Intro to Databases
EECS 116

Assignment 5
SQL Constraints

Aaron Zhong - 67737879 - alzhong@uci.edu
Tina Li - 92928656 - tinal7@uci.edu
Andy Le - 70829342 - andyl8@uci.edu

ICS Department
Donald Bren School of Information and Computer Science
University of California, Irvine

February 26, 2016

Contents

1 Questions

2 Answers

2.1 Problem 1.1

2.1.1 Part A

```
SELECT AVG(rating), SUM(rating), COUNT(rating), COUNT(*)
 FROM seller;
```

AVG(rating)	SUM(rating)	COUNT(rating)	COUNT(*)
5.3333	16	3	4

2.1.2 Part B

```
   SELECT (SUM(rating) / COUNT(rating)),(SUM(rating) / COUNT(*)), AVG(rating)
FROM seller;
```

(SUM(rating) / COUNT(rating))	(SUM(rating) / COUNT(*))	AVG(rating)
5.3333	4.0000	5.3333

If we divide the sum computed above by the count of the ratings, we would get exactly the same result as if we used AVG(). If we divided the sum computed above by the count of sellers, then we would have a different average than if we used AVG(). This is because COUNT(rating) only counts ratings which are not null, while COUNT(*) counts the number of tuples.

2.1.3 Part C

Left outer join of S1 and S2, join condition: S1.sid = S2.sid

```
SELECT *
 FROM S1 LEFT OUTER JOIN S2
    ON s1.sid=s2.sid;
```

sid	sname	rating	age	sid	sname	rating	age
18	jones	3	30	NULL	NULL	NULL	NULL
41	jonah	6	56	NULL	NULL	NULL	NULL

Right outer join of S1 and S2, join condition: S1.sid = S2.sid

```
SELECT *
 FROM S1 RIGHT OUTER JOIN S2
    ON s1.sid=s2.sid;
```

sid	sname	rating	age	sid	sname	rating	age
NULL	NULL	NULL	NULL	22	ahab	7	44
NULL	NULL	NULL	NULL	63	moby	NULL	15

Full outer join of S with itself (seller)

```
SELECT *
    FROM S LEFT OUTER JOIN Seller
    ON s.sid = seller.sid
UNION
SELECT *
 FROM S RIGHT OUTER JOIN Seller
```

sid	sname	rating	age	sid	sname	rating	age
18	jones	3	30	18	jones	3	30
41	jonah	6	56	41	jonah	6	56
22	ahab	7	44	22	ahab	7	44
63	moby	NULL	15	63	moby	NULL	15

2.2 Problem 1.2

2.2.1 Part A

```
CREATE TABLE Emp(
 eid int,
    ename varchar(100),
    age int,
 salary real,
    CHECK (salary < 10000)
    );
```

2.2.2 Part B

MySQL Version

```
CREATE TABLE Dept(
 did int,
    budget real,
    managerid int
);

ALTER TABLE dept
ADD CHECK (Dept.managerid IN (SELECT Emp.eid FROM emp, dept
    WHERE emp.age > 30 AND emp.eid = dept.managerid));
```

SQL (in class version)

```
CREATE TABLE Dept(
 did int,
    budget real,
    managerid int,
    CHECK (Dept.managerid IN (SELECT Emp.eid FROM emp, dept
        WHERE emp.age > 30 AND emp.eid = dept.managerid))
 );
```

2.2.3 Part C

```
CREATE ASSERTION OldManager CHECK
(NOT EXISTS
 (SELECT Emp.eid
    FROM emp, dept
    WHERE emp.age < 30 AND emp.eid = dept.managerid));
```

2.2.4 Part D

Assertion is better since it is not restricted a single relation that a check would normally validate. Compared to a check, assertions detect changes on any/all relations mentioned in the assert and is always guaranteed to hold.

2.2.5 Part E

SQL (in class version)

```
CREATE TRIGGER NoLowerAge
AFTER UPDATE ON emp
REFERENCING
 OLD AS OldTuple,
    NEW AS NewTuple
WHEN (OldTuple.age > NewTuple.age)
 UPDATE emp
    SET age = OldTuple.age
    WHERE name = NewTuple.name
FOR EACH ROW;
```

Version which works with MySQL

```
DELIMITER //

DROP TRIGGER IF EXISTS NoLowerAge//
CREATE TRIGGER NoLowerAge
BEFORE UPDATE ON emp
FOR EACH ROW
 BEGIN
```

```
IF NEW.age < OLD.age THEN
SET NEW.age = OLD.age;
END IF;
END;//

DELIMITER ;
```

2.3 Problem 1.3

2.3.1 Part A

dno	location
222	LA
333	SF
NULL	NULL

2.3.2 Part B

dno	location
333	SF
444	LA
NULL	NULL