**Solution to CS542 Homework 4**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Maximum:** 100pts  
**Note:** This homework is to be done by each student individually.   
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

You are given a database schema consisting of three relations, whose schemas are given below. You are expected to develop your own test data sets and to perform thorough testing on these data sets using Oracle. Also test your above SQL statements out on the sample data that we will provide.

Product(model, manufacturer, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

**Problem 1. Constraints Specification in SQL [25 pts]**

Assume no primary keys, nor foreign key constraints have been defined on this schema yet. Write SQL DDL to add the constraints below, whenever possible. Do not use triggers. Demonstrate that your solution indeed works or if not possible with DDL statements, say so and explain why not.

1. Write SQL DDL to add an additional constraint to the Product table. The constraint must enforce that the type of the product must either be ’PC’ or ’Laptop’.

ALTER TABLE Product

ADD CONSTRAINT validType

CHECK (type in (‘PC’, ’Laptop’));

1. Write SQL DDL to add an additional constraint that the price of any Laptop must be at least 500.

ALTER TABLE Laptop

ADD CONSTRAINT highPrice

CHECK(price>=500);

1. Add the constraint that a laptop with a larger model number must also have a higher price than one with a lower model number.

If the DBMS were to support complex CHECK constraints, then this constraint could be specified as follows:

ALTER TABLE Laptop

ADD CONSTRAINT bothHigh

CHECK (NOT EXIESTS (SELECT \*

FROM Laptop L1, Laptop L2

WHERE (L1.model>L2.model) AND (L1.price<=L2.price)));

Or we can write constraint like this:

ALTER TABLE Laptop

ADD CONSTRAINT bothHigh

CHECK (price>(select max(price)

FROM Laptop

WHERE Laptop.model<model));

However, Oracle does not support nested Check constraints. Hence, such a constraint could only be specified using a Trigger specification.

1. Add the constraint that any PC and Laptop corresponds to a model number that also exists in the Product table.

For this, you need to add foreign keys from the PC and Laptop tables to the Product table, respectively.

ALTER TABLE PC

ADD CONSTRAINT FK\_PC\_Product

FOREIGN KEY (model)

REFERENCES Product(model);

ALTER TABLE Laptop

ADD CONSTRAINT FK\_Laptop\_Product

FOREIGN KEY (model)

REFERENCES Product(model);

1. Add the constraint that in our product database table we only maintain products from at most 5 different manufacturers (so to assure that the quality we offer is high).

ALTER TABLE Product

ADD CONSTRAINT fiveProduct

CHECK ((SELECT COUNT(DISTINCT manufacturer)

FROM Product)<=5);

However, Oracle does not support nested Check constraints.

**Problem 2. Trigger Specification in SQL [50 pts]**

For below, consider first if triggers are able to express the desired semantics. Then determine how many triggers you need, what events you need to monitor, and lastly what actions are the most meaningful to take for each of the examples below. Test out your triggers, once designed.

1. Write one or more triggers to enforce overlap constraints, namely, to specify that when inserting a new laptop, the model number should not also appear in the PC table, and vice versa.

CREATE OR REPLACE TRIGGER PCNotInLaptop

BEFORE INSERT ON PC

FOR EACH ROW

DECLARE currCount int;

BEGIN

SELECT count(model) INTO currCount from Laptop

WHERE model = :new.model;

IF (currCount>0) THEN

RAISE\_APPLICATION\_ERROR(-20004,'Model conflicts.');

END IF;

END;

.

RUN;

**A similar constraint must also be specified for** **the Laptop table**. **You also need similar constraints on updates to assure that the updates of the ‘new’ model value are not equal to an already existing value.**

CREATE OR REPLACE TRIGGER LaptopNotInPC

BEFORE INSERT ON Laptop

FOR EACH ROW

DECLARE currCount INT;

BEGIN

SELECT COUNT(model) INTO currCount FROM PC

WHERE model = :new.model;

IF (currCount > 0) THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Model conflicts.');

END IF;

END;

.

RUN;

CREATE OR REPLACE TRIGGER PCUpdateCheck

BEFORE UPDATE ON PC

FOR EACH ROW

DECLARE currCount INT;

BEGIN

SELECT COUNT(model) INTO currCount FROM Laptop

WHERE model = :new.model;

IF (currCount > 0) THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Model conflicts during update.');

END IF;

END;

.

RUN;

CREATE OR REPLACE TRIGGER LaptopUpdateCheck

BEFORE UPDATE ON Laptop

FOR EACH ROW

DECLARE currCount INT;

BEGIN

SELECT COUNT(model) INTO currCount FROM PC

WHERE model = :new.model;

IF (currCount > 0) THEN

RAISE\_APPLICATION\_ERROR(-20006, 'Model conflicts during update.');

END IF;

END;

.

RUN;

1. Write one or more triggers to specify that for any tuple in the PC table the hard disk of the PC is at least 100 times greater than its RAM. (Note that the hard disk is in GB, while RAM is in MB).

CREATE OR REPLACE TRIGGER HDRamConstraint

BEFORE INSERT OR UPDATE ON PC

FOR EACH ROW

BEGIN

IF (:new.hd\*1000< :new.ram\*100) THEN

RAISE\_APPLICATION\_ERROR (-20004,hd, ram config error');

END IF;

END;

.

RUN;

Or, maybe do something else depending on the desired semantics:

CREATE OR REPLACE TRIGGER HDRamConstraint

BEFORE INSERT OR UPDATE ON PC

FOR EACH ROW

BEGIN

IF (:new.hd\*1000< :new.ram\*100) THEN

:new.hd := :new.ram/10;

END IF;

END;

.

RUN;

1. Write the needed triggers to enforce that whenever the prices of a Product model are being modified, that then there is a “log tuple” inserted into a special relational table (call it Product- Monitoring) that indicates the model number of the modified product listing, the type of the product (pc, printer, etc), the old price, and the new price, the time of the modification. Note that a command such as “to char(sysdate,’dd-mm-yyyy: hh24:mi’)” could be used to produce a date value. Also, remember to first create the product-Monitoring table with the appropriate attributes.

CREATE TABLE Product\_Monitoring(

Model INT,

type CHAR(2),

OldPrice INT,

NewPrice INT,

when DATE);

Please note that the price attribute cannot be found in the Product table.

It can only be found in the PC and/or the Laptop tables.

CREATE OR REPLACE TRIGGER log\_pricing\_changes

AFTER UPDATE OF Price ON PC

FOR EACH ROW

BEGIN

INSERT INTO Product\_Monitoring

VALUES (:new.model, “PC”, :old.price, :new.price,

to\_char(sysdate,'dd-mm-yyyy: hh24:mi'));

END;

.

RUN;

Similarly, we need to write a **trigger for the tables holding the other**

**product types, such as, Laptops,** etc.

CREATE OR REPLACE TRIGGER log\_laptop\_pricing\_changes

AFTER UPDATE OF Price ON Laptop

FOR EACH ROW

BEGIN

INSERT INTO Product\_Monitoring

VALUES (:new.model, “Laptop”, :old.price, :new.price,

to\_char(sysdate, 'dd-mm-yyyy: hh24:mi'));

END;

.

RUN;

1. Write one or more *triggers* to enforce the constraint that at all times the Product table is consistent with the other two tables. This is an extension of the “foreign” key constraint semantics. **So now, for this exercise, we assume here that you did not have access in your DBMS to any direct support for specifying foreign key constraints.** That is, if in the Product table, a product row is specified as being of PC type, then its model number should also appear in the corresponding PC table. Similarly, if a product in the product table is of type laptop, then its model number must indeed appear in the laptop table. If the type VALUE is “NULL”, then it should appear in none of the other tables. Or, vice versa, check that any tuple that is being inserted into the Laptop or the PC table, either already exists in the Product table (and, now you are simply changing its type over to the right type value), or to make things consistent you also have it added into the Product table as part of the current update.

For the last problem, there are different semantics that could be utilized to solve this problem.

Below we selected and showcase two of the possible solution approaches.

In terms of requirements, one interpretation of the problem could be to **simply forbid the insertion of new models to the PC/Laptop table, and instead require that new models can only be inserted into the Product first.** We provide the details of the solution based on this assumption below.

CREATE TABLE TempLog(model CHAR(10) PRIMARY KEY);

CREATE or replace TRIGGER Q4\_InsertProduct BEFORE INSERT ON Product

FOR EACH ROW DECLARE x NUMBER;

BEGIN

SELECT COUNT(\*) INTO x FROM PC PI WHERE PI.model = :new.model;

IF(:new.type = 'PC' AND x = 0) THEN

INSERT INTO TempLog VALUES (:new.model);

INSERT INTO PC VALUES (:new.model,null,null,null,null,null);

DELETE FROM TempLog;

END IF;

SELECT COUNT(\*) INTO x FROM Laptop LI WHERE LI.model = :new.model;

IF(:new.type = 'Laptop' AND x = 0) THEN

INSERT INTO TempLog VALUES (:new.model);

INSERT INTO Laptop VALUES (:new.model,null,null,null,null,null); DELETE FROM TempLog;

END IF;

END;

CREATE or replace TRIGGER Q4\_UpdateProduct BEFORE UPDATE ON Product

FOR EACH ROW

WHEN (new.type <> old.type)

BEGIN

IF(:old.type = 'Laptop') THEN

INSERT INTO TempLog VALUES (:old.model);

DELETE FROM Laptop LI WHERE LI.model = :old.model;

DELETE FROM TempLog;

IF(:new.type= 'PC') THEN

INSERT INTO TempLog VALUES (:old.model);

INSERT INTO PC VALUES (:new.model,null,null,null,null,null);

DELETE FROM TempLog;

END IF;

END IF;

IF(:old.type <> 'PC' AND :old.type <> 'Laptop') THEN

IF(:new.type = 'PC') THEN

INSERT INTO TempLog VALUES (:new.model);

INSERT INTO PC VALUES (:new.model,null,null,null,null,null);

DELETE FROM TempLog;

END IF;

END IF;

END;

CREATE or replace TRIGGER Q4\_DeleteProduct BEFORE DELETE ON Product

FOR EACH ROW

DECLARE x NUMBER;

BEGIN

SELECT COUNT(\*) INTO x FROM TempLog WHERE TempLog.model =:old.model;

IF (x = 0) THEN

INSERT INTO TempLog VALUES (:old.model);

DELETE FROM PC PI WHERE PI.model = :old.model;

DELETE FROM Laptop LI WHERE LI.model = :old.model;

DELETE FROM TempLog;

END IF;

END;

CREATE or replace TRIGGER Q4\_ForbiddenDeletePC

BEFORE DELETE ON PC

FOR EACH ROW

DECLARE x NUMBER;

BEGIN

SELECT COUNT(\*) INTO x FROM TempLog WHERE TempLog.model = :old.model;

IF (x = 0) THEN

INSERT INTO TempLog VALUES (:old.model);

DELETE FROM Product WHERE Product.model = :old.model;

DELETE FROM TempLog;

END IF;

END;

CREATE or replace TRIGGER Q4\_ForbiddenInsertPC BEFORE INSERT ON PC

FOR EACH ROW

DECLARE x NUMBER;

BEGIN

SELECT COUNT(\*) INTO x FROM TempLog WHERE TempLog.model = :new.model;

IF (x = 0) THEN

RAISE \_APPLICATI0N\_ERR0R(-20004,'Insert to PC Directly is Not Allowed');

END IF;

END;

Another assumption could be that **the new model insertion should only be allowed in the PC/Laptop table, and then indirectly is triggered to be also inserted into the Product. Insertion first into Product would thus not be permitted.**

CREATE OR REPLACE TRIGGER Product\_Insert\_Consistency

BEFORE INSERT ON Product

FOR EACH ROW

BEGIN

IF (:new.type IS NOT NULL AND :new.type IN ('PC', 'Laptop')) THEN RAISE\_APPLICATI0N\_ERR0R(-20003, 'Wrong type in product table.');

END IF;

END;

CREATE OR REPLACE TRIGGER PC\_Product\_Consistency

BEFORE INSERT ON PC

FOR EACH ROW

DECLARE

product\_count INTEGER; product\_type CHAR(IO);

BEGIN

SELECT Count(\*) INTO product\_count

FROM Product P

WHERE :new.model = P.model;

IF (product\_count = 0) THEN RAISE\_APPLICATI0N\_ERR0R(-20004,

'Corresponding product does not exist.');

END IF;

SELECT P.type INTO product\_type

FROM Product P

WHERE :new.model = P.model;

IF (product\_type IS NULL) THEN

UPDATE Product SET type = 'PC' WHERE P.model = :new.model;

ELSIF ('PC' <> product\_type) THEN RAISE\_APPLICATION\_ERROR(-20003,'Wrong type in product table.');

END IF;

END;

CREATE OR REPLACE TRIGGER Product\_Update\_Consistency

BEFORE UPDATE ON Product

FOR EACH ROW

BEGIN

IF (:old.model <> :new.model OR (:old.type IS NOT NULL AND :new.type IS NOT NULL AND :old.type <> :new.type))

THEN RAISE\_APPLICATION\_ERROR(-20004,'Product type or model cannot be changed.');

END IF;

END;

CREATE OR REPLACE TRIGGER PC\_Update\_Consistency

BEFORE UPDATE ON PC

FOR EACH ROW

BEGIN

IF (:old.model <> :new.model) THEN

RAISE\_APPLICATI0N\_ERR0R(-200O6, 'PC model cannot be changed.');

END IF;

END;

# Problem 3: View Specification in SQL [25 pts]

Consider a database schema consisting of three relations, whose schemas are given below (the key fields are underlined). Please load data into your tables and test your answers on Oracle.

*Product( model, manufacturer, type)*

*PC( model. speed, ram, hd, rd, price) Laptop( model, speed, ram, hd, screen, price)*

Consider the following view PCPriceList defined as:

*CREATE VIEW PCPriceList AS Select model, price from PC.*

**Answers (in italics) are interleaved with the questions below.**

1. Use the above view to find the PC with the cheapest price. Show the SQL query. Can you delete from this view directly? Discuss. Show how!

*select \**

*from PCPriceList*

*where price= (select MIN(price) from PCPriceList);*

*By inference, I can conclude that model is the key of this view table. Yes, since this is a single-relation view, the delete will correctly proceed to find the correct tuple to delete.*

1. Can you perform an insert such as: INSERT INTO PCPriceList(model) VALUES (2005)? When yes and when no? Discuss.

*Yes, since this is a single-relation view, this insert can unambiguously proceed.*

*Since 'model' is the key of this relation and there are no other constraints specified, this insert will not cause any constraint violations. That is, all other fields of the base relation can be set to NULL.*

1. What about an insert such as: INSERT INTO PCPriceList (price) VALUES (1700)? Discuss. Show what happens.

*On first sight, since this is a single-relation view, this insert can be translated down into a base update as well. However, since 'model' is the primary key and thus cannot be NULL, this NOT NULL constraint is being violated by the insert and thus the insert is ultimately rejected by the DBMS system.*

1. Now using SQL DDL, define a second view extendedPC(manufacturer, model, speed, ram, hd, rd, price, type). This view will give every PC made by each manufacturer. Can you delete from this extendedPC view? Discuss. Show what happens.

*CREATE VIEW extendedPC ( manufacturer, model,speed, ram, hd, rd, price, type) AS SELECT manufacturer, product.model, speed, ram, hd, rd, price, type*

*FROM Product, PC*

*WHERE Product.model= PC.model;*

*In Oracle, you cannot delete this.*

*However, in principle, you should be able to delete this, because this join view exposes the primary keys of both relations, and because a key-foreign-key connection is used to conduct the join between those 2 tables. Thus we are guaranteed that those keys 'model' serve as keyin the view table. Hence, a delete would map 1-1 to base tuples. So either answer is acceptable, as Jong as you explain your answer.*

*SQL> delete from extendedPC where model=2005; delete from extendedPC where model=2005*

*\**

*ERROR at line 1:*

*ORA-02292: integrity constraint (MAYUB.SYS\_ C00343667) violated* - *child record found*

Sample results for Problem 1:

select \* from PC;

MODEL SPEED RAM HD RD PRICE

---------- ---------- ---------- ---------- ---------- ----------

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000 | 700 | 64 | 10 48xCD | 799 |
| 1002 | 1500 | 128 | 60 12xDVD | 2499 |
| 1003 | 700 | 64 | 10 8xDVD | 899 |
| 1004 | 700 | 64 | 10 6xDVD | 759 |

SQL> select \* from Product; MANUFACTUR MODEL TYPE

|  |  |  |
| --- | --- | --- |
| E | 1000 | PC |
| A | 2002 | Laptop |
| A | 2004 | Laptop |
| A | 2006 | Laptop |

|  |  |  |
| --- | --- | --- |
| B | 1002 | PC |
| B | 1003 | PC |
| C | 1004 | PC |
| B | 2005 | Laptop |

8 rows selected.

SQL> select\* from Laptop;

MODEL

SPEED

RAM HD SCREEN PRICE

## 2002

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2004 | 850 | 64 | 10 | 15.1 | 2583 |
| 2005 | 800 | 96 | 10 | 15.1 | 1999 |
| 2006 | 850 | 64 | 10 | 15.1 | 2584 |

700 64

5 12.1

14

SQL> create view PCPriceList as select model. price from PC;

View created.

SQL> select \* from PCPriceList; MODEL PRICE

|  |  |
| --- | --- |
| 1000 | 799 |
| 1002 | 2499 |
| 1003 | 899 |
| 1004 | 759 |

SQL> select\* from PCPriceList where price= (select min(price) from PCPriceList); MODEL PRICE

## 1004 759

SQL> select \* from PCPriceList; MODEL PRICE

|  |  |
| --- | --- |
| 1000 | 799 |
| 1002 | 2499 |
| 1003 | 899 |
| 1004 | 759 |

SQL> delete from PCPriceList where model=1000;

1 row deleted.

SQL> select \* from PCPriceList;

MODEL PRICE

|  |  |
| --- | --- |
| 1002 | 2499 |
| 1003 | 899 |
| 1004 | 759 |

SQL> select\* from PC;

MODEL

SPEED

RAM HDRD

PRICE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1002 | 1500 | 128 | 60 12xDVD | 2499 |
| 1003 | 700 | 64 | 10 8xDVD | 899 |
| 1004 | 700 | 64 | 10 6xDVD | 759 |

SQL> select \* from Product; MANUFACTUR MODEL TYPE

|  |  |  |
| --- | --- | --- |
| E | 1000 | PC |
| A | 2002 | Laptop |
| A | 2004 | Laptop |
| A | 2006 | Laptop |
| B | 1002 | PC |
| B | 1003 | PC |
| C | 1004 | PC |
| B | 2005 | Laptop |

8 rows selected.

SQL> insert into PCPriceList(model) values (2005) ; 1 row created.

SQL> select \* from PCPriceList; MODEL PRICE

|  |  |
| --- | --- |
| 1002 | 2499 |
| 1003 | 899 |
| 1004 | 759 |
| 2005 |  |

SQL> select\* from PC;

MODEL

SPEED

RAM HDRD

PRICE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1002 | 1500 | 128 | 60 12xDVD | 2499 |
| 1003 | 700 | 64 | 10 8xDVD | 899 |

1004

2005

700 64 10 6xDVD 759

SQL> insert into PCPriceList(price) values (1700) ; insert into PCPriceList(price) values (1700)

\*

ERROR at line 1:

ORA-01400: cannot insert NULL into ("MAYUB"."PC"."MODEL")

SQL> create view extendedPC(manufacturer, model. speed, ram, hd, rd, price, type) as select manufacturer, Product.model, speed, ram, hd, rd, price, type from Product. PC where Product.model= PC.model;

View created.

SQL> select\* from extendedPC;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| MANUFACTUR | MODEL | SPEED | RAM | HD | RD | PRICE | TYPE |
| B | 1002 | 1500 | 128 | 60 | 12xDVD | 2499 | PC |
| B | 1003 | 700 | 64 | 10 | 8xDVD | 899 | PC |
| C | 1004 | 700 | 64 | 10 | 6xDVD | 759 | PC |
| B | 2005 |  |  |  |  |  | Laptop |

SQL> delete from extendedPC where model=2005 ; delete from extendedPC where model=2005

\*

ERROR at line 1:

ORA-02292: integrity constraint (MAYUB.SYS\_C00343667) violated - child record found

SQL> spool off;