Test Pearl 010 — Databases

Pearls of Computer Science (201700139)

Bachelor module M1, Technical Computer Science, EWI

20 september 2018, 08:45–09:45

Module coordinator: Doina Bucur

Instructor: Maurice van Keulen

- You may use 1 A4 document with your own notes for this exam and a *simple* calculator.
- Scientific or graphical calculators, laptops, mobile phones, books etc. are not allowed. ***Put those in your bag now!***

Total number of points: 40.
Total number of pages: 5.

1 *SQL (20 points)*

A student house in Enschede uses a database system for registering the wine consumption of the residents of the house. A description of the table structure of this database can be found in Figure 2. Every time when new wine is bought by the wine commissioner (the person who is responsible for purchasing wine), new rows are inserted in the WineStock table. For each crate of wine, the price per bottle is stored (prices for wine vary often). The system stores the date/time and from which stock the wine was consumed. Each month, the wine commissioner calculates the costs per person.

Figure 3 contains some example data. There are four residents of which Bob Idontdrink (resid 43) has consumed no wine at all. The other three have consumed several bottles, most on 13/09/18. The 8 bottles of Zonin (brandid 12), which is a Merlot wine (typeid 2), are bought in two purchases: 2 bottles for 2.95 (stockid 21) and 6 bottles for 2.75 (stockid 22). The most expensive wine that was bought is Zonnebloem (brandid 11), also a Merlot wine (typeid 2). They only bought 2 bottles: one for 12.50 (stockid 24) and one for 10.95 (stockid 23). The latter was consumed by Michael Broadbent (resid 40) on 13/09/18 (conid 32). If something is still unclear about this database, just raise your hand so that the teacher can explain.

Tip: Figure 4 contains an informal description of the syntax of SQL.

(a) Give an SQL query that produces an overview of the first and last names of all residents sorted by lastname.
(b) Give an SQL query that lists all names of wine brands from Italy.
(c) Give an SQL query that lists all brand names with their wine type of all wines from Australia sorted by brand name.
(d) Give an SQL statement that changes the location of the brand "Zonin" to "Europe".
(e) Give an SQL query that produces an overview with per resident how many bottles of wine they drank.
(f) Give an SQL query that lists all wines consumed by Michael Broadbent with brand name, wine type, date of consumption, and the number of bottles consumed.
(g) Give an SQL statement that adds the wine brand "Olarra", which is a Rioja wine from Spain, to the database. You are allowed to directly use the typeid '3' for "Rioja".
(h) Give an SQL query that determines the maximum price of a bottle that has been consumed (the result of the query should be 10.95, because the 12.50 bottle has not been consumed).

□

Answer to 1. In the evaluation of these questions, special attention was given to correct usage of

1. join conditions (J),
2. value conditions (C),
3. group by and count, i.e., aggregation (A),
4. inserts, deletes, and updates (U), and
5. SQL in general, i.e., SELECT-FROM-WHERE-DISTINCT-ORDERBY (S)

(a) This question requires a selection of attributes and an order by (S).

```
SELECT firstname, lastname
FROM Resident
ORDER BY lastname
```

(b) Simple query with one condition (C)

```
SELECT brandname
FROM WineBrand
WHERE location = "Italy"
```

(c) A query with a join (J), a condition (C) and an order by (S)

```
SELECT wb.brandnaam, wt.typenaam
FROM WineBrand wb INNER JOIN WineType wt ON wb.typetId = wt.typeId
WHERE wb.location = "Australia"
ORDER BY wb.brandname
```

or

```
SELECT wb.brandnaam, wt.typenaam
FROM WineBrand wb, WineType wt
WHERE wb.location = "Australia"
  AND wb.typeid = wt.typeid
ORDER BY wb.brandname
```

(d) An update statement (U) with one condition (C).

```
UPDATE WineBrand
SET location = "Europe"
WHERE brandname = "Zonin"
```

(e) An aggregate query with a group by (A) and a join (J).

```
SELECT firstname, lastname, SUM(number)
FROM Resident r, Consumption c
WHERE r.resid = c.resid
GROUP BY firstname, lastname
```

(f) All tables joined (4J) and 2 conditions (2C). Note that the right attribute for number of bottles consumed should be used: “number” and not “nrob” (which is the number of bottles *bought*).

```
SELECT wb.brandname, wt.typename, date, number
FROM WineType wt, WineBrand wb, WineStock ws, Consumption c, Resident r
WHERE r.firstname = "Michael"
AND r.lastname = "Broadbent"
AND c.resid = r.resid
AND c.stockid = ws.stockid
AND wb.brandid = ws.brandid
AND wb.typeid = wt.typeid
```

(g) A simple INSERT statement (U). The part “(brandid, brandname, location, typeid)” may be omitted.

```
INSERT INTO WineBrand (brandid, brandname, location, typeid)
VALUES (13, "Olarra", "Spain", 3)
```

(h) An aggregate query without group by (A) but with a join.

```
SELECT MAX(ppb)
FROM WineStock ws, Consumption c
WHERE ws.stockid = c.stockid
```

2 *Databases (10 points)* For these questions, please use correct database terminology as much as possible.

(a) What is the difference between *data distribution* and *data replication*?
(b) SQL is at the same time a QL, DML, DDL, and SDL. To which of these kinds of languages does the CREATE-statement belong?
(c) Explain the difference between the terms “database”, “database server”, and “database management system”. Base your answer on the precise technical meaning of the terms, not on how they are typically (mis)used.
(d) An application uses SELECT queries, UPDATE statements, and INSERT statements on a certain table. They decide to create an index on this table. Do the SELECTs become faster or slower? Same question for the UPDATEs and for the INSERTs. Explain your answers.

□

Answer to 2. These questions can all be answered based on the information from the book chapter “Databases” and the slides of the lectures.

(a) Data distribution *divides* the data into chunks and each chunk is placed on a different computer. Data replication places *copies* of the data on different computers.
(b) *DDL*. This is the Data Definition Language and CREATE defines a data structure.
(c) “Database” is the actual data (the tables with the contents). “Database server” is the *computer* where the database is stored or hosted. “Database management system” is the *software* running on this server manipulating the data in the database.
(d) An index is supposed to make (certain) *queries faster*. But because it now needs to maintain this index, the UPDATEs and INSERTs become slower (the changes affect not only the table, but also the index).

3 *Database design (10 points)*

Figure 1 contains part of the datamodel belonging to a database storing information about facilities and attractions all over the world. The FacilityCategory entity is meant to only represent categories of facilities, such as “Police station”, “Public swimming pool”, “Museum”, etc. Obviously each city can have more than one facility category and several cities can have

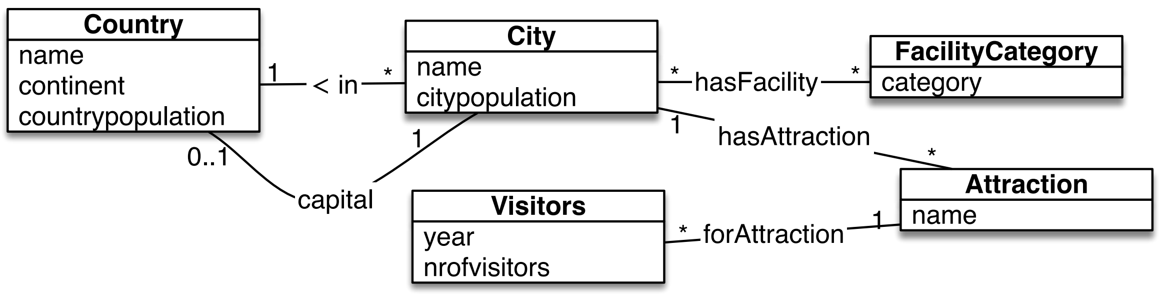

Figure 1: ER model of a purchasing database.

the same facility category (e.g., "Museum": we do not store the individual museums but just the category "Museum"). Cities can also have "Attractions", but these are stored individually, such as the amusement park "Efteling". The entity "Visitors" is meant to keep track of the number of visitors per year for each attraction. For each city, its name and citypopulation is stored as well as in which country the city is. About the country, the name, continent and countrypopulation is stored as well as which city is its capital.

- Given this ER-model, design a table structure for this model as a list of tables with for each table: (i) the name of the table, (ii) the names of the attributes (iii) which attribute(s) form the primary key, and (iv) which attribute(s) are foreign keys and what they refer to.

□

Answer to 3.

Primary keys in **bold**.

Country: **cid**, name, continent, countrypopulation, *capital* *capital is a foreign key referring to the cityid in table City.*
City: **cityid**, name, citypopulation, *in* *'in' is a foreign key referring to the cid in table Country.*
Attraction: **aid**, name, *in* *'in' is a foreign key referring to the cid in table City. It represents the relationship "hasAttraction"*
Visitors: **vid**, year, nrofvisitors, *forAttraction* *'forAttraction' is a foreign key referring to the aid in table Attraction.*
FacilityCategory: **fid**, category *no foreign keys*
hasFacility: **fid**, **cid** This table represents the relationship "hasFacility". Since it is a n:m relationship, it needs its own table. The primary key is (fid,cid) together. fid is a foreign key referring to the fid in table FacilityCategory. cid is a foreign key referring to the cid in table City.

Table	**Attributes / Description**
WineType	**typeid**, typename, ingredients
	Table for storing wine types. typeid is a unique number for each wine type, typename is the name of the wine type (e.g, Chardonnay), ingredients is a description of the main ingredients (e.g., grapejuice).
WineBrand	**brandid**, brandname, location, *typeid*
	Table for storing different wine brands. brandid is a unique number for each wine brand, brandname is the name of the wine, location is the location where the wine brand is brewn (e.g., France), typeid is a reference to the WineType table indicating the type of wine.
WineStock	**stockid**, *brandid*, ppb, nrob, deposit
	When a new crate is bought by the wine commissioner, a new row will be added to this table. stockid is a unique number for the current stock purchase, brandid is a reference to the WineBrand table (what brand of wine was bought), ppb is the price per bottle, nrob is the number of bottles bought, deposit is the deposit per bottle (e.g., €0.10 per bottle).
Consumption	**conid**, *resid*, *stockid*, date, number
	This table keeps track of the consumption history of the residents. conid is a unique number for a consumption (i.e., drinking) session, resid is a reference to the Resident table (who drank the wine), stockid is a reference to the WineStock table (what was consumed), date is the date and time of consumption, amount is the number of bottles that was consumed.
Resident	**resid**, firstname, lastname
	The table Resident contains all the residents of the house. resid is a unique number for each person in the house, firstname is the firstname of the resident, lastname is the lastname of the resident.

Figure 2: Table structure of a student house's "wine consumption" database (Primary keys in **bold**; Foreign keys in *italics*).

WineType

typeid	typename	ingredients
1	Chardonnay	...
2	Merlot	...
3	Rioja	...

WineBrand

brandid	brandname	location	typeid
10	Hardys VR	Australia	1
11	Zonnebloem	S. Africa	2
12	Zonin	Italy	2

WineStock

stockid	brandid	ppb	nrob	deposit
20	10	5.95	6	0.10
21	12	2.95	2	0.10
22	12	2.75	6	0.10
23	11	10.95	1	0.25
24	11	12.50	1	0.10

Consumption

conid	resid	stockid	date	number
30	40	21	12/09/18	2
31	41	22	13/09/18	1
32	40	23	13/09/18	1
33	42	20	13/09/18	3

Resident

resid	firstname	lastname
40	Michael	Broadbent
41	Jean-Charles	Boisset
42	Randall	Grahm
43	Bob	Idontdrink

Figure 3: Example data for the "wine consumption" database

In the informal syntax, we use the following notations

- $A|B$ to indicate a choice between A and B
- $[A]$ to indicate that A is optional
- $A*$ to indicate that A appears 0 or more times
- $A+$ to indicate that A appears 1 or more times
- 'A' to indicate that the symbol A is literally that symbol

We are not precise in punctuation in the syntax, but this is irrelevant in this exam anyway.

SQL

createtable : `CREATE TABLE` *tablename* '(' *columndef*+ *constraint*∗ ')'
columndef : *colname type* [`NOT NULL`] [`UNIQUE`] [`PRIMARY KEY`]
[`REFERENCES` *tablename* (*colname*+)]
constraint : `PRIMARY KEY` (*colname*, ...) | `CHECK` (*condition*)
| `FOREIGN KEY`(*colname*, ...) `REFERENCES` *tablename*(*colname*, ...)
query : `SELECT` (*column* [`AS` *colname*]) +
`FROM` (*tablename* [`AS` *colname*])+
`WHERE` *condition* [`GROUP BY` *column* +] [`ORDER BY` *column* +]
column : [*tablename* '.'] *colname* | ' ∗ '
Examples of *condition*: *column* = *value* [(`OR` | `AND`) [`NOT`] *column* <> *value*]
| *column* `IS` [`NOT`] `NULL`
| *column* [`NOT`] `IN` (*value*, ...) ...
statement : *delete* | *update* | *insert*
delete : `DELETE FROM` *tablename* `WHERE` *condition*
update : `UPDATE` *tablename* `SET` (*column* = *value*) ∗
`FROM` (*tablename* [`AS` *colname*])+
`WHERE` *condition*
insert : `INSERT INTO` *tablename* ['(' *colname* + ')']
query | `VALUES` '(' *value* + ')'

Figure 4: Informal syntax of SQL