CO526 Databases: Exercises
2018

In **family_history** database, there is a person table, where people are identified by their name, and always have their gender, date of birth (dob) and place of birth (born_in) recorded. In addition, each person may optionally have recorded the name of their father, and the name of their mother. If the person has died, then the date of death dod must be present. Note that only a fragment of the data held in the database is listed below.

person						
name	gender	dob	dod?	father?	mother?	born_in
Alice	F	1885-02-25	1969-12-05	null	null	Windsor
Andrew	M	1960-02-19	null	Philip	Elizabeth II	London
Andrew of Greece	M	1882-02-02	1944-12-03	George I of Greece	null	Athens
Anne (Princess)	F	1950-08-15	null	Philip	Elizabeth II	London
Charles	M	1948-11-14	null	Philip	Elizabeth II	London
Elizabeth II	F	1926-04-21	null	George VI	Elizabeth	London

$$\vdots$$

$$\text{person(father)} \overset{fk}{\Rightarrow} \text{person(name)}$$
$$\text{person(mother)} \overset{fk}{\Rightarrow} \text{person(name)}$$

Questions

In the following questions you can test for a value v being null using the predicate isNull(v), and v being not null using isNotNull(v). You may use subcripts on relation names to creat aliases of relations, such that person_a, person_b, *etc* are aliases for person.

1. Describe how you would enhance the database schema (with additonal tables, columns, primary keys or foreign keys) to allow the storage of which person is a monarch, and ensure that we record for just monarchs (i) the year of succession to the throne as succ_year, and (ii) the name of the country which they are monarch of as cname.

   The cleanest technique is to note that this implies a subset of persons in the form
   monarch(name,succ_year,country)
   $\text{monarch(name)} \overset{fk}{\Rightarrow} \text{person(name)}$

2. Write a query in each of the following languages that returns the scheme (name,born_in) containing the name and place of birth of all people known to have been born in the same place as their mother.

   (a) RA

   (b) Datalog

   (a) $\pi_{\text{person.name,person.born\_in}}\, \sigma_{\text{person.mother}=\text{person}_\text{m}.\text{name} \wedge \text{person.born\_in}=\text{person}_\text{m}.\text{born\_in}}(\text{person} \times \text{person}_\text{m})$

   (b) We will assume SQL type handling of NULLs in Datalog

   ```
   shared_maternal_birthplace(Name, BornIn) :-
           person(Name, _, _, Mother, BornIn),
           person(Mother, _, _, _, BornIn).
   ```

3. Write a query in each of the following languages that returns the scheme (name) containing names of all people known to be parents.

(a) RA

(b) Datalog

(a)

$\pi_{\text{mother}}\, \sigma_{\text{IsNotNull(mother)}}\, \text{person} \cup \pi_{\text{father}}\, \sigma_{\text{IsNotNull(father)}}\, \text{person}$

(b)

```
parent(Name) :-
        person(_, _, Name, _, _),
        isNotNull(Name).
parent(Name) :-
        person(_, _, _, Name, _),
        isNotNull(Name).
```

4. Write a query in each of the following languages that returns the scheme (name) containing the names of all men not known to be fathers.

   (a) RA

   (b) Datalog

(a)

$\pi_{\text{name}}\, \sigma_{\text{gender}=\text{'M'}}\, \text{person} - \pi_{\text{father}}\, \sigma_{\text{isNotNull(father)}}\, \text{person}$

(b) Using the general approach in the lectures:

```
isNotFather(Man) :-
        person(Man, 'M', _, _, _),
        ¬isFather(Man).
isFather(Man) :-
        person(_, _, Man, _, _).
```

In this case, since isFather contains only one predicate, one can simplify it to:

```
isNotFather(Man) :-
        person(Man, 'M', _, _, _),
        ¬person(_, _, Man, _, _).
```

5. Write a query in each of the following languages returning the scheme (name) listing those people that have had at least one child of each gender that appears in the database.

   (a) RA

   $\pi_{\text{father as name,gender}}\, \sigma_{\text{IsNotNull(father)}}\, \text{person} \div \pi_{\text{gender}}\, \text{person} \cup$
   $\pi_{\text{mother as name,gender}}\, \sigma_{\text{IsNotNull(mother)}}\, \text{person} \div \pi_{\text{gender}}\, \text{person}$

   Note that the above answer does make the assumption that each person can be just a father, or just a mother. If you want to capture the concept of a person sometimes being a mother, and sometimes being a father, you would need a union on the LHS of each division.

   (b) Datalog

```
all_genders(Name) :-
        parent(Name, _),
        ¬missing_gender(Name).
missing_gender(Name) :-
        parent(Name, _),
        person(_, Gender, _, _, _, _),
        ¬parent(Name, Gender).
parent(Name, Gender) :-
        person(_, Gender, _, _, Name, _, _),
        isNotNull(Name).
parent(Name, Gender) :-
        person(_, Gender, _, _, _, Name, _),
        isNotNull(Name).
```

6. Suppose the following RA query q has been executed:

$$\pi_{\text{person}_b.\text{name},\text{person}_a.\text{father}}\, \sigma_{\text{person}_a.\text{name}=\text{person}_b.\text{mother}\wedge\text{isNotNull}(\text{person}_a.\text{father})}(\text{person}_a \times \text{person}_b)$$

   (a) If the query has been executed at one point in time, since which Δ row have been inserted into person to give $\text{person}' = \text{person} \cup \Delta_p$, give an RA query that returns any additional answers to the original query. Your answer should work even if the foreign key definitions had not been given.

   (b) Now taking into account the foreign key definitions given, how could your answer be improved?

   (a) Note that the additional rows might be mothers, or children, or both. So

$$\pi_{\Delta_p.\text{name},\text{person.father}}\, \sigma_{\text{person.name}=\Delta_p.\text{mother}\wedge\text{isNotNull}(\text{person.father})}(\text{person} \times \Delta_p) \cup$$
$$\pi_{\text{person.name},\Delta_p.\text{father}}\, \sigma_{\Delta_p.\text{name}=\text{person.mother}\wedge\text{isNotNull}(\Delta_p.\text{father})}(\Delta_p \times \text{person}) \cup$$
$$\pi_{\Delta_{p_a}.\text{name},\Delta_{p_b}.\text{father}}\, \sigma_{\Delta_{p_b}.\text{name}=\Delta_{p_a}.\text{mother}\wedge\text{isNotNull}(\Delta_{p_b}.\text{father})}(\Delta_{p_b} \times \Delta_{p_a})$$

   (b) If one observes that the FK definitions mean that Δ_p cannot be adding a mother of an existing record in person, and hence we do not need to consider any existing rows of person in the RHS of the union. Thus the query simplifies to:

$$\pi_{\Delta_p.\text{name},\text{person.father}}\, \sigma_{\text{person.name}=\Delta_p.\text{mother}\wedge\text{isNotNull}(\text{person.father})}(\text{person} \times \Delta_p) \cup$$
$$\pi_{\Delta_{p_a}.\text{name},\Delta_{p_b}.\text{father}}\, \sigma_{\Delta_{p_b}.\text{name}=\Delta_{p_a}.\text{mother}\wedge\text{isNotNull}(\Delta_{p_b}.\text{father})}(\Delta_{p_b} \times \Delta_{p_a})$$

7. Write an SQL query that returns the scheme (name,born_in) ordered by name containing the name and place of birth of all people known to have been born in the same place as their mother.

```
SELECT  person.name,
        person.born_in
FROM    person
        JOIN  person  AS  mother
        ON    person.mother=mother.name AND person.born_in=mother.born_in
```

8. Write an SQL query that returns the scheme (name) ordered by name containing the name of all people known to be parents.

```
SELECT mother AS name
FROM   person AS mother
WHERE  mother IS NOT NULL
UNION
SELECT father AS name
FROM   person AS father
WHERE  father IS NOT NULL
ORDER BY name
```

9. Write an SQL query that returns the scheme (name) ordered by name that lists parents for whom all known children are of the same sex.

```
SELECT mother AS name,
       gender
FROM   person AS mother
WHERE  mother IS NOT NULL
AND    gender=ALL (SELECT gender
                   FROM   person
                   WHERE  person.mother=mother.mother)
UNION
SELECT father AS name,
       gender
FROM   person AS father
WHERE  father IS NOT NULL
AND    gender=ALL (SELECT gender
                   FROM   person
                   WHERE  person.father=father.father)
ORDER BY name;
```

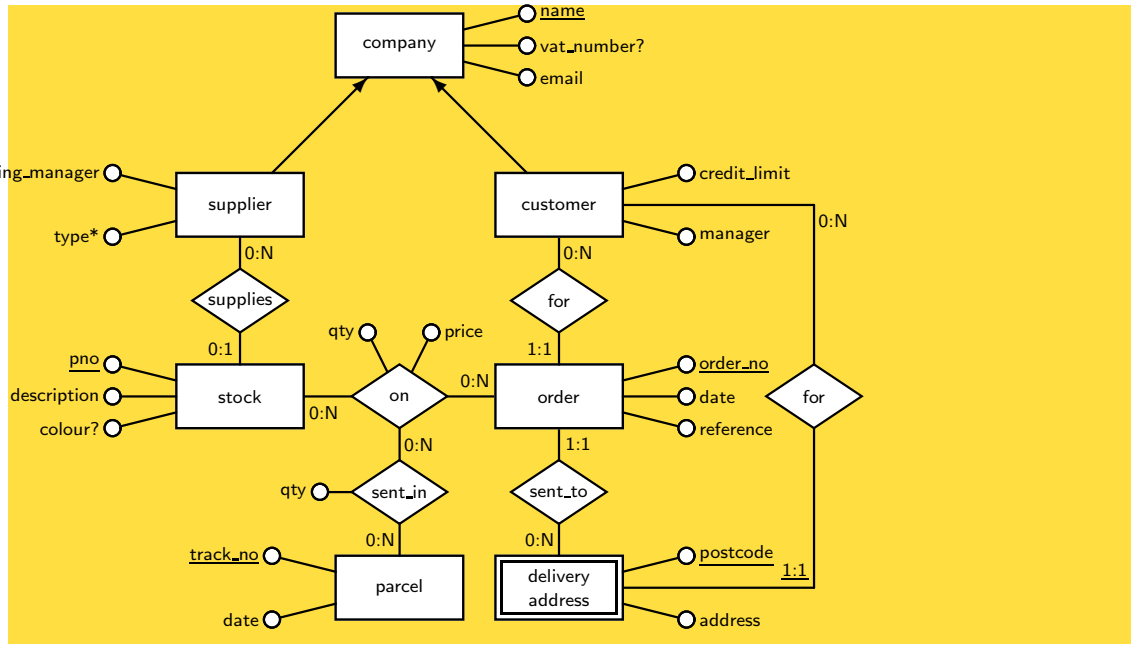
10. Suppose you have to design a new database to hold the following information about the companies that do business with ACME Computing Ltd. These companies may be customers, suppliers, or both. For all companies we record their name and contact email address, and for companies that are VAT registered, we must record their VAT number.

    For suppliers, we record the purchasing manager that deals with the supplier, and record a number of types of product that the supplier can supply. We associate to each supplier all the stock items currently being supplied. It is company policy that each stock item may come from only one supplier, and some stock items are manufactured by ACME Computing Ltd itself, and therefore have no supplier. Each stock item has a part number and description, and some stock items have their colour recorded.

    For customers, we record the sales manager that deals with the customer, and a credit limit. We also record all orders made by the customer. Each order is on a particular date, is given an order number, and has a reference number given by the customer. An order may have any number of stock items, with the quantity and price for each stock item recorded.

    When orders are sent, we record the tracking number of each parcel the order was sent in, and the date on which the parcel was sent. We also wish to record how much of each stock item was put in each parcel, in case the parcel gets lost and an insurance claim must be made. We record the delivery address for each order, from a record of delivery addresses, for which we record the customer name, postcode and address. We identify an address record by the combination of the customer name and postcode.

    (a) Design an ER$^{\mathcal{ADHKLMNOSVW}}$ schema to represent this new database.



(b) Map the ER schema you designed in (i) into a relational schema.

company(name,vat_number?,email)
supplier(name,purchasing_manager)
supplier_type(name,type)
customer(name,credit_limit,manager)
stock(pno,description,colour?,name?)
order(ono,date,reference,cname,dname,postcode)
parcel(track_no,date)
on(pno,ono,qty,price)
sent_in(pno,ono,track_no,qty)
delivery_address(name,postcode,address)
customer(name) $\stackrel{fk}{\Rightarrow}$ company(name)
supplier(name) $\stackrel{fk}{\Rightarrow}$ company(name)
supplier_type(name) $\stackrel{fk}{\Rightarrow}$ supplier(name)
stock(name) $\stackrel{fk}{\Rightarrow}$ supplier(name)
on(pno) $\stackrel{fk}{\Rightarrow}$ stock(pno)
on(ono) $\stackrel{fk}{\Rightarrow}$ order(ono)
sent_in(pno,ono) $\stackrel{fk}{\Rightarrow}$ on(pno,ono)
sent_in(track_no) $\stackrel{fk}{\Rightarrow}$ parcel(track_no)
order(dname,postcode) $\stackrel{fk}{\Rightarrow}$ delivery_address(name,postcode)
order(cname) $\stackrel{fk}{\Rightarrow}$ customer(name)
delivery_address(name) $\stackrel{fk}{\Rightarrow}$ customer(name)

11. Suppose that a relation $R(A, B, C, D, E, F, G, H)$ has the functional dependency set $S = \{A \to BE, AC \to G, AFG \to E, B \to ACG, CF \to D, D \to G, DEG \to FCD, G \to H\}$

(a) Compute a minimum cover S_c of S.

Rewrite with single attribute on RHS
Since $D \to G$
$DEG \to FCD \Rightarrow DE \to FCD$
Since $D \to D$
$DE \to FCD \Rightarrow DE \to CF$
Since $A \to B, B \to CG$
$AC \to G \Rightarrow \emptyset$
Since $A \to BE$
$AFG \to E \Rightarrow \emptyset$
Therefore $S_c = \{A \to BE, B \to ACG, CF \to D, D \to G, DE \to CF, G \to H\}$

(b) Identify and justify all the candidate keys of R.

We find $A^+ = A, B, C, E, G, H$
Thus A is not a candiate key, and we should consider adding D or F to A
adding D (ie AD^+) will cover all attributes
Therefore, AF^+ covers all attributes (because of $A \to C, CF \to D$)
Therefore BD^+ and BF^+ cover all attributes (because of $B \to A$)
Since nothing determines A or B except each other, all candidate keys must include A or B, and therefore we need not consider keys based on any other attributes.
So the candidate keys are AD, BD, AF, BF

(c) Decompose the relation R into 3NF.

Non-prime attributes are $CEGH$
Since $G \to H$, and G is not a key, remove H from R to get
$R_1(G, H)$
Since $D \to G$, and D is not a key, remove G from R to get
$R_2(D, G)$
Since $A \to CE$, and A is not a key, remove CE from R to get
$R_3(A, C, E)$
This leaves
$R_4(A, B, D, F)$
R_1, R_2, R_3, R_4 are in 3NF.
However, the FDs $B \to G, CF \to D, D \to G, DE \to CF$ are not preserved.
$B \to G$ can be preserved by adding G to R_3 to get $R_5(A, C, E, G)$ (since $A \to B$ and $B \to A$, it is as good to preserve $A \to G$).
$DE \to CF$ can be preserved by adding $R_6(C, D, E, F)$.
Now R_1, R_2, R_4, R_5, R_6 are in 3NF and preserve FDs.
Note that it is wrong to overnormalise the relations, and you should use the algorithm to perform 3NF rather than just use all FDs to breakup the original relation.

(d) Decompose the relation R into BCNF.

Since $A \to B$ and A is not a key, R_4 is not in BCNF, so remove B from R_4 to get
$R_7(A, B)$ and $R_8(A, D, F)$
Note that R_7 can combine into R_5 (since $A \to B$ and $B \to A$) to get $R_9(A, B, C, E, G)$
Since $CF \to D$ and CF is not a key of R_6, we no longer can have this relation to preserve FDs. However we can add $R_{10}(C, D, F)$ to preserve $CF \to D$. To preserve $DE \to CF$ we must add $R_{11}(D, E, C)$ and $R_{12}(D, E, F)$. Now BCNF is $R_1, R_2, R_8, R_9, R_{10}, R_{11}, R_{12}$.

12. The following histories describe the sequence of operations performed respectively by four transactions T_1–T_4.

$H_1 = r_1[c_{CZ}], r_1[c_R], r_1[c_B], r_1[c_{GB}], c_1$
$H_2 = r_2[c_B], r_2[c_R], r_2[c_{CZ}], w_2[c_{CZ}], c_2$

$H_3 = r_3[c_B], w_3[c_B], r_3[c_{CZ}], w_3[c_{CZ}], c_3$
$H_4 = w_4[c_R], w_4[c_B], w_4[c_{GB}], c_4$

(a) Briefly explain if the following concurrent execution is serialisable and recoverable. If non-serialisable, explain what anomaly occurs.

$H_a = r_1[c_{CZ}], r_1[c_R], r_3[c_B], w_3[c_B], r_3[c_{CZ}], w_3[c_{CZ}],$
$r_1[c_B], r_1[c_{GB}], c_1, c_3,$

Conflicts $r_1[c_{CZ}] \rightarrow w_3[c_{CZ}]$ and $w_3[c_B] \rightarrow r_1[c_B]$ form a cycle, therefore not serialisable.
Anomaly is inconsistent analysis by T_1
No dirty $r_1[c_B]$ is committed whilst still dirty, so non-recoverable

(b) Briefly explain if the following concurrent execution is serialisable and recoverable. If non-serialisable, explain what anomaly occurs.

$H_b = r_3[c_B], w_4[c_R], w_4[c_B], w_4[c_{GB}], w_3[c_B],$
$r_3[c_{CZ}], w_3[c_{CZ}], c_4, c_3$

Conflicts $r_3[c_B] \rightarrow w_4[c_B]$ and $w_4[c_B] \rightarrow w_3[c_B]$ form a cycle, therefore not serialisable.
No dirty reads so recoverable and ACA. However, dirty write $w_3[c_B]$ means not ST.

(c) Briefly explain if the following concurrent execution is serialisable and recoverable. If non-serialisable, explain what anomaly occurs.

$H_c = r_1[c_{CZ}], w_4[c_R], r_1[c_R], w_4[c_B], r_1[c_B], w_4[c_{GB}], c_4, r_1[c_{GB}], c_1$

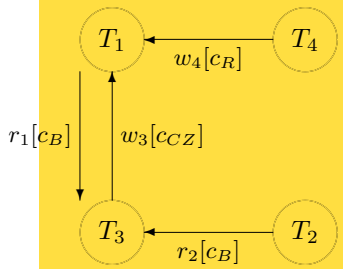
Conflicts $w_4[c_R] \rightarrow r_1[c_R]$, $w_4[c_B] \rightarrow r_1[c_B]$, and $w_4[c_{GB}] \rightarrow r_1[c_{GB}]$, do not form a cycle, and therefore the history is serialisable.
Two dirty reads, and therefore not ACA (or ST), but since at time of c_1 reads are no longer dirty, the history is recoverable.

(d) Briefly explain which (if any) pair of the transactions taken from T_1–T_4 will be serialisable for all concurrent executions of the pair, and also briefly explain which (if any) pair of the transactions taken from T_1–T_4 will be recoverable for all concurrent executions of the pair.

Since T_1 and T_2 share only one conflict (between $r_1[c_{CZ}]$ and $w_2[c_{CZ}]$), and therefore can sequence those two operations in any order, it means all possible concurrent executions of T_1, T_2 are serialisable.
Since all pairs of histories have at least one read-write conflict, there will always be a possible non-recoverable concurrent history where the read follows the write in the conflicting pair, and the transaction containing the read commits first.

(e) Give a concurrent execution of the four transactions, which produces a deadlock involving all four transactions, and draw a waits-for graph for the deadlock state.

$r_1[c_{CZ}], r_1[c_R], r_3[c_B], w_3[c_B], r_3[c_{CZ}], \textit{deadlock}$



13. The table below lists the contents of a database log, which keeps only UNDO records on a table country.

UNDO	$w_1[\mathsf{c_R}, \mathsf{population} = 148,179,000]$
UNDO	$w_4[\mathsf{c_{TR}}, \mathsf{population} = 62,481,123]$
UNDO	$w_2[\mathsf{c_{CH}}, \mathsf{population} = 7,392,444]$
UNDO	$w_2[\mathsf{c_{GB}}, \mathsf{population} = 58,543,111]$
UNDO	$w_3[\mathsf{c_{CH}}, \mathsf{population} = 7,312,222]$
UNDO	$w_5[\mathsf{c_{CH}}, \mathsf{population} = 7,210,000]$
LOG	c_4
UNDO	$w_3[\mathsf{c_B}, \mathsf{population} = 11,020,000]$
LOG	c_3

country

name	code	capital	area	population
Czech Republic	CZ	Prague	78,703	10,321,120
Switzerland	CH	Bern	41,290	7,207,060
Russia	R	Moscow	17,075,200	148,178,487
Belgium	B	Brussels	30,510	10,170,241
Turkey	TR	Ankara	780,580	62,484,478
United Kingdom	GB	London	244,820	58,489,975
Egypt	ET	Cairo	1,001,450	63,575,107

(a) If the country table has the contents illustrated above, describe the actions performed by the recovery procedure using the above log, and what population figures will be left after recovery.

Working back from the end of the log, we must perform the oldest UNDO action of each object from transactions which did not commit, where that UNDO is after the last UNDO action for that object from a committed transaction.

UNDO $w_1[\mathsf{c_R}, \mathsf{population} = 148,179,000]$
UNDO $w_2[\mathsf{c_{GB}}, \mathsf{population} = 58,543,111]$
UNDO $w_5[\mathsf{c_{CH}}, \mathsf{population} = 7,210,000]$

Hence all rows of country will be the same, except for R the value will be 148,179,000, GB the value will be 58,543,111, CH the value will be 7,210,000.

(b) If an additional LOG record were added for a_5 to record the completion of aborting T_5, would your answer to (a) change, and if so, how does it change?

You do not need to perform UNDO records associated with T_5, since the purpose of recording of the abort is to indicate that the UNDO for T_5 has been completed.

(c) Considering the time just after when c_4 occurs, describe and justify which updates from the above log must have been written to disc, which might have been written to disc, and which must not have been written to disc.

With UNDO logging, everything must be written to disc from a committed transaction, so all w_4 operations must be on disc. Since T_3 has yet to commit, this rule does not apply the the preceeeding w_3.

Any transactions might have been flushed out, so all the write operations preceding c_4 from T_2, T_3, T_4 and T_5 might be written to disc.

In UNDO only logging, there are no restrictions on which items must not have been flushed out.