceeds the length of the match
Trigger	Check if inserted team_id of the team really played in that game

GAMES table

Type	Name	Purpose
Foreign key	FK_GAMES_A_TEAM_ID	To ensure that there exists such team
Foreign key	FK_GAMES_B_TEAM_ID	To ensure that there exists such team
Foreign key	FK_GAMES_PHASE_ID	To ensure that there exists such phase
Unique constraint	PK_GAMES	To avoid repetition of PK game_id
Trigger	T_GAME_DELETE	Deletes all odds concerning this game from Odds and inserts them into HISTORY_ODDS before deleting the game
Trigger	T_GAME_INSERT_NEW_ODD	Inserts initial odds on that game into Odds after inserting a new game

Propositions

Type	Purpose
Trigger	Check if match_date fits in phase time period

HISTORY_BETS table

Type	Name	Purpose
Unique constraint	HISTORY_BETS_PK	To avoid repetition of PK bet_id

HISTORY_COMPARISON table

Type	Name	Purpose
Foreign key	FK_HISTORY_COMP_A_TEAM_ID	To ensure that there exists such team
Foreign key	FK_HISTORY_COMP_B_TEAM_ID	To ensure that there exists such team
Unique constraint	PK_HISTORY_COMPARISON	To avoid repetition of PK (A_team_id, B_team_id)

HISTORY_GAMES table

Type	Name	Purpose
Unique constraint	PK_HISTORY_GAMES	To avoid repetition of PK game_id

Propositions

Type	Purpose
Trigger	Check if score is not negative

HISTORY_ODDS table

Type	Name	Purpose
Unique constraint	PK_ODD_TYPE	To avoid repetition of PK odd_id

ODD_TYPE table

Type	Name	Purpose
Foreign key	FK_ODDS_GAME_ID	To avoid repetition of PK odd_type_id

ODDS table

Type	Name	Purpose
Foreign key	FK_ODDS_GAME_ID	To ensure that there exists such game
Foreign key	FK_ODDS_ODD_TYPE_ID	To ensure that there exists such odd_type
Unique constraint	PK_ODDS	To avoid repetition of PK odd_id

Propositions

Type	Purpose
Trigger	Check if odd_date is not greater than match_date and block inserting new odds 15 min before beginning of the match

PAYOUTS table

Type	Name	Purpose
Foreign key	FK_PAYOUTS_BET_ID	To ensure that there exists such bet
Foreign key	FK_PAYOUTS_CLIENT_ID	To ensure that there exists such client
Unique constraint	PK_PAYOUTS	To avoid repetition of PK payout_id
Trigger	J_UPDATE_PAYMENTS_BALANCE	Increases the amount of money on client's account after inserting a payout

Propositions

Type	Purpose
Trigger	Check if money is not negative

PHASES table

Type	Name	Purpose
Foreign key	FK_PHASES_COMPETITION_ID	To ensure that there exists such competition
Unique constraint	PK_PHASES	To avoid repetition of PK phase_id

Propositions

Type	Purpose
Trigger	Check if start_date does not exceeds end_date
Trigger	Check if start_date and end_date consists of whole competition duration

PROBABILITY_A table

Type	Name	Purpose
Foreign key	FK_PROB_A_A_TEAM_ID	To ensure that there exists such HISTORY_COMPARISON
Unique constraint	PK_PROB_A_ID	To avoid repetition of PK prob_A_id

PROBABILITY_B table

Type	Name	Purpose
Foreign key	FK_PROB_B_PROB_A_ID	To ensure that there exists such PROBABILITY_A
Unique constraint	PK_PROB_B_ID	To avoid repetition of PK (A_team_id, B_team_id)

TEAM_STATISTICS table

Type	Name	Purpose
Foreign key	FK_TEAM_STATS_COMPETITION_ID	To ensure that there exists such competition
Foreign key	FK_TEAM_STATS_TEAM_ID	To ensure that there exists such team
Unique constraint	PK_TEAM_STATISTICS	To avoid repetition of PK (team_id, competition_id)

Propositions

Type	Purpose
Trigger	Check if sum of won, lost and draw matches is equal to played games

TEAMS table

Type	Name	Purpose
Foreign key	PK_TEAMS_TEAM_ID	To avoid repetition of PK team_id

Propositions

Type	Purpose
Trigger	Check if team_id fits proper format (uppercase without spaces, maximum 5 chars)

Physical parameters of tables

According to information provided in lectures and Oracle documentation we performed calculations to obtain particular values of physical parameters of five selected tables from our database.

Block parameters

Name	Description
INITIAL	Initial size of block (in bytes)
NEXT	Size to be assigned when initial value is exceeded (in bytes)
MAXEXTEND	Max number of extensions
MINEXTEND	Number of extents assigned to the segment when it is created
PCTINCREASE	% increase in the size of extents assigned after Next
PCTFREE	What percentage of the block should be free for updates
PCTUSED	Minimum occupancy percentage of the block
FIXED HEADER (HF)	57 bytes (default according to Oracle documentation)
VARIABLE HEADER (HV)	5 bytes / each record • 2 bytes – header of records • 1 byte of number of columns • 2 bytes in the Row Directory

Calculations

Name	Description
Average size of registration (T.M.R)	Sum(Average size of fields) + 5 byte/record + 1 byte for each column of record
Free space in the block (E.L.B)	Block size * (100 – PTCFREE) / (100 – Fixed header)
Number of records per block (N.R.B)	Block free space / Average Record size (Rounded down)
Number of blocks (N.B.)	Number of existing records / number of records per block (Rounded up)
Initial table space (E.I.T)	Number of blocks * block size
Table next space (E.N.T)	(Number of records / Number of records per block) * block size

Default values

Our calculations are based on by default assigned values which we obtained by querying USER_TABLES table

```
1.      SELECT INITIAL_EXTENT, NEXT_EXTENT, MAX_EXTENTS, MIN_EXTENTS,
2.                  PCT_INCREASE, PCT_FREE, PCT_USED
3.          FROM USER_TABLES
4.          WHERE TABLE_NAME = <<table_name>>;
```

Name	Output
INITIAL	65536
NEXT	1048576
MAXEXTEND	2147483645
MINEXTEND	1
PCTINCREASE	NULL
PCTFREE	10
PCTUSED	NULL
FIXED HEADER (HF)	57 bytes
VARIABLE HEADER (HV)	5 bytes

CALCULATIONS

To automatize process of calculations and have brief access to results we created new table in our database called PHYSICAL_PARAMETERS which stores all values calculated by formulas given on lectures. To perform calculations we developed procedure Z_CALC_PHYSICAL_PARAMETERS as follows:

```
1. create procedure Z_CALC_PHYSICAL_PARAMETERS(p_table_name varchar2,
2.                                             p_new_estimated_records number) is
3.     v_count        number := 0;
4.     v_sum_size     number := 0;
5.     v_column_count number := 0;
6.     v_block_size   number := 0;
7.     v_PCTFree      number := 0;
8.     v_records      number := 0;
9.     v_sql          varchar2(100);
10.  --
11.      cursor c_columns is
12.          select *
13.          from USER_TAB_COLUMNS
14.          where TABLE_NAME = p_table_name;
15.  --
16.      cursor c_table_info is
17.          SELECT INITIAL_EXTENT,
18.                 NEXT_EXTENT,
19.                 MAX_EXTENTS,
20.                 MIN_EXTENTS,
21.                 PCT_INCREASE,
22.                 PCT_FREE,
23.                 PCT_USED
24.          FROM USER_TABLES
25.          WHERE TABLE_NAME = p_table_name;
26.  --
27.      v_TMR          number;
28.      v_ELB          number;
29.      v_NRB          number;
30.      v_NB           number;
31.      v_EIT          number;
32.      v_ENT          number;
33.
34.  begin
35.
36.      DBMS_OUTPUT.PUT_LINE('--------------Physical parameters-----------------');
37.      for info in c_table_info
38.          loop
39.              DBMS_OUTPUT.PUT_LINE('INITIAL_EXTENT: ' || info.INITIAL_EXTENT);
40.              DBMS_OUTPUT.PUT_LINE('NEXT_EXTENT: ' || info.NEXT_EXTENT);
41.              DBMS_OUTPUT.PUT_LINE('MAX_EXTENTS: ' || info.MAX_EXTENTS);
42.              DBMS_OUTPUT.PUT_LINE('MIN_EXTENTS: ' || info.MIN_EXTENTS);
43.              DBMS_OUTPUT.PUT_LINE('PCT_INCREASE: ' || info.PCT_INCREASE);
44.              DBMS_OUTPUT.PUT_LINE('PCT_FREE: ' || info.PCT_FREE);
45.              DBMS_OUTPUT.PUT_LINE('PCT_USED: ' || info.PCT_USED);
46.
47.              v_PCTFree := info.PCT_FREE;
48.              exit;
49.          end loop;
50.
51.      DBMS_OUTPUT.PUT_LINE('----------------Column names----------------------');
52.      for col in c_columns
53.          loop
54.              DBMS_OUTPUT.PUT_LINE(col.COLUMN_NAME || ' ' || col.DATA_TYPE || ' '
   || col.DATA_LENGTH || ' bytes');
55.              v_sum_size := v_sum_size + col.DATA_LENGTH;
56.              v_column_count := v_column_count + 1;
57.          end loop;
58.
59.
```

```
60.
61.
62.     DBMS_OUTPUT.PUT_LINE('total size: ' || v_sum_size || ' bytes');
63.
64.     DBMS_OUTPUT.PUT_LINE('----------------Calculations----------------------');
65.
66.     v_TMR := v_sum_size + 5 + v_column_count;
67.     DBMS_OUTPUT.PUT_LINE('TMR: ' || v_TMR);
68.
69.     select distinct bytes / blocks
70.     into v_block_size
71.     from user_segments;
72.
73.     --     -----------------ELB------------------
74.
75.     v_ELB := round((v_block_size * (100 - v_PCTFree)) / 100 - 57);
76.     DBMS_OUTPUT.PUT_LINE('ELB: ' || v_ELB);
77.
78.     --     -----------------NRB------------------
79.
80.     v_NRB := FLOOR(v_ELB / v_TMR);
81.     DBMS_OUTPUT.PUT_LINE('NRB: ' || v_NRB);
82.
83.     --     -----------------NB-------------------
84.
85.     v_sql := 'select  count(*)  from ' || p_table_name;
86.     execute immediate v_sql into v_records;
87.
88.     v_NB := CEIL(v_records / v_NRB);
89.     DBMS_OUTPUT.PUT_LINE('NB : ' || v_NB);
90.
91.     --     -----------------EIT------------------
92.
93.     v_EIT := round(v_NB * v_block_size);
94.     DBMS_OUTPUT.PUT_LINE('EIT: ' || v_EIT);
95.
96.     --     -----------------ENT------------------
97.
98.     v_ENT := round((p_new_estimated_records / v_NRB) * v_block_size);
99.     DBMS_OUTPUT.PUT_LINE('ENT: ' || v_ENT);
100.
101.
102.    select count(*)
103.    into v_count
104.    from PHYSICAL_PARAMETERS
105.    where TABLE_NAME = p_table_name;
106.
107.    if (v_count = 0) then
108.        insert into PHYSICAL_PARAMETERS(TABLE_NAME, TMR, ELB, NRB, NB, EIT,
    ENT)
109.        values (p_table_name, v_TMR, v_ELB, v_NRB, v_NB, v_EIT, v_ENT);
110.    else
111.        update PHYSICAL_PARAMETERS
112.        set TMR = v_TMR,
113.            ELB = v_ELB,
114.            NRB = v_NRB,
115.            NB  = v_NB,
116.            EIT= v_EIT,
117.            ENT = v_ENT
118.        where TABLE_NAME = p_table_name;
119.    end if;
120. end;
121. /
```

CALCULATION COUTCOMES

We performed our calculation using prepared procedure and according to default values set in oracle database for each table we obtained following results for five selected tables:

CLIENTS

Average size of registration (T.M.R)	135 bytes
Free space in the block (E.L.B)	7316 bytes
Number of records per block (N.R.B)	54
Number of blocks (N.B.)	2
Initial table space (E.I.T)	16384 bytes
Table next space (E.N.T)	30341 bytes *(with estimated 200 new insertions)*

HISTORY_GAMES

Average size of registration (T.M.R)	182 bytes
Free space in the block (E.L.B)	7316 bytes
Number of records per block (N.R.B)	40
Number of blocks (N.B.)	22
Initial table space (E.I.T)	180224 bytes
Table next space (E.N.T)	71680 bytes *(with estimated 350 new insertions)*

BETS

Average size of registration (T.M.R)	109 bytes
Free space in the block (E.L.B)	7316 bytes
Number of records per block (N.R.B)	67
Number of blocks (N.B.)	21
Initial table space (E.I.T)	172032 bytes
Table next space (E.N.T)	183403 bytes *(with estimated 1500 new insertions)*

HISTORY_ODDS

Average size of registration (T.M.R)	109 bytes
Free space in the block (E.L.B)	7316 bytes
Number of records per block (N.R.B)	67
Number of blocks (N.B.)	55
Initial table space (E.I.T)	450560 bytes
Table next space (E.N.T)	489075 bytes *(with estimated 4000 new insertions)*

HISTORY_ODDS

Average size of registration (T.M.R)	109 bytes
Free space in the block (E.L.B)	7316 bytes
Number of records per block (N.R.B)	67
Number of blocks (N.B.)	55
Initial table space (E.I.T)	450560 bytes
Table next space (E.N.T)	489075 bytes *(with estimated 4000 new insertions)*

HISTORY_ODDS

Average size of registration (T.M.R)	108 bytes
Free space in the block (E.L.B)	7316 bytes
Number of records per block (N.R.B)	67
Number of blocks (N.B.)	207
Initial table space (E.I.T)	1695744 bytes
Table next space (E.N.T)	1711761 bytes *(with estimated 14000 new insertions)*