Exercise 4: Theoretical Problems

Student Number: 100269314

3. Triggers

(a) Write CREATE TABLE commands for the relations. Add two checks: insuranceValue cannot be negative, and year must be between 1900–2100. None of the attributes can be NULL. Remember to define keys as well.

Solution.

```
CREATE TABLE Property(
    id INTEGER PRIMARY KEY,
    address VARCHAR(100) NOT NULL,
    city VARCHAR(100) NOT NULL,
    year INTEGER CHECK (year>=1900 AND year<=2100) NOT NULL,
    owner CHAR(11) REFERENCES Owner(ssNo) NOT NULL,
    insuranceValue FLOAT CHECK (insuranceValue>=0.0) NOT NULL
);

CREATE TABLE Owner(
    ssNo CHAR(11) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(100) NOT NULL
);
```

(b) Insert at least 10 rows for the tables, so triggers can be tested in part (f).
Solution.

	ssNo	name	address
1	12345678911	John	address 1
2	12345678912	John2	address 2
3	12345678913	John3	address 3
4	12345678914	John4	address 4
5	12345678915	John5	address 5
6	12345678916	John6	address 6
7	12345678917	John7	address 7
8	12345678918	John8	address 8
9	12345678919	John9	address 9
10	12345678910	John10	address 0

	id	address	city	year	owner	insuranceValue
1	1	address 1	CITY 1	2008	12345678910	19.2
2	2	address 2	CITY 1	2038	12345678911	190.2
3	3	address 3	CITY 1	1908	12345678911	100
4	4	address 4	CITY 2	1998	12345678910	200
5	5	address 5	CITY 1	2028	12345678915	300
6	6	address 6	CITY 1	2000	12345678913	400
7	7	address 7	CITY 4	2007	12345678910	500
8	8	address 8	CITY 1	2003	12345678912	77.77
9	9	address 9	CITY 5	2028	12345678915	123.45
10	10	address 10	CITY 6	2019	12345678910	19.3

(c) When inserting a row into table Property, check that the city is written with all caps and fix if needed.

Solution.

```
CREATE TRIGGER CityAllCaps
        BEFORE INSERT
            ON Property
      FOR EACH ROW
          WHEN (NEW.city! = upper(NEW.city) )
BEGIN
```

```
    INSERT INTO Property VALUES (
                            NEW.id,
                            NEW.address,
                            upper(NEW.city),
                            NEW.year,
                            NEW.owner,
                            NEW.insuranceValue
                        );
    SELECT RAISE(IGNORE);
END;

--test 3c (part f - see images)
-- doesn't trigger CityAllCaps
INSERT INTO Property(id, address,city,year,owner,insuranceValue)
VALUES
(999, "address 1", "CITY 1", 2008, "12345678910", 19.2);
-- triggers CityAllCaps
INSERT INTO Property(id, address,city,year,owner,insuranceValue)
VALUES
(1000, "address 2", "ciTy 1", 2018, "12345678910", 19.1);
```

(d) When updating a row in table Property, check that the new value of insuranceValue is at least 75% of the previous value. If it is smaller, set the new value of insuranceValue to 75% of the previous value.

Solution.

```
CREATE TRIGGER insuranceValueFloor
        AFTER UPDATE
           ON Property
     FOR EACH ROW
         WHEN (NEW.insuranceValue < OLD.insuranceValue * 0.75)
BEGIN
    UPDATE Property
       SET insuranceValue = OLD.insuranceValue * 0.75
     WHERE id = NEW.id;
END;
--test 3d (part f - see image)
-- doesn't trigger insuranceValueFloor
UPDATE Property SET insuranceValue = 70 WHERE id = 3;
-- triggers insuranceValueFloor
UPDATE Property SET insuranceValue = 175 WHERE id = 4;
```

(e) When removing a row from table Owner, check that removed person is not owner of any property in the database. If person still owns something, prevent the removal.

Solution.

```
CREATE TRIGGER ownerDeleteProtection
       BEFORE DELETE
```

```
            ON Owner
      FOR EACH ROW
          WHEN EXISTS(
    SELECT 1
      FROM Owner, Property
     WHERE Property.owner=OLD.ssNo
)
BEGIN
    SELECT RAISE(IGNORE);
END;
-- test 3e
-- doesn't trigger ownerDeleteProtection
DELETE FROM Owner WHERE ssNo=="12345678918";
-- triggers ownerDeleteProtection
DELETE FROM Owner WHERE ssNo=="12345678910";
```

(f) After that, write two commands for each trigger (b), (c), and (d). The other should launch the trigger and the other should not. List the contents of the tables Property and Owner both before and after running all those statements (intermediate results are not required). For example, screenshot from SQLiteStudio is enough.

Solution.

3c - Before and After

	id	address	city	year	owner	insuranceValue
1	1	address 1	CITY 1	2008	12345678910	19.2
2	2	address 2	CITY 1	2038	12345678911	190.2
3	3	address 3	CITY 1	1908	12345678911	100
4	4	address 4	CITY 2	1998	12345678910	200
5	5	address 5	CITY 1	2028	12345678915	300
6	6	address 6	CITY 1	2000	12345678913	400
7	7	address 7	CITY 4	2007	12345678910	500
8	8	address 8	CITY 1	2003	12345678912	77.77
9	9	address 9	CITY 5	2028	12345678915	123.45
10	10	address 10	CITY 6	2019	12345678910	19.3

	id	address	city	year	owner	insuranceValue
1	1	address 1	CITY 1	2008	12345678910	19.2
2	2	address 2	CITY 1	2038	12345678911	190.2
3	3	address 3	CITY 1	1908	12345678911	100
4	4	address 4	CITY 2	1998	12345678910	200
5	5	address 5	CITY 1	2028	12345678915	300
6	6	address 6	CITY 1	2000	12345678913	400
7	7	address 7	CITY 4	2007	12345678910	500
8	8	address 8	CITY 1	2003	12345678912	77.77
9	9	address 9	CITY 5	2028	12345678915	123.45
10	10	address 10	CITY 6	2019	12345678910	19.3
11	999	address 1	CITY 1	2008	12345678910	19.2
12	1000	address 2	CITY 1	2018	12345678910	19.1

3d - Before and After

	id	address	city	year	owner	insuranceValue
1	1	address 1	CITY 1	2008	12345678910	19.2
2	2	address 2	CITY 1	2038	12345678911	190.2
3	3	address 3	CITY 1	1908	12345678911	100
4	4	address 4	CITY 2	1998	12345678910	200
5	5	address 5	CITY 1	2028	12345678915	300
6	6	address 6	CITY 1	2000	12345678913	400
7	7	address 7	CITY 4	2007	12345678910	500
8	8	address 8	CITY 1	2003	12345678912	77.77
9	9	address 9	CITY 5	2028	12345678915	123.45
10	10	address 10	CITY 6	2019	12345678910	19.3
11	999	address 1	CITY 1	2008	12345678910	19.2
12	1000	address 2	CITY 1	2018	12345678910	19.1

	id	address	city	year	owner	insuranceValue
1	1	address 1	CITY 1	2008	12345678910	19.2
2	2	address 2	CITY 1	2038	12345678911	190.2
3	3	address 3	CITY 1	1908	12345678911	75
4	4	address 4	CITY 2	1998	12345678910	175
5	5	address 5	CITY 1	2028	12345678915	300
6	6	address 6	CITY 1	2000	12345678913	400
7	7	address 7	CITY 4	2007	12345678910	500
8	8	address 8	CITY 1	2003	12345678912	77.77
9	9	address 9	CITY 5	2028	12345678915	123.45
10	10	address 10	CITY 6	2019	12345678910	19.3
11	999	address 1	CITY 1	2008	12345678910	19.2
12	1000	address 2	CITY 1	2018	12345678910	19.1

3e - Before and After

	ssNo	name	address
1	12345678911	John	address 1
2	12345678912	John2	address 2
3	12345678913	John3	address 3
4	12345678914	John4	address 4
5	12345678915	John5	address 5
6	12345678916	John6	address 6
7	12345678917	John7	address 7
8	12345678918	John8	address 8
9	12345678919	John9	address 9
10	12345678910	John10	address 0

	ssNo	name	address
1	12345678911	John	address 1
2	12345678912	John2	address 2
3	12345678913	John3	address 3
4	12345678914	John4	address 4
5	12345678915	John5	address 5
6	12345678916	John6	address 6
7	12345678917	John7	address 7
8	12345678919	John9	address 9
9	12345678910	John10	address 0

4. **Indices** Consider the relation schema OrderContent(orderID, product, amount) from the online store database introduced in the Exercise Round 1. Let's assume the relation occupies 120 pages of space. On average, each order includes 5 products and each product belongs to 30 orders. There is no clustering of any attributes.

Two kinds of queries are frequent for the table: searching for certain order ID (type Q1, fraction p1) and searching for orders with certain products (type Q2, fraction p2).

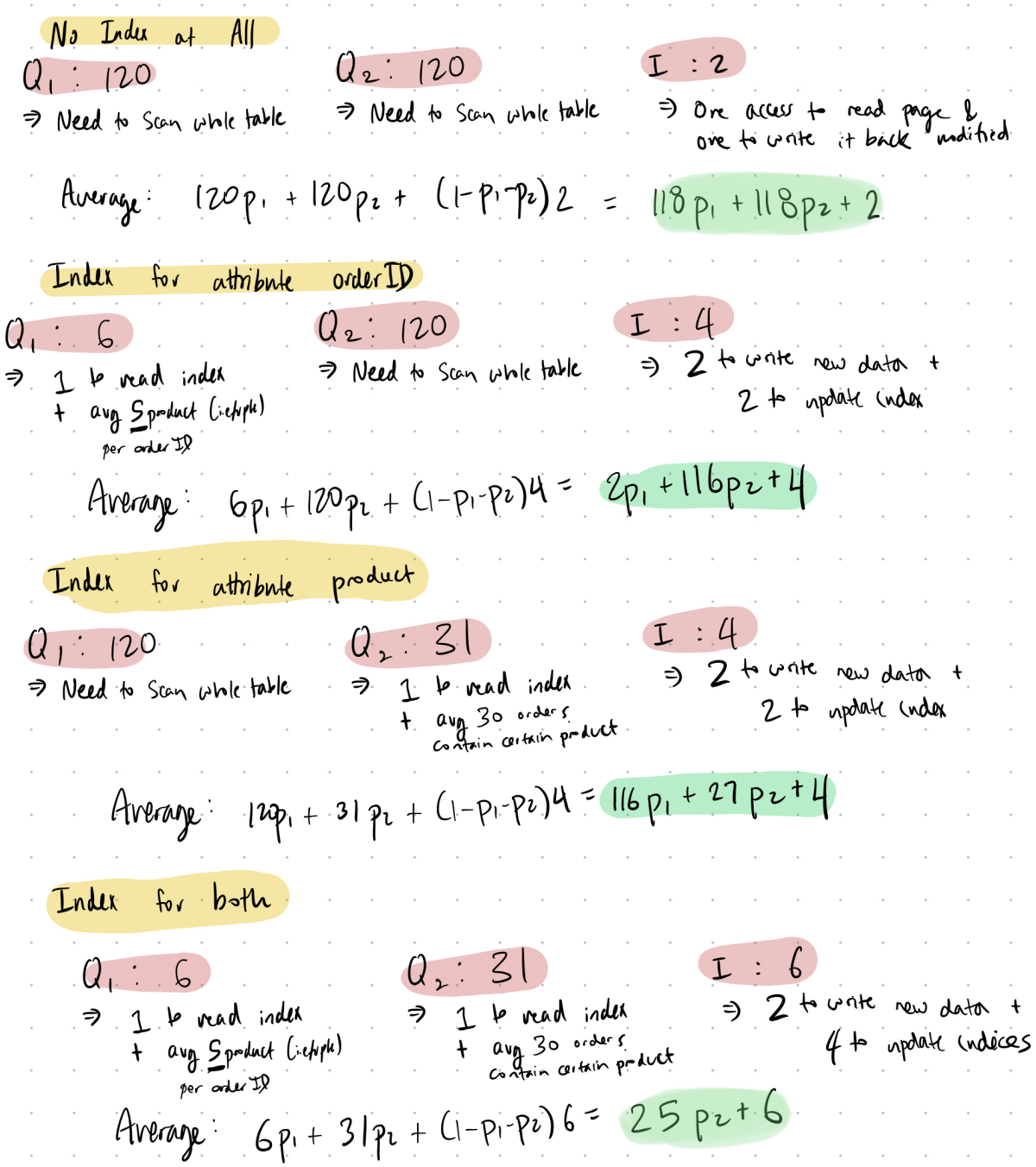
Insertions to the table take the fraction 1 - p1 - p2 of all operations on the table (type I).

Give formulas in terms of p1 and p2 to measure the cost of queries Q1 and Q2 and insertion I under the following four combinations (similarly to the exercise sessions).

- No indices at all

- Index for the attribute orderID
- Index for the attribute product
- Index for both attributes (orderID and product)

Solution.



5. **Transactions**

Solution.

a) i) '123' ⇒ stock = 65

Is possible : one such scenario (rep. as ordering of transactions)...

$T_2 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4$ — no impact!

T_2: value from 50 → 45

T_1: increases from 45 → 65 //

□

ii) '123' ⇒ stock < 50

Is possible : one such scenario (rep. as ordering of transactions)...

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$

T_1: doesn't increase stock as stock ≮ 50 (i.e. stock = 50)

T_2: decrease from 50 → 45 //

$T_3 \rightarrow T_4$: no impact

⇒ 45 < 50

□

iii) table products contains only one tuple

Impossible : due to all transactions $T_1 \ldots T_4$ uphold ACID principles, thanks to the atomicity of each transaction, T_4 will never run while T_3 is running (which is the only opportunity for deletion of tuples). Hence, the minimum # of tuples that can be in the table after the Transactions is 3 (w/ only mouse tuple removed)

□

iv) can order 'ABC shampoo' but not 'mouse'

Is possible : one such scenario (rep. as ordering of transactions)...

$T_2 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4$ — no impact

T_2: as 'ABC shampoo' 's stock ≯ 5 ⇒ can order

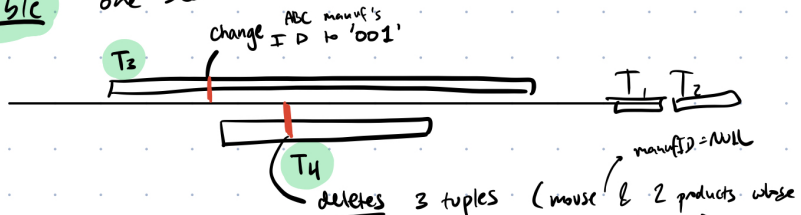
as 'mouse' 's stock < 5 ⇒ cannot order

└ i.e. 3

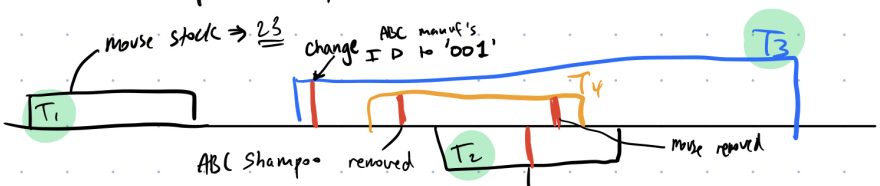
□

b) i) table products contains only one tuple w/o atomicity

Possible one scenario



ii) Possible ; one such scenario:



ABC shampoo no longer in product table ⇒ cannot
& mouse stock ≥ 5 ⇒ can order
⇒ mouse ordered & ABC not ordered

(Note: This depends on whether the deletion of T_3 is an atomic substep or also non-atomic. If it is non-atomic it is possible as indicated. BUT if the deletion occurs atomically, then it is not possible as mouse will always be removed & shampoo is orderable any time it it is not removed as a whole)

iii) Yes possible. one scenario is shown in 5a(ii)