UiT
THE ARCTIC UNIVERSITY OF NORWAY

ASSESSMENT GUIDELINE

For exam in: **INF-2700 Database Systems**
Date: **Thursday 30.11.2017**

The assessment guideline contains 10 pages, including this cover page

Contact person: Weihai Yu.
Phone: 41429077

Question 1 (40%)

Below are some database tables with example data for a football application.

- Teams

tid	name
t1	team1
t2	team2
t3	team3

- Players

  The player `pid` with `name` plays for team `tid`.

pid	name	tid
p11	ola	t1
p12	odin	t1
p21	per	t2
p22	peter	t2
p31	martin	t3
p32	markus	t3

- Matches

  The match `mid` between two teams `tid1` and `tid2` was played on `date` at `start` time.

mid	tid2	tid2	date	start
m1	t1	t2	2017-10-01	18:00
m2	t3	t2	2017-11-01	18:00
m3	t3	t1	2017-11-20	17:00

- Goals

  The goal `gid` in match `mid` was scored by player `pid`.

  A `true` value of attribute `own` indicates an own-goal. An *own-goal* occurs when a player (mistakenly) sets the ball into the goal of the player's own team, resulting in a goal being scored for the opposition.

gid	mid	pid	own
g1	m1	p11	false
g2	m1	p21	true
g3	m1	p22	false
g4	m1	p11	false
g5	m3	p11	false
g6	m3	p12	false
g7	m3	p11	false
g8	m3	p12	false

The *primary keys* of the tables are in **bold** text.

Foreign key in Players:

- `tid`: references `tid` of Teams

Foreign keys in Matches:

- `tid1`: references `tid` of Teams
- `tid2`: references `tid` of Teams

Foreign keys in Goals:

- `mid`: references `mid` of Matches
- `pid`: references `pid` of Players

Write queries to find the required information.

Queries 1–5 must be formulated in *both relational algebra and SQL*.

Queries 6–10 need only be formulated in *SQL*.

Note: In the result tables of your SQL queries, there should be *no* identical (duplicate) rows.

Relational algebra *and* SQL (1–5):

1. List of different names of all players.

   The result for the example database is:

name
ola
odin
per
peter
martin
markus

$\Pi_{name} Players$

```
SELECT DISTINCT name
FROM   players;
```

2. Dates and start time of all `t1`'s matches.

   The result for the example database is:

date	start
2017-10-01	18:00
2017-11-20	17:00

$\Pi_{date,start}(\sigma_{tid1='t1' \vee tid2='t1'} Matches)$

```
SELECT date, start
FROM   matches
WHERE  tid1 = 't1' OR tid2 = 't1';
```

3. Names of all players of `team1`.

   The result for the example database is:

name
ola
odin

$\Pi_{Players.name}(Players \bowtie_{Players.tid=Teams.tid \wedge Teams.name='team1'} Teams)$

```
SELECT DISTINCT p.name
FROM   players p JOIN teams t USING(tid)
WHERE  t.name = 'team1';
```

4. Matches without goals. For each match, show the names of the teams, date and start time.

   The result for the example database is:

name	name	date	start
team3	team2	2017-11-01	18:00

$m \leftarrow Matches \bowtie (\Pi_{mid} Matches - \Pi_{mid} Goals)$
$\Pi_{t_1.name, t_2.name, date, start}(\rho_{t_1} Teams \bowtie_{t_1.tid = m.tid1} m \bowtie_{t_2.tid = m.tid2} \rho_{t_2} Teams)$

```
SELECT t1.name, t2.name, date, start
FROM   teams t1, teams t2, matches m
WHERE  t1.tid = m.tid1 AND t2.tid = m.tid2
       AND m.mid NOT IN (SELECT mid FROM goals);
```

5. Names of players who *only* made own-goals.

   The result for the example database is:

name
per

$\Pi_{name}(Players \bowtie (\sigma_{own=true} Goals - \sigma_{own=false} Goals))$

```
SELECT name
FROM   players
WHERE  pid IN (SELECT pid FROM goals WHERE own= true) AND
       pid NOT IN (SELECT pid FROM goals WHERE own= false);
```

SQL *only* (6–10):

6. Number of teams.

   The result for the example database is:

numberOfTeams
3

```
SELECT COUNT(*) AS number_of_teams
FROM   teams;
```

7. List of team names and the numbers of players of the teams,

   The result for the example database is:

name	numberOfPlayers
team1	2
team2	2
team3	2

```
SELECT t.name, COUNT(pid) as number_of_players
FROM   teams t JOIN players USING (tid)
GROUP BY tid;
```

8. List of players who scored at least two goals (own-goals are *not* counted).

   The list should show the names of the players and the numbers of goals the players scored. The players are listed in the descending order of the numbers of goals they scored.

   The result for the example database is:

name	numberOfGoals
ola	4
odin	2

```
SELECT name, COUNT(*) AS number_of_goals
FROM   players NATURAL JOIN goals
WHERE  own = false
GROUP BY pid
HAVING COUNT(*) > 1
ORDER BY COUNT(*) DESC;
```

9. The matches and scores of `team1`.

   The scores include the goals scored by the team and the own-goals made by the opponent team.

   The result for the example database is:

mid	scores
m1	3
m3	4

```
SELECT mid,COUNT(gid) AS scores
FROM   teams t JOIN players USING(tid) NATURAL JOIN goals
       JOIN matches m USING(mid)
WHERE  (t.tid = m.tid1 OR t.tid = m.tid2) AND
       ((t.name = 'team1' AND own = false) OR
        (t.name != 'team1' AND own = true))
GROUP BY mid;
```

10. Names of players who scored in all matches of the team (own-goals are *not* included).

    The result for the example database is:

name
ola

```
SELECT DISTINCT  p.name
FROM   players p
WHERE  NOT EXISTS
       (SELECT mid
        FROM   matches m
        WHERE  m.tid1 = p.tid OR m.tid2 = p.tid
        EXCEPT
        SELECT mid
        FROM   goals g
        WHERE  g.pid = p.pid AND own = false);
```

Question 2 (20%)

Now consider the physical data organization of the database in Question 1.

In the questions below, we will focus on queries like this one:

```sql
SELECT t1.name, t2.name
FROM   Matches m, Teams t1, Teams t2
WHERE  date = '2017-11-30' AND t1.tid = m.tid1 AND t2.tid = m.tid2;
```

The tables involved in the queries are organized as below:

- Table `Teams` is organized as a B$^+$-tree on attribute `tid`.
- Table `Matches` is organized as a B$^+$-tree on attribute `date`.

Answer the following questions.

1. What is the primary performance overhead of database systems in general?

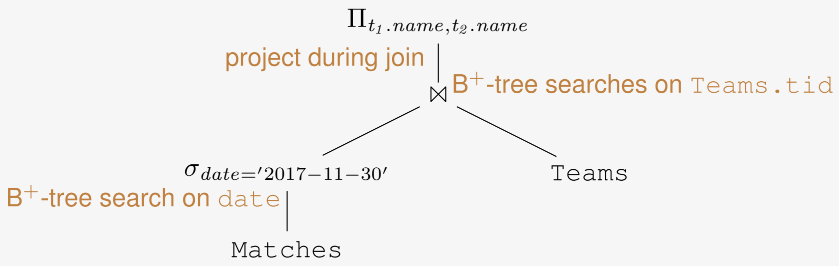
   Disk IOs. For hard disk: number of seeks and number of block transfers.
   Because seek time is usually much longer than the time for transferring a single block, random disk access is more expensive than sequential disk access.

2. Describe the file structure of the `Teams` table.

   B$^+$-tree file organization:

   - balanced tree
   - one block for a node
   - search-key (`tid` of `Teams`) values are ordered
   - non-leaf nodes: $\boxed{P_1 \mid K_1 \mid P_2 \mid \ldots \mid P_{m-1} \mid K_{m-1} \mid P_m}$
     - a B$^+$-tree has a fixed n
     - node *fanout* m: $\lceil \frac{n}{2} \rceil \leq m \leq n$
     - root: 2 (if also leaf, 0) $\leq m \leq n$
   - leaf nodes: $\boxed{P_{prev} \mid R_1 \mid \ldots \mid R_{k-1} \mid R_k \mid P_{next}}$
     - are linked
     - contain `Teams` records
     - also between half full and full (not measured with n, because records are larger than pointers)

3. Sketch an *execution plan* of the above query.



This is a possible execution plan:

- B$^+$-tree search of `Matches` on `date`. Let's name the search result `m`.

  The operation is performed first, because it is the most selective.

  The result can be kept in memory, because the size should be very small.

  The overhead is the depth of the `Matches` B$^+$-tree.

- Perform the join as B$^+$-tree searches of `Teams` on `tid`, using `m.tid1` and `m.tid2`.

  For each record in `m`, the `Teams` B$^+$-tree is searched twice.

  The overhead of each search is the depth of the `Teams` B$^+$-tree.

- The project to `t1.name`, `t2.name` is performed during the join.

4. What is the performance overhead of your execution plan?

One search on the `Matches` and two searches on `Teams` for every record in `m`.
Each search performs the number of random disk reads as the depths of the corresponding B$^+$-tree.

Question 3 (20%)

Answer the following questions. Please explain the relevant concepts while answering the questions.

1. What is *functional dependency* $X \to Y$ of a relation instance r?

Y's value is determined by X's value in r. More precisely, for any pair of tuples in r, if they have the same value in X, they also have the same value in Y.

For the relation instance below, check if the following functional dependencies are satisfied. If your answer is "no", explain why.

A	B	C
x	1	t
x	2	t
y	3	u
z	4	u

a) $A \rightarrow B$

No.

b) $A \rightarrow C$

Yes.

c) $AB \rightarrow C$

Yes.

d) $AC \rightarrow B$

No.

2. What is a *superkey* of a relation schema?

A set of attributes K that uniquely identifies tuples in the relation. In any instance of relation schema R, no two distinct tuples have the same value on all attributes in K.

Can you define a superkey using functional dependencies?

$K \rightarrow R$

3. What is a relation schema in *Boyce-Codd Normal Form* (BCNF)?

For schema R with a set of functional dependencies F, R is in BCNF if for any $\alpha \rightarrow \beta \in F$, one of the following is true:

- $\beta \in \alpha$ ($\alpha \rightarrow \beta$ is trivial),
- α is a superkey for R ($\alpha \rightarrow R$).

4. We have a relation schema $Addresses\ (stname, stnr, postcode, city)$, where $stname$ stands for "street name" and $stnr$ for "street number".

The $Addresses$ schema has the following functional dependencies:

- $\{stname, stnr, city\} \rightarrow postcode$
- $postcode \rightarrow city$

The $Addresses$ schema is *not* in BCNF. Why? What is the problem with not being in BCNF?

Because of $postcode \rightarrow city$ and $postcode$ is not a superkey of $Addresses$.
The problem is redundancy in data. If two rows have the same $postcode$ value, they must also have the same $city$ value. This will lead to (insertion, update and deletion) anomalies.

5. How would you solve the problem?

Decompose $Addresses$ into BCNF schema $Addr(stname, stnr, postcode)$ and $Postcity(postcode, city)$.
The decomposition is lossless because the common attribute $postcode$ is a superkey of $Postcity$.

6. Does your solution introduce any new problem?

Yes. The decomposition does not preserve the functional dependency $\{stname, stnr, city\} \rightarrow postcode$.

Question 4 (20%)

1. What is an *ACID transaction*?

A transaction is a group of operations on shared (database) data.

Atomicity The final effect on the data is all or nothing.

Consistency A transaction, if executed in isolation, meets its specification (dynamic consistency) and keeps the database consistent (static consistency).

Isolation Interleaved executions of concurrent transactions have the same effect as isolated (serial) executions.

Durability If a transaction commits, the result is not affected by possible subsequent undesirable events.

2. Is the following transaction schedule *serializable*? Explain why.

$$read_1(x), read_2(y), write_1(x), read_2(x), write_2(x), commit_2, commit_1$$

A schedule is *serializable* if its effects is equal to a serial execution of the transactions.
The schedule *is* serializable because the effect is equal to serial execution T_1, T_2. The two transactions conflict on x and T_1 accesses x first.

3. Is the above transaction schedule *strict*? Explain why.

A schedule is *strict* if there is no dirty read or dirty write.
The schedule is *not* strict because $read_2(x)$ is dirty (since $write_1(x)$ has not committed).

4. Describe a concurrency control mechanism that enforces serializable and strict transaction schedules.

Strict two-phase locking (S2PL) enforces serializable and strict schedules.

- A transaction must acquire a shared lock in order to read the data item and an exclusive lock to write. A transaction has to wait when another transaction holds a lock in conflicting mode. Two locks on a data item conflict if at least on of them is exclusive.
- With 2PL, all transactions have two phases: in the *growing phase*, a transaction can only acquire locks; in the *shrinking phase*, a transaction can only release locks.
- 2PL enforces serializable schedules. Conflicting transactions are serialized in the order they change phases (i.e. they start releasing locks).
- With S2PL, exclusive locks (write locks) are released at the termination of the transactions.
- S2PL enforces strict schedules.

–END–