

where the attributes and tables have the following meaning

Song: *code*: code of the song
title: title of the song
music: author of the music
lyrics: author of the lyrics

Edition: *num*: number of the contest edition
city: location for the contest
date: date of the contest

Performance: In the edition *edition* the country *country* participates with the song *song* performed by *performer* and obtaining the score *score*.

Voting: In the edition *edition* the country *voter* assigns to the country *voted* a total of *points* points.

And consider the following extension of the previous schema. We will refer to this extension as database (DB):

Song			
code	title	music	lyrics
1	La plaza	José Luis	Juan Carlos
2	La fuente	Juan	Jaime
3	El árbol	José María	Ana
4	El coche	Sergio	
5	Saludo		David

Edition		
num	city	date
1	Valencia	12/3/2005
2	Barcelona	14/6/2004
3	Madrid	10/5/2005

Performance				
edition	country	song	performer	score
1	Spain	1	Andrés	30
1	France	2	Jean	15
1	Portugal	4	José	25

Voting			
edition	voter	voted	points
1	Spain	France	10
1	Portugal	France	5
1	France	Portugal	10
1	France	Spain	15
1	Spain	Portugal	15
1	Portugal	Spain	15

FINAL EXAM: "DATABASES" – 10/06/05 – Questionnaire Type A

1. What would be the final state of the tables *Performance* and *Voting* after executing the following instruction: UPDATE Performance SET country='Italy' WHERE country='Portugal' over the database DB?

a) Nothing changes because the instruction violates the integrity constraint defined in the table *Performance* over the foreign key *edition*.

b)

Performance				
edition	country	song	performer	score
1	Spain	1	Andrés	30
1	France	2	Jean	15
1	Italy	4	José	25

The other relations remain the same.

c)

Performance				
edition	country	song	performer	score
1	Spain	1	Andrés	30
1	France	2	Jean	15
1	Italy	4	José	25

Voting			
edition	voter	voted	points
1	Spain	France	10
1	Italy	France	5
1	France	Portugal	10
1	France	Spain	15
1	Spain	Portugal	15
1	Italy	Spain	15

The other relations remain the same.

d)

Performance				
Edition	country	song	performer	score
1	Spain	1	Andrés	30
1	France	2	Jean	15
1	Italy	4	José	25

Voting			
Edition	voter	voted	points
1	Spain	France	10
1	Italy	France	5
1	France	Italy	10
1	France	Spain	15
1	Spain	Italy	15
1	Italy	Spain	15

The other relations remain the same.

2. Suppose that all the integrity constraints are DEFERRABLE INITIALLY IMMEDIATE and let T1 be a transaction which is executed over the Oracle DBMS:

```
TRANSACTION T1
  SET CONSTRAINT ALL DEFERRED;
  INSERT INTO Voting VALUES (1, 'Brasil', 'France', 5);
  INSERT INTO Voting VALUES (1, 'Brasil', 'Brasil', 15);
  INSERT INTO Performance VALUES (1, 'Brasil', 3, 'Jordinho', 0);
COMMIT;
```

- The transaction works correctly because the checking of the integrity constraints is performed when the transaction is finished and everything is ok.
 - The transaction fails because a country cannot vote to itself and consequently the database is not modified.
 - The transaction fails because a country cannot vote to itself, but a tuple is added to the DB in the table *Voting* and another tuple in the table *Performance*.
 - The transaction fails because we cannot change the strategy for checking constraints.
3. Which of the following statements is FALSE?
- In order to insert a new song it is necessary to include the author of the music and the lyrics.
 - We cannot repeat the song in several performances.
 - A country cannot vote to itself.
 - When adding a new performance, we must indicate who the performer is.

4. Which query over the WORKING SCHEMA does this relational algebra expression solve?

$$((\text{Performance}[\text{song}] - ((\text{Performance}[\text{song}, \text{score}] \times \text{Performance}[\text{song}, \text{score}] ((\text{song}, c2), (\text{score}, p2)))) \text{WHERE score} > p2) [\text{song}])(\text{song}, \text{code}) \bowtie \text{Song}) [\text{title}]$$

- Titles of the songs with the lowest score of any edition.
 - Titles of the songs with the highest score of any edition.
 - Titles of the songs which have participated exactly in two editions of the contest with different scores.
 - Titles of the songs which have participated at least in two editions of the contest.
5. Which statement regarding the WORKING SCHEMA is TRUE?
- If the foreign key {edition, voter} in the relation *Voting* were defined as a partial referential integrity, we could have a vote without knowing the country which has made the vote.
 - The same country can participate twice in the same contest edition with different songs.
 - A modification of the value of the attribute *edition* in the relation *Performance* might cause the modification, in cascade, of the attribute *num* in *Edition*.
 - In order to ensure that there are not two songs with the same title it is necessary to add a uniqueness constraint over the attribute *title* in the relation *Song*.

6. Given the following SQL query:

```
SELECT * FROM SONG C
WHERE NOT EXISTS (SELECT * FROM PERFORMANCE A WHERE C.code=A.song);
```

Which of the following expressions in relational algebra would be equivalent to this query?

- $(\text{Song}(\text{cod}, \text{song}) \bowtie (\text{Performance} \text{ WHERE Edition IS NULL AND country IS NULL})) [\text{song}, \text{title}, \text{music}, \text{lyrics}]$;
- $\text{Song} - (\text{Song} \bowtie \text{Performance}[\text{song}] (\text{song}, \text{cod}))$
- $\text{Song} \bowtie (\text{Song}[\text{cod}] \cap \text{Performance}(\text{song}, \text{cod})[\text{cod}])$
- $\text{Song} \bowtie (\text{Song}[\text{cod}] \cup \text{Performance}(\text{song}, \text{cod})[\text{cod}])$

7. Given the following integrity constraint:

```
CREATE ASSERTION r1 CHECK
  (NOT EXISTS(SELECT * FROM Performance A
              WHERE NOT EXISTS(SELECT * FROM Song C
                               WHERE C.code = A.song))));
```

Which consequences would it have if we include this constraint in the WORKING SCHEMA?

- We need to include this constraint to indicate that there cannot be songs in the database if they do not have an assigned performance.
- We cannot include the definition of this constraint in the database since it would preclude the correct functioning of the schema because it would not allow countries with assigned songs to participate.
- We could only implement this with SQL if we use *triggers*.
- It can be included but it is redundant because this constraint is already covered by other constraints in the WORKING SCHEMA.

8. Suppose that all the secondary storage of the database is moved from disk C:\ to disk W:\. Which of the following instructions would be necessary taking into account that the *binding* between schemas is performed when the execution of each application that uses the database is initiated?

- No action should be performed because the database structure is the same.
- All the applications should be recompiled after the storage change.
- It would be enough to re-initiate the execution of the applications after the change.
- There is no need of any action if we just redefine some views in the External Schemas.

9. Consider the WORKING SCHEMA, which of the following statements is TRUE when a song is deleted?

- No song can be deleted because deletion has been defined as “restrictive” with respect to relation *Performance*.
- Any song can be deleted without problems because the update has been defined on cascade with respect to relation *Performance*.
- A song can only be deleted if it does not appear in the relation *Performance*.
- We can never delete a song because every song has a performance, as it is indicated with the NNV constraint in the attribute *song* of the relation *Performance*.

10. On the WORKING SCHEMA, the constraint which indicates that in every tuple in *Voting* the country which votes (*voter*) must be different from the country which is voted (*voted*), can be implemented in SQL through:

- CREATE ASSERTION R2
CHECK (NOT EXISTS (SELECT * FROM VOTING V
 WHERE VOTED <> VOTER))
- This can be implemented through a typical domain constraint.
- This constraint cannot be implemented.
- Through a table constraint in the definition of the table *Voting* of the form: CHECK (VOTED <> VOTER)

11. When executing the following instruction over the database DB:

```
DELETE FROM Voting WHERE voter = 'Portugal';
```

- This deletion is not allowed because it violates the foreign key constraints in *Voting*.
- It can only be allowed if we would change the clause of RESTRICTED deletion to ON DELETION CASCADE in the foreign keys in the table *Voting*.
- It can only be allowed if we would change the clause of RESTRICTED deletion to ON DELETION SET TO NULL in the foreign keys in the table *Voting*.
- The following tuples would be deleted (1, 'Portugal', 'France', 5) and (1, 'Portugal', 'Spain', 15).

12. Given the following view definition over the WORKING SCHEMA:

```
CREATE VIEW PrimEdic  
AS SELECT * FROM Performance A WHERE A.Edition=1  
WITH CHECK OPTION;
```

and the instruction:

```
INSERT INTO PrimEdic VALUES (1, 'Italy', 3, 'J&R', 0);
```

Which of the following statements is TRUE?

- It cannot be executed because we cannot perform data modifications on views.
- The tuple (1, 'Italy', 3, 'J&R', 0) would be added to the relation *Performance*.
- It cannot be executed because the clause WITH CHECK OPTION has been included in the view definition.
- The tuple (1, 'Italy', 3, 'J&R', 0) would be added to the view *PrimEdic*, but not to the relation *Performance*.

13. Given the database DB, consider the instruction

```
INSERT INTO Voting VALUES (1, null, 'Spain', 10);
```

Which of the following statements is TRUE?:

- This insertion could only be done if the referential integrity of the FK: {edicion, voter} is partial.
- This insertion could only be done if the referential integrity of the FK: {edicion, voter} is partial or weak.
- This insertion could only be done if we add to the table *Performance* a tuple with Edition=1 and country=null.
- This insertion cannot be performed in any case.

14. On the WORKING SCHEMA, the maximum cardinality of the relation *Performance* is:

- Infinite.
- The cardinality of the relation *Song*.
- The cardinality of the relation *Edition*.
- The product of the cardinality of *Song* times the cardinality of *Edition*.

FINAL EXAM: DATABASES – 10/06/05 – Problems

Given the WORKING SCHEMA presented before, solve the following exercises in standard SQL:

1. List the performers which have participated in two editions with consecutive numbers representing the same country. (0.5 points)
2. List the countries which have never given points to 'Spain' (0.5 points)
3. List the authors which have composed both the music and the lyrics of songs which have attained the maximum score in some edition (0.75 points)
4. List the title of the song, the city in which was performed, the country it was representing, of those songs which received more than 3 "votings" with more than 5 points each. (0.75 points)
5. List the city and the date of those editions in which all the "votings" which Spain has received have been greater than 7 points. (We assume that 'Spain' has received at least a point in one voting in each edition) (0.75 points)
6. List, for each and every performance, the edition, the country the performance is representing, and how many countries have voted for the performance. (1 point)
7. List the number and the city of the editions where **most** countries have participated, also indicating the number of countries which participated. (1 point)
8. The attribute *score* of the relation *Performance* is a derived attribute which is obtained by summing the *points* obtained by that performance:
 - a. Enumerate the events over the database which may affect the value of the derived attribute. (0.75 points)
 - b. Design a trigger (*trigger*) in SQL to update *score* whenever there is a deletion in *Voting*. (0.5 points)

SOLUTIONS TO THE QUESTIONNAIRE

1	d)
2	b)
3	a)
4	a)
5	d)
6	b)
7	d)
8	c)
9	c)
10	d)
11	d)
12	b)
13	d)
14	b)

SOLUTIONS TO THE PROBLEMS:

1. SELECT DISTINCT A1.performer
FROM Performance A1, Performance A2
WHERE A1.edition=A2.edition -1 AND A1.country=A2.country AND
A1.performer=A2.performer;
2. SELECT DISTINCT country
FROM Performance A
WHERE NOT EXISTS (SELECT * FROM Voting V
WHERE A.country =V.votante AND V.voted = 'Spain');
3. SELECT DISTINCT music
FROM Song
WHERE music = lyrics AND
code IN (SELECT song FROM Performance A
WHERE score = (SELECT MAX(score)
FROM Performance A2
WHERE A2.edition = A.edition));

4. SELECT C.title, E.city, A.country
 FROM Song C, Edition E, Performance A
 WHERE A.song=C.cod AND A.edition=E.num AND
 (SELECT COUNT(*) FROM Voting
 WHERE edition=E.num AND voted=A.country AND points>5) >3;

or also:

SELECT C.title, E.city, A.country
 FROM Song C, Edition E, Performance A, Voting V
 WHERE A.song=C.cod AND A.edition=E.num AND V.edition=E.num
 AND V.voted=A.country AND V.points>5
 GROUP BY A.edition, A.country, C.title, E.city
 HAVING COUNT(*)>3;

5. SELECT city, date
 FROM Edition E
 WHERE NOT EXISTS (SELECT *
 FROM Voting V
 WHERE V.edition=E.num and V.voted='Spain' and V.points<=7);

6. SELECT A.edition, A.country, COUNT(V.votante)
 FROM Performance A LEFT JOIN Voting V
 ON A.edition = V.edition AND A.country = V.voted
 GROUP BY A.edition, A.country;

7. SELECT E.num, E.city, COUNT(*)
 FROM Edition E, Performance A
 WHERE E.num=A.edition
 GROUP BY E.num, E.city
 HAVING COUNT(*) = (SELECT MAX(COUNT(*)) FROM Performance GROUP BY
 edition);

8 a. INSERT into Voting,
 DELETE from Voting,
 UPDATE edition, voted or points in Voting

As well as the operations:

INSERT into Performance
 UPDATE score in Performance
 which must be controlled

8 b. CREATE TRIGGER delete_voting
 AFTER DELETE ON Voting
 REFERENCING OLD AS MyOld
 FOR EACH ROW
 BEGIN ATOMIC
 UPDATE Performance SET score = score - MyOld.points
 WHERE edition = MyOld.edition AND country = MyOld.voted;
 END;