COMP3311 20T3 Database Systems

COMP3311 20T3 Exam

Sample Solutions

These solutions are simply suggestions. In most cases many alternatives exist which would be equally correct and also worth full marks. Note that the order of tuples does not matter one bit in the SQL questions. The test scripts set the order themselves.

Q1

```
-- COMP3311 20T3 Final Exam
-- Q1: view of teams and #matches

create or replace view Q1(team, nmatches)
as
select t.country, count(*)
from   Teams t join Involves m on (m.team=t.id)
group  by t.country
```

Q2

```
-- COMP3311 20T3 Final Exam
-- Q2: view of players scoring several amazing goals

create or replace view Q2(player,ngoals)
as
select p.name as player, count(g.id) as goals
from   Players p join Goals g on (g.scoredBy = p.id)
where  g.rating='amazing'
group  by p.name
having count(g.id) > 1 ;
```

Q3

```
-- COMP3311 20T3 Final Exam
-- Q3: team(s) with most players who have never scored a goal

create or replace view PlayersAndGoals (player,team,ngoals)
as
select p.name, t.country, count(g.id)
from   Teams t
        join Players p on (p.memberof = t.id)
        left outer join Goals g on (p.id = g.scoredby)
group  by p.name, t.country ;

create or replace view CountryAndGoalless(team,nplayers)
as
select team, count(*) as players
from   PlayersAndGoals
where  ngoals = 0
group  by team ;

create or replace view Q3(team,nplayers)
as
select team, players
```

```
from   CountryAndGoalless
where  players = (select max(players) from CountryAndGoalless) ;
```

Q4

```
-- COMP3311 20T3 Final Exam
-- Q4: function that takes two team names and
--     returns #matches they've played against each other

create or replace function
    MatchesFor(text) returns setof integer
as $$
select m.id
from   Matches m
       join Involves i on (m.id = i.match)
       join Teams t on (i.team = t.id)
where  t.country = $1
$$ language sql;

create or replace function
    Q4(_team1 text, _team2 text) returns integer
as $$
declare
    nmatches integer;
begin
    perform * from Teams where country = _team1;
    if (not found) then return NULL; end if;
    perform * from Teams where country = _team2;
    if (not found) then return NULL; end if;
    select count(*) into nmatches
    from   ((select * from MatchesFor(_team1))
            intersect
            (select * from MatchesFor(_team2))
           ) as X;
     return nmatches;
end;
$$ language plpgsql;
```

Q5

```
-- COMP3311 20T3 Final Exam
-- Q5: show "cards" awarded against a given team

-- should have parameterised these views via an SQL function :-(

create or replace view RedCardsFor(team,ncards)
as
select t.country, count(c.id)
from   Players p
        join Teams t on (p.memberof = t.id)
        join Cards c on (c.givento = p.id)
where  c.cardtype='red'
group  by t.country ;

create or replace view RedCards(team,ncards)
as
select t.country, coalesce(c.ncards,0)
from   Teams t left outer join RedCardsFor c on (t.country=c.team) ;
```

```
create or replace view YellowCardsFor(team,ncards)
as
select t.country, count(c.id)
from   Players p
        join Teams t on (p.memberof = t.id)
        join Cards c on (c.givento = p.id)
where  c.cardtype='yellow'
group  by t.country ;

create or replace view YellowCards(team,ncards)
as
select t.country, coalesce(c.ncards,0)
from   Teams t left outer join YellowCardsFor c on (t.country=c.team) ;

drop function if exists q5(text);
drop type if exists RedYellow;

create type RedYellow as (nreds integer, nyellows integer);

create or replace function
        Q5(_team text) returns RedYellow
as $$
declare
        reds integer;
        yellows integer;
        result RedYellow;
begin
        select r.ncards, y.ncards into reds, yellows
        from   RedCards r
               join YellowCards y on (r.team = y.team)
        where  r.team = _team;
        if (not found) then
                result.nreds := NULL;
                result.nyellows := NULL;
        else
                result.nreds := reds;
                result.nyellows := yellows;
        end if;
        return result;
end;
$$ language plpgsql
;
```

Q6

```
-- q6.sql

drop view if exists Q6;
drop view if exists MatchScores;
drop view if exists TeamScores;
drop view if exists TeamsInMatches;
drop view if exists GoalsByTeamInMatch;

create view GoalsByTeamInMatch
as
select g.scoredIn as match, p.memberOf as team, count(*) as goals
from   Goals g join Players p on (p.id = g.scoredBy)
group  by g.scoredIn, p.memberOf;
;
```

```
create view TeamsInMatches
as
select i.match as match, i.team as team, t.country as country
from   Involves i join Teams t on (i.team = t.id)
;

create view TeamScores
as
select tim.match, tim.country, coalesce(gtm.goals, 0) as goals
from   TeamsInMatches tim left join GoalsByTeamInMatch gtm
        on (tim.team = gtm.team and tim.match = gtm.match)
;

create view MatchScores
as
select t1.match,
        t1.country as team1, t1.goals as goals1,
        t2.country as team2, t2.goals as goals2
from   TeamScores t1 join TeamScores t2
        on (t1.match = t2.match and t1.country < t2.country)
;

create view Q6
as
select m.city as location, m.playedOn as date,
        ms.team1, ms.goals1, ms.team2, ms.goals2
from   Matches m join MatchScores ms on (m.id = ms.match)
;
```

```
#!/usr/bin/python3
# COMP3311 20T2 Exam
# Q6: print match reports for a specified team in a given year

import sys
import psycopg2

def getResult(g1,g2):
   if g1 > g2:
      result = "won"
   elif g1 < g2:
      result = "lost"
   else:
      result = "drew"
   return result

db = None
cur = None

if len(sys.argv) < 3:
   print(f"Usage: {sys.argv[0]} TeamName Year")
   exit(1)
team = sys.argv[1]
year = sys.argv[2]
if not year.isnumeric:
   print(f"Invalid year {year}")
start_year = f"{year}-01-01"
end_year = f"{year}-12-31"

qT = "select count(*) from Teams where country = %s"
q6 = """
select *
```

```
from   q6
where  (team1 = %s or team2 = %s) and date between %s and %s
order  by date
"""

try:
   db = psycopg2.connect("dbname=footy")
   cur = db.cursor();
   cur.execute(qT, [team])
   tup = cur.fetchone()
   if not tup:
      print(f"No team '{team}'")
      exit(1)
   cur.execute(q6, [team,team,start_year,end_year])
   res = cur.fetchall()
   if len(res) == 0:
      print("No matches")
      exit(1)
   for tup in res:
      where,date,t1,g1,t2,g2 = tup
      if t1 == team:
         result = getResult(g1,g2)
         goals = f"{g1}-{g2}"
         opponent = t2
      else:
         result = getResult(g2,g1)
         goals = f"{g2}-{g1}"
         opponent = t1
      print(f"played {opponent} in {where} on {date} and {result} {goals}")
except psycopg2.Error as err:
        print("DB error: ", err)
finally:
   if db:
      db.close()
   if cur:
       cur.close()
```

Q7

```
#!/usr/bin/python3
# COMP3311 20T2 Final Exam
# Q7: print a specified player's career performance

# and, yes, John was naughty using a query inside a for loop ...

import sys
import psycopg2

db = None
cur = None

if len(sys.argv) < 2:
   print(f"Usage: {sys.argv[0]} PlayerName")
   exit(1)
player = sys.argv[1]

qPlayer = "select id,name from Players where name = %s";
qGames  = """
select m.id, m.city, m.playedOn
```

```
from   Teams t join Involves i on (i.team=t.id)
       join Matches m on (m.id=i.match)
       join Players p on (t.id=p.memberof)
where  p.id = %s
order  by m.playedOn
"""
qGoals = "select count(*) from Goals where scoredIn = %s and scoredBy = %s"
qTeam  = """
select t.country
from   Teams t join Players p on (t.id = p.memberof)
where  p.id = %s
"""

totMatches = 0
totGoals = 0

try:
   db = psycopg2.connect("dbname=footy")
   cur = db.cursor();
   cur.execute(qPlayer,[player])
   res = cur.fetchone()
   if not res:
      print("No such player")
      exit(1)
   pid,name = res
   cur.execute(qGames, [pid])
   for g in cur.fetchall():
      totMatches = totMatches + 1
      mid,city,date = g
      cur.execute(qGoals, [mid,pid])
      ngoals = cur.fetchone()[0];
      totGoals = totGoals + ngoals
      if ngoals == 0:
         continue
      elif ngoals == 1:
         goals = " and scored 1 goal"
      else:
         goals = f" and scored {ngoals} goals"
      print(f"played in {city} on {date}{goals}")
   cur.execute(qTeam, [pid])
   team = cur.fetchone()[0]
   print(f"Summary: played for {team}, {totMatches} matches, {totGoals} goals")
except psycopg2.Error as err:
       print("DB error: ", err)
finally:
   if cur:
       cur.close()
   if db:
      db.close()
```

Q8

a. **ER-style mapping** for subclasses:

```
create table Employee (
        id          integer,
        name        text,
        position    text,
        primary key (id)
);
create table PartTime (
        id          integer references Employee(id),
        fraction    float check (0.0 < fraction and fraction < 1.0),
        primary key (id)
);
create table Casual (
        id          integer references Employee(id),
        primary key (id)
);
create table HoursWorked (
        id          integer references Casual(id),
        onDate      date,
        starting    time,
        ending      time,
        primary key (id,onDate),
        constraint timing check (starting < ending)
);
```

We cannot enforce the total participation constraint (an employee may have no associated subclass tuples). We cannot enforce the disjoint subclasses constraint (an employee may have several associated subclass tuples).

b. **Single-table mapping** for subclasses:

```
create table Employee (
        id          integer,
        name        text,
        position    text,
        etype       text not null check (etype in ('part-time','casual')),
        fraction    float check (0.0 < fraction and fraction < 1.0),
        primary key (id),
        constraint  CheckValidTypeData
                       check ((etype = 'part-time' and fraction is not null)
                              or (etype = 'casual' and fraction is null))
);
create table HoursWorked (
        id          integer references Employee(id),
        onDate      date,
        starting    time,
        ending      time,
        primary key (id,onDate),
        constraint timing check (starting < ending)
);
```

With an appropriate `CheckValidTypeData` constraint we can enforce the disjoint subclass constraint. With the not null requirement on `etype`, we can enforce the total participation constraint. The `etype` field could be replaced by a boolean which checks `isCasual`.

It is also feasible to omit the `etype` field and simply assume that `fraction` being not null means that the employee is part-time.

In neither case can we enforce that part-time employees do not have hours-worked associated with them.

Q9

a. Trigger to handle adding a new `CourseEnrolments` tuple:

```
create function fixCoursesOnAddCourseEnrolment() returns trigger
as $$
```

```
declare
        _nS integer;  _nE integer;  _sum integer;  _avg float;
begin
        select nS,nE,avgEval into _nS,_nE,_avg
        from Courses where id=new.course;
        -- add one more student
        _ns := _nS + 1;
        if (new.stuEval is not null) then
                -- got another evaluation
                _nE := _nE + 1;
                if (_nS ≤ 10 or (3*_nE) ≤ _nS) then
                        -- added a new student, but still not enough for valid eval
                        _avg := null;
                else
                        -- compute new evaluation
                        select sum(stuEval) into _sum
                        from CourseEnrolments where course=new.course;
                        _sum := _sum + new.stuEval;
                        _avg := _sum::float / _nE;
                end if;
        end if;
        -- update Course record
        update Courses set ns = _nS, nE = _nE, avgEval = _avg
        where id=new.course;
        -- since "after" trigger, return value irrelevant
        return new;
end;
$$
language plpgsql;
```

b. Trigger to handle dropping a `CourseEnrolments` tuple:

```
create function fixCoursesOnDropCourseEnrolment() returns trigger
as $$
declare
        _nS integer;  _nE integer;  _sum integer;  _avg float;
begin
        select nS,nE,avgEval into _nS,_nE,_avg
        from Courses where id=old.course;
        -- we always add one more student
        _nS := _nS - 1;
        if (old.stuEval is not null) then
                -- lost an evaluation
                _nE := _nE - 1;
                if (_nS ≤ 10 or (3*_nE) ≤ _nS) then
                        -- no longer enough for valid eval
                        _avg := null;
                else
                        -- compute new evaluation
                        select sum(stuEval) into _sum
                        from CourseEnrolments
                        where course=old.course and student<>old.student;
                        _avg := _sum::float / _nE;
                end if;
        end if;
        -- update Course record
        update Courses set nS = _nS, nE = _nE, avgEval = _avg
        where id=old.course;
        -- since "after" trigger, return value irrelevant
        return old;
end;
```

```
$$
language plpgsql;
```

c. Trigger to handle updating a `CourseEnrolments` tuple:

```
create function fixCoursesOnModCourseEnrolment() returns trigger
as $$
declare
        _newEval integer;  _oldEval integer;
        _nE integer;  _nS integer;  _sum integer;  _avg float;
begin
        select nS,nE,avgEval into _nS,_nE,_avg
        from Courses where id=old.course;
        if (old.stuEval is null and new.stuEval is not null) then
                -- update involves adding evaluation
                _nE := _nE + 1;
        end if;
        -- treat NULL as zero for arithmetic
        _oldEval := coalesce(old.stuEval,0);
        _newEval := coalesce(new.stuEval,0);
        if (_oldEval <> _newEval) then
                -- compute new evaluation
                select sum(stuEval) into _sum
                from CourseEnrolments where course=old.course;
                _avg := (_sum - _oldEval + _newEval)::float / _nE;
        end if;
        -- update Course record
        update Courses set nS = _nS, nE = _nE, avgEval = _avg
        where id=old.course;
        -- since "after" trigger, return value irrelevant
        return new;
end;
$$
language plpgsql;
```

Q10

a. The code prints a list of teams and the number of matches they have played in each city.

b. The outer query (`teams`) is executed once, and returns 100 tuples (assumption). For each of these, one (inner) query (`count`) is executed. Total calls to `execute()` = 101.

c. Python code to achieve the same effect with a single query:

```
q = """
select t.country, m.city, count(*)
from   Teams t
       join Involves i on (i.team = t.id)
       join Matches m on (i.match = m.id)
group  by t.country, m.city
order  by t.country, m.city
"""
db = psycopg2.connect("dbname=footy")
cur = db.cursor()
cur.execute(q)
results = cur.fetchall()
for tuple in results:
   team, city, nmatches = tuple
   print(f"{t} {c} {n}")
```

Q11

a. FDs: A→BC, DE→F, ADE→G (also accept A→B, A→C instead of A→BC)

b.

Step	Attrs	FDs	Key	Notes
1	ABCDEFG	A→BC, DE→F, ADE→G	ADE	A→BC violates BCNF, LHS is partial key, so partition
2a	ABC	A→BC	A	No FDs violate BCNF, so ABC is part of solution
2b	ADEFG	DE→F, ADE→G	ADE	DE→F violates BCNF, LHS is partial key, so partition
3a	DEF	DE→F	DE	No FDs violate BCNF, so DEF is part of solution
3b	ADEG	ADE→G	ADE	No FDs violate BCNF, so ADEG is part of solution

Solution: three tables: ABC, DEF, ADEG (i.e. Student, Assessment, Mark)

Q12

a. Which employees earn more than $20 per hour (give their employee id and name)

```
Tmp1 = Sel[payRate>20]Employees
Res  = Proj[eno,ename]Tmp1
```

b. Who are the department managers (give just their name)

```
Tmp1 = Employees Join Departments  (on eno)
Res  = Proj[ename]Tmp1
```

c. Which employees worked on every day during the last week (give just their name)

```
Tmp1 = Proj[day]Timesheet
Tmp2 = Proj[eno,day]Timesheet
Tmp3 = Tmp2 / Tmp1
Tmp4 = Employees Join Tmp3  (on eno)
Res  = Proj[ename]Tmp4
```

Would expect to see division used ... if not, but still correct, ok, e.g.

```
Tmp1 = Proj[eno](Sel[day='Mon']Timesheet)
Tmp2 = Proj[eno](Sel[day='Tue']Timesheet)
...
Tmp7 = Proj[eno](Sel[day='Sun']Timesheet)
Tmp8 = Tmp1 Intersect Tmp2 Intersect ... Tmp7
Tmp9 = Employees Join Tmp8
Res  = Proj[ename]Tmp9
```