Test Pearl 010 — Databases

Pearls of Computer Science (201300070)
Bachelor module 1.1, Technical Computer Science, EWI

21 september 2017, 08:45–09:45

Module coordinators: Maurice van Keulen, 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)*

For this question, we use the tables belonging to a 'flight database' database. A description of the table structure can be found in Figure 2. It is a simplification of the example used in the lectures. The database stores flights, airports, passengers, and bookings.

Figure 3 contains some example data. There are three passengers who made in total 4 bookings ('Maurice' made two bookings). There are quite some flights in the database. As you may notice, the same flight number is being re-used multiple times. The reason is that, in practice, a flight number refers to a certain flight in a week, for example, 'KL123' is the flight number of the Monday morning flight from Amsterdam to Vienna, which goes every week. Therefore, the primary key of this table is the combination of flight number and date. Otherwise, the tables and data speak for themselves; if something is still unclear, just raise your hand so that the teacher can explain.

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

(a) Write an SQL query that produces the city and country of the airport with code "MST".
(b) Write an SQL query that produces all flight numbers and dates for all flights booked by a passenger with name "Maurice".
(c) Write an SQL query that produces per date how many flights depart from an airport in The Netherlands.
(d) Passenger Arend wants to cancel all his bookings. Write an SQL statement that deletes all bookings for the passenger with id 3.
(e) Write an SQL query that produces the names of all passengers who depart from the airport in city "Amsterdam".
(f) Passenger Pieter-Tjerk wants to postpone his flight to one week later. Write an SQL statement that changes his booking of '19/09/17': the date should become '26/09/17'. The rest of the data remains the same (flight number, booking date, etc.).
(g) Write an SQL query that produces the flight numbers and times of all flights that fly from or to Vienna. Note that such flights happen every week and we don't want the result to contain the same rows more than once (e.g., 'KL123' with time 10:45 occurs three times, but we only want it once in the result).

□

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), plays a role in all questions..

(a) This question only required a value condition (C).

```
SELECT city, country
FROM airport
WHERE code = "MST"
```

(b) Simple join query (J, C)

```
SELECT flightnumer, date
FROM passenger p, booking b
WHERE p.pas_id = b.pas_id
  AND p.name = "Maurice"
```

Note that only querying table “booking” with a value condition “pas_id=1” is not correct. The question does not ask for all bookings with a passenger id 1, this number is not in the question, only the name of the passenger.

(c) Group by query with a join and value condition (A,J,C).

```
SELECT date, count(*)
FROM flight f, airport a
WHERE f.from = a.code
  AND a.country = "Netherlands"
GROUP BY f.date
```

Note that both the `GROUP BY` as well as the `COUNT` are needed and the same attributes should be mentioned with both.

(d) A delete statement with a value condition (U, C)

```
DELETE FROM booking WHERE pas_id = 3
```

(e) This query joins all 4 tables (J, J, J, C).

```
SELECT p.name
FROM passenger p, booking b, flight f, airport a
WHERE a.city = "Amsterdam"
  AND a.code = f.from
  AND f.flightnumber = b.flightnumber
  AND f.date = b.date
  AND b.pas_id = p.pas_id
```

(f) Changing data means an UPDATE statement which also contains a value condition and a join condition (U, J, C).

```
UPDATE booking
SET date = "26/09/17"
FROM passenger
WHERE passenger.pas_id = booking.pas_id
  AND passenger.name = "Pieter-Tjerk"
  AND booking.date = "19/09/17"
```

(g) This query requires a DISTINCT to get rid of the duplicates in the result. Furthermore, it is important to use brackets to make sure the logical condition is specified in the right way (C, J).

```
SELECT DISTINCT f.flightnumber, f.time
FROM flight f, airport a
WHERE a.city = "Vienna"
AND (f.from = a.code OR f.to = a.code)
```

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

(a) Explain what is meant with the technical term "concurrency" in the context of databases.
(b) Explain the difference between "projection" and "selection" (context: SQL querying)
(c) Explain what a "Cartesian product" is

□

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) "Concurrency" is the problem of multiple users or applications accessing and manipulating the same data at the same time.
(b) "Projection" means choosing for not all but a particular list of *attributes* or *columns* for a query result. "Selection" on the other hand means choosing for not all but a particular set of *rows* or *records*. Both are a form of selecting, but the one selects columns and the other selects rows.
(c) A cartesian product of two tables A and B is a table containing a row for each *combination* of records from A and B, where this row is composed of a *concatenation* of these two records.

3 *Database design (10 points)*

Figure 1 contains part of the datamodel belonging to the database of a company selling products. It maintains their customers and products as well as the purchases these customers made. The Customer class obviously represents all the company's customers (with their name, address, and number of the discount card), and the Product class its products (with their name and price). The class ShoppingVisit represents one visit to one of the shops (we store the city of the shop and the name of the manager). During such a visit, the customer can purchase several products; for each such purchase we also store the quantity (i.e., how many items of that product were bought). The ShoppingVisit class stores the date and time of the visit as well as the total amount of money of all purchases made during that visit.

- 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.

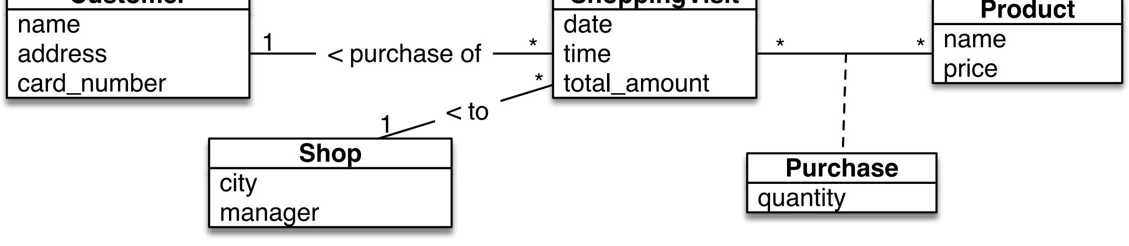

Figure 1: ER model of a purchasing database.

□

Answer to 3.

Primary keys in **bold**.

Customer: name, address, **card_number** *no foreign keys*
Shop: **shop_id**, city, manager *no foreign keys*
Product: **prod_id**, name, price *no foreign keys*
ShoppingVisit: **visit_id**, purchase_of, to_shop, date, time, total_amount *purchase_of is a foreign key referring to the card_number of a Customer.* *to_shop is a foreign key referring to the shop_id of the visited Shop.*
Purchase: **visit_id**, **prod_id**, quantity *visit_id is a foreign key referring to the id of the ShoppingVisit* *prod_id is a foreign key referring to the id of the Product that was purchased.*

Table	**Attributes / Description**
passenger	**pas_id**, name, address
	For every passenger, a unique number (pas_id; also primary key) and the name and address of the passenger.
flight	**flightnumber**, **date**, time, from, to
	For every flight, its flight number (e.g., "KL123"), the date and time of the flight, and the airport code from where the flight departs (from) and the airport code where the flight ends (to). It happens that every week, the same flight number is used for a flight on the same date and time in that week, for example, "KL123" could be the flight on every Monday 10:45. Therefore, the flight number is not globally unique (it re-occurs every week), hence the primary key is the combination of the flight number and the date together.
booking	**pas_id**, **flightnumber**, **date**, bookdate
	For every booking, the bookdate, the id of the passenger, and the flightnumber and date of the booked flight.
airport	**code**, city, country
	For every airport, its code and the city and country where this airport is located.

Figure 2: Table structure of an airline's "flight booking" database (Primary keys in **bold**). It is a simplification of the ER-model used as an example in the lectures.

passenger

pas_id	name	address
1	Maurice	...
2	Pieter-Tjerk	...
3	Arend	...

airport

code	city	country
AMS	Amsterdam	Netherlands
BRU	Brussels	Belgium
VIE	Vienna	Austria
NYJ	New York	USA
MST	Maastricht	Netherlands

booking

pas_id	flightnumber	date	bookdate
1	KL123	18/09/17	01/09/17
1	OS45	21/09/17	01/09/17
2	KL234	19/09/17	05/08/17
3	NW678	11/09/17	06/09/17

flight

flightnumber	date	time	from	to
KL123	11/09/17	10:45	AMS	VIE
KL123	18/09/17	10:45	AMS	VIE
KL123	25/09/17	10:45	AMS	VIE
OS45	21/09/17	9:30	VIE	AMS
OS45	28/09/17	9:30	VIE	AMS
KL234	19/09/17	17:15	AMS	BRU
KL234	26/09/17	17:15	AMS	BRU
NW678	11/09/17	13:05	AMS	NYJ
NW678	18/09/17	13:05	AMS	NYJ

Figure 3: Example data for the "flight booking" 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