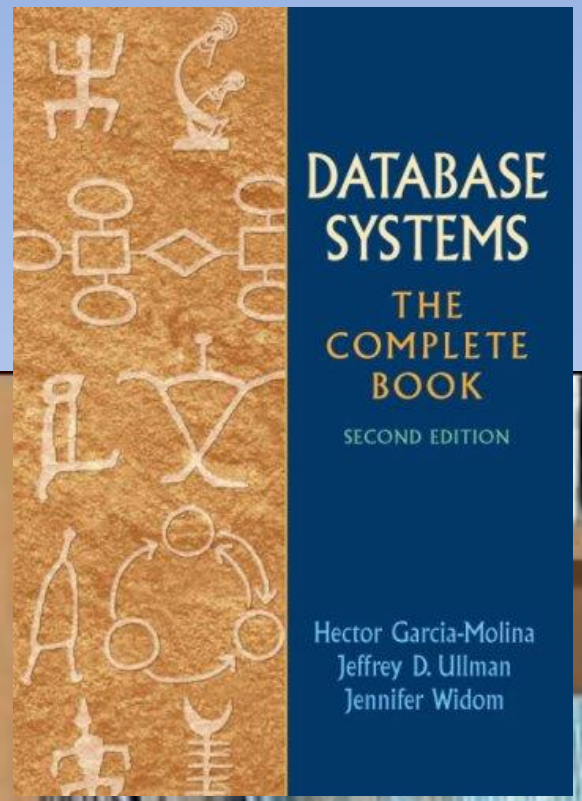
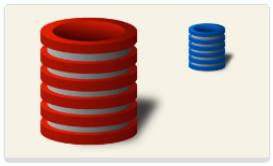
Chapter 8: Views & **Indexes**



DB Courses



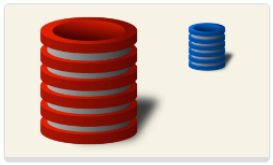
Unified Modeling Language

Databases - DB9
Started - Jun 09, 2014

View Course

Your final grade: **100%**.

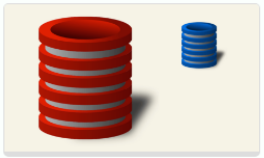
Download Statement (PDF)



Relational Design Theory

Databases - DB8
Started - Jun 09, 2014

Your final grade: **92%**.



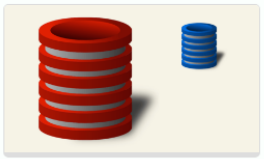
Views and Authorization

Databases - DB12
Started - Jun 09, 2014

View Course

Your final grade: **90%**.

Download Statement (PDF)



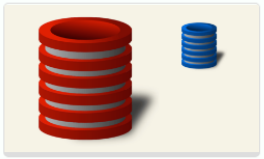
Constraints and Triggers

Databases - DB11
Started - Jun 09, 2014

View Course

Your final grade: **92%**.

Download Statement (PDF)



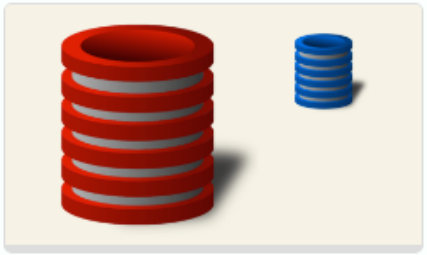
Indexes and Transactions

Databases - DB10
Started - Jun 09, 2014

View Course

Indexes

- *Index* = data structure used to speed access to tuples of a relation, given values of one or more attributes.
- Could be a hash table, but in a DBMS it is always a balanced search tree with giant nodes (a full disk page) called a *B-tree*.



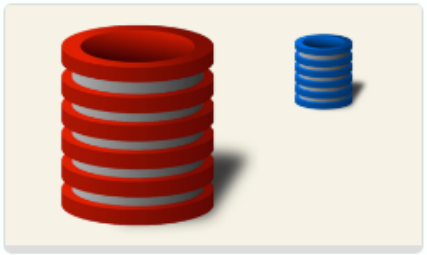
Recursion in SQL

Databases - DB14
Started - Jun 09, 2014

View Course

Your final grade: **100%**.

Download Statement (PDF)



On-Line Analytical Processing

Databases - DB13
Started - Jun 09, 2014

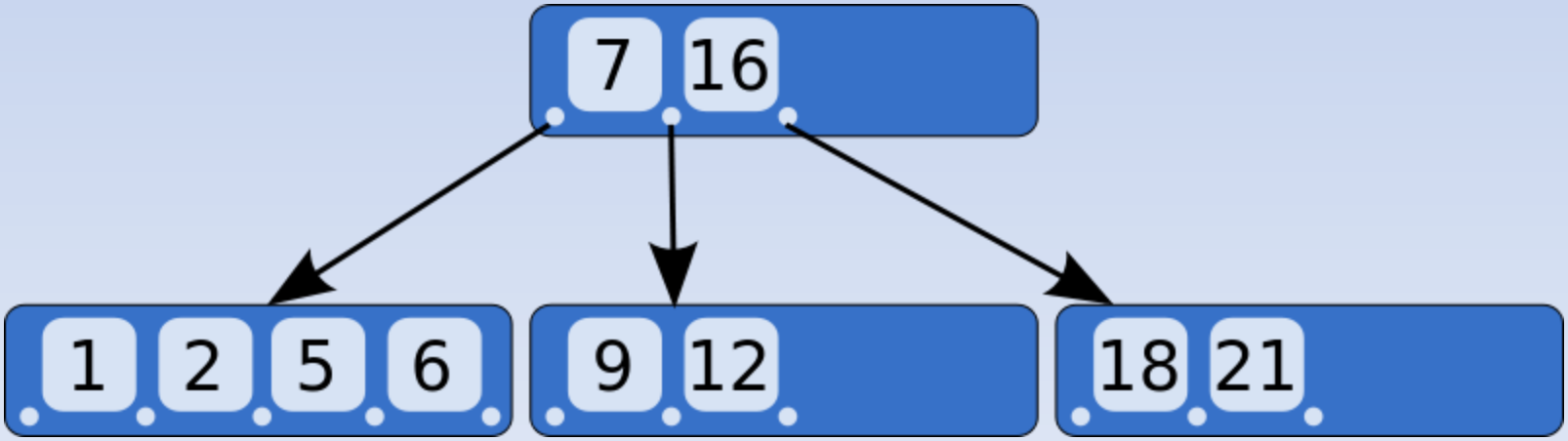
View Course

Your final grade: **100%**.

Download Statement (PDF)

B-Tree: Quick Glance

- According to Knuth's definition, a B-tree of order *m* is a tree which satisfies the following properties:
  - Every node has at most *m* children.
  - Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.
  - The root has at least two children if it is not a leaf node.
  - A non-leaf node with *k* children contains *k*−1 keys.
  - All leaves appear in the same level.
- For example, in a 2-3 B-Tree (often simply referred to as a **2-3 tree**), each internal node may have only 2 or 3 child nodes.
- Example below is 3-5 Tree



Declaring Indexes

- No standard!
- Typical syntax:

```
CREATE INDEX PersonInd ON
  Person(Pid);
CREATE INDEX MovieInd ON
  Movies(mid);
```

Using Indexes

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use PersonInd and MovieInd to find ratings for person and movie.

Using Indexes

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: In Movies DB
  - use pid (Person ID) and mid (Movie ID) to find ratings for person and movie.

```
alter table movie_ratings add index (pid);
alter table movie_ratings add index (mid);
```

Using Indexes

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: In Movies DB
  - use pid (Person ID) and mid (Movie ID) to find ratings for person and movie.

```
alter table movie_ratings add index (pid);
alter table movie_ratings add index (mid);
```

Using Indexes

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: In Movies DB
  - use pid (Person ID) and mid (Movie ID) to find ratings for person and movie.

```
SELECT count(*) FROM ratings a, ratings b
WHERE a.mid = b.mid AND a.rating = b.rating
AND a.pid < b.pid;
```

40 seconds w/ Index

10 minutes plus w/o Index

Database Tuning

- A major problem in making a database run fast is deciding which indexes to create.
- Pro: An index speeds up queries that can use it.
- Con: An index slows down all modifications on its relation because the index must be modified too.

Example: Tuning

- Suppose the only things we did with our Movie Ratings database was:
  1. Insert new facts into a relation (10%).
  2. Find the average rating for a movie (90%).

Tuning Advisors

- A major research thrust.
  - Because hand tuning is so hard.
- An advisor gets a *query load*, e.g.:
  1. Choose random queries from the history of queries run on the database, or
  2. Designer provides a sample workload.

Tuning Advisors --- (2)

- The advisor generates candidate indexes and evaluates each on the workload.
  - Feed each sample query to the query optimizer, which assumes only this one index is available.
  - Measure the improvement/degradation in the average running time of the queries.

Question

MULTIPLE CHOICE (2/2 points)

[Q1] Consider the following relational schema:

```
Course(courseName unique, department, instrID)
Instructor(instrID unique, office)
Student(studentID unique, major)
Enroll(studentID, courseName, unique (studentID,courseName))
```

Suppose there are five types of queries commonly asked on this schema:

- Given a course name, find the department offering that course.
- List all studentIDs together with all of the departments they are taking courses in.
- Given a studentID, find the names of all courses the student is enrolled in.
- List the offices of instructors teaching at least one course.
- Given a major, return the studentIDs of students in that major.

Which of the following indexes could NOT be useful in speeding up execution of one or more of the above queries?

Index on Student.studentID

Index on Enroll.studentID

Index on Student.major

Index on Enroll.courseName

Question

MULTIPLE CHOICE (2/2 points)

[Q1] Consider the following relational schema:

```
Course(courseName unique, department, instrID)
Instructor(instrID unique, office)
Student(studentID unique, major)
Enroll(studentID, courseName, unique (studentID,courseName))
```

Suppose there are five types of queries commonly asked on this schema:

- Given a course name, find the department offering that course.
- List all studentIDs together with all of the departments they are taking courses in.
- Given a studentID, find the names of all courses the student is enrolled in.
- List the offices of instructors teaching at least one course.
- Given a major, return the studentIDs of students in that major.

Which of the following indexes could NOT be useful in speeding up execution of one or more of the above queries?

- (●) Index on Student.studentID ✔
- ( ) Index on Enroll.studentID
- ( ) Index on Student.major
- ( ) Index on Enroll.courseName

[Q2] Consider a table storing temperature readings taken by sensors:

```
Temps(sensorID, time, temp)
```

Assume the pair of attributes [sensorID,time] is a key. Consider the following query:

```
select * from Temps
where sensorID = 'sensor541'
and time = '05:11:02'
```

Consider the following scenarios:

A - No index is present on any attribute of Temps

B - An index is present on attribute sensorID only

C - An index is present on attribute time only

D - Separate indexes are present on attributes sensorID and time

E - A multi-attribute index is present on (sensorID,time)

Suppose table Temps has 50 unique sensorIDs and each sensorID has exactly 20 readings. Furthermore there are exactly 10 readings for every unique time in Temps.

For each scenario A-E, determine the maximum number of tuples that might be accessed to answer the query, assuming one "best" index is used whenever possible. (Don't count the number of index accesses.) Which of the following combinations of values is correct?

> A:1000, B:20, E:1
>
> A:1000, D:20, E:10
>
> C:1000, D:20, E:1
>
> A:1000, B:20, E:10

[Q2] Consider a table storing temperature readings taken by sensors:

```
Temps(sensorID, time, temp)
```

Assume the pair of attributes [sensorID,time] is a key. Consider the following query:

```
select * from Temps
where sensorID = 'sensor541'
and time = '05:11:02'
```

Consider the following scenarios:

A - No index is present on any attribute of Temps

B - An index is present on attribute sensorID only

C - An index is present on attribute time only

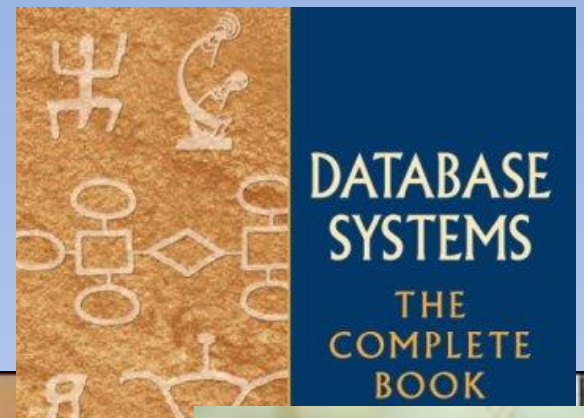
D - Separate indexes are present on attributes sensorID and time

E - A multi-attribute index is present on (sensorID,time)

Suppose table Temps has 50 unique sensorIDs and each sensorID has exactly 20 readings. Furthermore there are exactly 10 readings for every unique time in Temps.

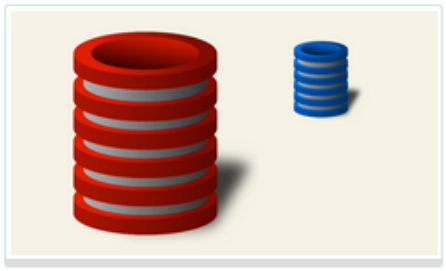
For each scenario A-E, determine the maximum number of tuples that might be accessed to answer the query, assuming one "best" index is used whenever possible. (Don't count the number of index accesses.) Which of the following combinations of values is correct?

- (●) A:1000, B:20, E:1 ✔
- ( ) A:1000, D:20, E:10
- ( ) C:1000, D:20, E:1
- ( ) A:1000, B:20, E:10



Chapter 7: Constraints and **Triggers**
Chapter 9: SQL in a Server Environment
Stored Procedures

Stanford Online



Databases

Course Started - Jun 09, 2014 at 15:00 UTC

DB11 Constraints and Triggers

Your final grade: **92%**.

Download Your Statement (PDF)

View Course

Email Settings *Unenroll*

Triggers

"Event-Condition-Action Rules"

When *event* occurs, check *condition*; if true, do *action*

1) Move monitoring logic from apps into DBMS

2) Enforce constraints
   - Beyond what constraint system supports
   - Automatic constraint "repair"

✳ Implementations vary significantly

Triggers in SQL

```
Create Trigger name
Before|After|Instead Of [insert|delete|update]

[ For Each Row ]

When ( condition )

action
```

Triggers in SQL

Calculate the total amount inserted.

```
CREATE TRIGGER tr_AccountSum
AFTER INSERT ON BankAccount
FOR EACH ROW
BEGIN
   set @sum = @sum + new.amount;
END
```

Triggers in SQL

Calculate the total amount inserted.

```
set @sum=0;
insert into BankAccount values
(123, 801, 500),
(235, 701, 500),
(235, 702, 500),
select @sum;
```

Triggers in SQL

Calculate the total amount inserted.

```
87
88  set @sum=0;
89  insert into BankAccount values
90     (100, 901, 500),
91     (123, 801, 500),
92     (235, 701, 500),
93     (235, 702, 500),
94     (400, 401, 500);
95  select @sum;
96
97  select trigger_schema, Trigger_name, Event_manipulation
98     from information_schema.Triggers;
99
```

r: File: Autosize:

@sum
2500

Triggers in SQL

Monies distribution between accounts table.

```
CREATE TABLE Ivel(
 PersonID char(3), AccountID int,
 Percentage int,
 InsertDate date default null,
 InsertTime time default null,
 ModifyDate date default null,
 ModifyTime time default null,
 LastUpdatedBy varchar(132) default null,
 FOREIGN KEY (PersonID) REFERENCES
  person(PersonID)
   On Delete Cascade On Update Set Null,
 FOREIGN KEY (AccountID) REFERENCES
  BankAccount(AccountID)
   On Delete Cascade On Update Set Null);
```

Triggers in SQL

Monies distribution between accounts table.

```
CREATE TRIGGER tr_IvelInsertDate
    before insert on Ivel
for each row
begin
    set new.InsertDate = curdate();
    set new.InsertTime = curtime();
    set new.ModifyDate = curdate();
    set new.ModifyTime = curtime();
    set new.LastUpdatedBy = user();
end
```

Triggers in SQL

Monies distribution between accounts table.

```
CREATE TRIGGER tr_IvelModifyDate
before update on Ivel
for each row
begin
  set new.ModifyDate = curdate();
  set new.ModifyTime = curtime();
  set new.LastUpdatedBy = user();
end
```

Triggers in SQL

Monies distribution between accounts table.

```
103 • insert into Ivel(PersonID, AccountID, Percentage) values
104     (235, 701, 75),
105     (235, 702, 25);
106
107 • select * from Ivel;
108
```

lter: | File: | Autosize:

PersonID	AccountID	Percentage	InsertDate	InsertTime	ModifyDate	ModifyTime	LastUpdatedBy
235	701	75	2013-11-01	15:27:43	2013-11-01	15:27:43	cs126@localhost
235	702	25	2013-11-01	15:27:43	2013-11-01	15:27:43	cs126@localhost

Triggers in SQL

Monies distribution between accounts table.

```
108
109 update Ivel set percentage = 50
110   where personid = 235;
111
112
113
114
```

er: File: Autosize:

PersonID	AccountID	Percentage	InsertDate	InsertTime	ModifyDate	ModifyTime	LastUpdatedBy
235	701	50	2013-11-01	15:27:43	2013-11-01	15:30:02	cs126@localhost
235	702	50	2013-11-01	15:27:43	2013-11-01	15:30:02	cs126@localhost

Triggers in SQL

Inserting an account where person doesn't exist.

```
CREATE TRIGGER tr_NewPerson
before insert on BankAccount
for each row
begin
if not exists
  (select * from Person
    where PersonID = new.PersonID) then
  set new.PersonID = 000;
end if;
end
```

Triggers in SQL

Inserting an account where person doesn't exist.

```
insert into Person
   values(000, NULL, '0000-00-00');

set @sum=0;
insert into BankAccount values
  (100, 901, 500), (123, 801, 500),
  (235, 701, 500), (235, 702, 500),
  (400, 401, 500);
select @sum;
```

Triggers in SQL

Inserting an account where person doesn't exist.

```
76  insert into Person
77    values(123, 'John Sigma123', '1933-01-01');
78  insert into Person
79    values(235, 'John Sigma235', '1955-01-01');
80  insert into Person
81    values(000, NULL, '0000-00-00');
82
83
84  delete from BankAccount;
85  insert into BankAccount values(100, 901, 500);
86  insert into BankAccount values(123, 801, 500);
87  insert into BankAccount values(235, 701, 500);
88  insert into BankAccount values(235, 702, 500);
89  insert into BankAccount values(400, 401, 500);
90
91  set @sum=0;
92  insert into BankAccount values
93     (100, 901, 500),
94     (123, 801, 500),
95     (235, 701, 500),
96     (235, 702, 500),
97     (400, 401, 500);
98  select @sum;
```

Filter: File: Autosize:

@sum
2500

```
1  SELECT * FROM bank.bankaccount;
```

Filter: File: Autosize:

PersonID	AccountID	Amount
0	901	500
123	801	500
235	701	500
235	702	500
0	401	500

Triggers in SQL (general syntax)

```
Create Trigger name
Before|After|Instead Of events
[ referencing-variables ]
[ For Each Row ]
When ( condition )
action
```

Exercise 7.2.2a

- Write the following constraints on attribues from our example schema:

Product(maker, model, ctype)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)

a) The speed of a laptop must be at least 2.0.

In MySQL we'll do this with a Trigger.

Exercise 7.2.2a

a) The speed of a laptop must be at least 2.0.

```
delimiter //
CREATE TRIGGER tr_ex722a_i
BEFORE insert ON laptop
FOR EACH ROW
BEGIN
   IF (NEW.speed<2.0) THEN
      SIGNAL sqlstate '45000'
        SET message_text = ‘The speed of a laptop must be at least 2.0.’;
   END IF;
END//
delimiter ;
```

Exercise 7.2.2a

```
159 • insert into laptop value
160   (2002,  1.73,   1024,   80, 17.0,   949);
161
```

Output

Action Output

	Time	Action	Message
28	11:13:25	CREATE TRIGGER tr_ex722a_i BEFORE insert ON laptop FOR EACH ROW BEGIN ...	0 row(s) affected
29	11:13:36	insert into laptop value (2002,1.73,1024,80,17.0,949)	Error Code: 1644. The speed of a laptop must be at least 2.0.

IF (NEW.speed<2.0) THEN

```
162 • insert into laptop value
163   (2002,  2.73,   1024,   80, 17.0,   949);
164
```

Output

Action Output

	Time	Action	Message
29	11:13:36	insert into laptop value (2002,1.73,1024,80,17.0,949)	Error Code: 1644. The speed of a laptop must be at least 2.0.
30	11:16:40	insert into laptop value (2002,2.73,1024,80,17.0,949)	1 row(s) affected

Exercise 7.2.2a
A little more advanced.

a) The speed of a laptop must be at least 2.0.

```
delimiter //
CREATE TRIGGER tr_ex722a_i
BEFORE insert ON laptop
FOR EACH ROW
BEGIN
      DECLARE msg varchar(255);
      IF (NEW.speed<2.0)
      THEN
          SET msg =
            concat('Constraint tr_ex722a_i violated: Laptop speed must be >2.0: ',
              cast(new.speed as char));
          SIGNAL sqlstate '45000' SET message_text = msg;
      END IF;
   END//
delimiter ;
```

Exercise 7.2.2a

a) The speed of a laptop must be at least 2.0.

delimiter //
CREATE TRIGGER tr_ex722a_i
BEFORE insert ON laptop
FOR EACH ROW

```
delimiter //
CREATE TRIGGER tr_ex722a_i BEFORE insert ON laptop
     FOR EACH ROW
     BEGIN
        DECLARE msg varchar(255);
        IF (NEW.speed<2.0)
        THEN
           SET msg = concat('Constraint tr_ex722a_i violated: Laptop speed must be >2.0: ', cast(new.speed as char));
           SIGNAL sqlstate '45000' SET message_text = msg;
        END IF;
     END//
delimiter ;
```

Output

Action Output

	Time	Action	Message
18	11:01:48	drop trigger tr_ex722a_i	0 row(s) affected
19	11:01:55	CREATE TRIGGER tr_ex722a_i BEFORE insert ON laptop FOR EACH ROW BEGIN ...	0 row(s) affected

Exercise 7.2.2a

a) The speed of a laptop must be at least 2.0.

```
148 • insert into laptop value
149   (2002,  1.73,   1024,   80, 17.0,   949);
150
```

Output

Action Output

	Time	Action	Message
19	11:01:55	CREATE TRIGGER tr_ex722a_i BEFORE insert ON laptop FOR EACH ROW BEGIN ...	0 row(s) affected
20	11:03:34	insert into laptop value (2002,1.73,1024,80,17.0,949)	Error Code: 1644. Constraint tr_ex722a_i violated: Laptop speed must be >2.0: 1.73

```
IF (NEW.speed<2.0)
THEN
    SET msg =
      concat('Constraint tr  ex722a  i violated: Laptop speed must be >2.0: ',
```

```
151 • insert into laptop value
152   (2002,  2.73,   1024,   80, 17.0,   949);
153
```

Output

Action Output

	Time	Action	Message
20	11:03:34	insert into laptop value (2002,1.73,1024,80,17.0,949)	Error Code: 1644. Constraint tr_ex722a_i violated: Laptop speed must be >2.0: 1.73
21	11:04:56	insert into laptop value (2002,2.73,1024,80,17.0,949)	1 row(s) affected

Exercise 7.2.2a
Will Need: Update & Insert Trigger

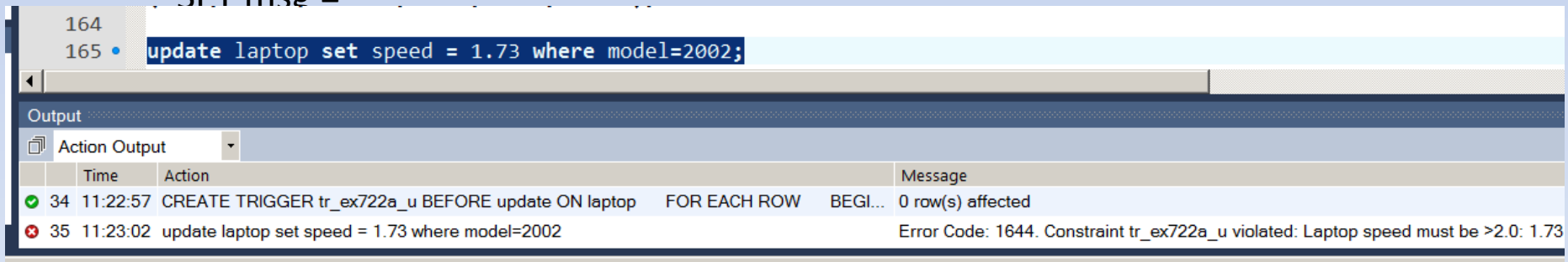
a) The speed of a laptop must be at least 2.0.

```
delimiter //
CREATE TRIGGER tr_ex722a_u
BEFORE update ON laptop
    FOR EACH ROW
    BEGIN
      DECLARE msg varchar(255);
      IF (NEW.speed<2.0)
      THEN
        SET msg =
          concat('Constraint tr_ex722a_u violated: Laptop speed must be >2.0: ',
          cast(new.speed as char));
        SIGNAL sqlstate '45000' SET message_text = msg;
      END IF;
    END//delimiter ;
```

Exercise 7.2.2a
Will Need: Update & Insert Trigger

a) The speed of a laptop must be at least 2.0.

```
delimiter //
CREATE TRIGGER tr_ex722a_u
BEFORE update ON laptop
    FOR EACH ROW
    BEGIN
      DECLARE msg varchar(255);
      IF (NEW.speed<2.0)
      THEN
        SET msg =
```



Exercise 7.5.3a

- Write the following as triggers. In ease case, disallow or undo the modification if it does not satisfy the stated constraint. The database schema is from the battleships schema.

  Classes(class, stype, country, numGuns, bore, displacement);
  Battles(bname, bdate);
  Outcomes(ship, battle, result);
  Ships(sname, class, launched);

a) When a new class is inserted into Classes, also insert a ship with the name of that class and a NULL launch date.

Exercise 7.5.3a

a) When a new class is inserted into Classes, also insert a ship with the name of that class and a NULL launch date.

```
delimiter //
CREATE TRIGGER NewClassTrigger
AFTER INSERT ON Classes
FOR EACH ROW
BEGIN
    INSERT INTO Ships VALUE (new.class, New.class, NULL);
END//
delimiter ;
```

Exercise 7.5.3a

a) When a new class is inserted into Classes, also insert a ship with the name of that class and a NULL launch date.

```
delimiter //
CREATE TRIGGER NewClassTrigger
AFTER INSERT ON Classes
FOR EACH ROW
BEGIN
    INSERT INTO Ships VALUE (new.class, New.class, NULL);
END//
delimiter ;

insert into classes values
 ('test1', 'bb', ‘USA', 8, 15, 33000),
 ('test2', 'bb', ‘USA', 8, 15, 36000),
 ('test3', 'bb', ‘USA', 8, 15, 42000);
```

Exercise 7.5.3a

a) W of t

```
delimi
CREAT
AFTER
FOR EA
BEGIN
    INS
END//
delimi

insert
 ('test1
 ('test2
 ('test3
```

```
80 • select * from classes;
81
```

Result Set Filter: | Edit: | Export/Import: | Wrap Cell Content:

Export recordset to an external file

class	stype	country	numGuns	bore	displacement
Bismarck	bb	Germany	8	15	42000
Iowa	bb	USA	9	16	46000
Kongo	bc	Japan	8	14	32000
North Carolina	bb	USA	9	16	37000
Pennsylvania	bb	USA	12	14	29158
Renown	bc	Gt. Britain	6	15	32000
Revenge	bb	USA	12	14	32000
Yamato	bb	Japan	9	18	65000
NULL	NULL	NULL	NULL	NULL	NULL

classes 1 ×

Output

Action Output

	Time	Action	Message
3	11:41:55	CREATE TRIGGER NewClassTrigger AFTER INSERT ON Classes FOR EACH ROW BEGIN...	0 row(s) affected
4	11:43:17	select * from classes LIMIT 0, 1000	8 row(s) returned

Exercise 7.5.3a

```
64  insert into classes values
65  ('test1', 'bb', 'USA', 8, 15, 33000),
66  ('test2', 'bb', 'USA', 8, 15, 36000),
67  ('test3', 'bb', 'USA', 8, 15, 42000);
68
```

Output

Action Output

	Time	Action	Message
4	11:43:17	select * from classes LIMIT 0, 1000	8 row(s) returned
5	11:44:20	insert into classes values ('test1', 'bb', 'USA', 8, 15, 33000), ('test2', 'bb', 'USA', 8, 15, 360...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

class	stype	country	numGuns	bore	displacement
Bismarck	bb	Germany	8	15	42000
Iowa	bb	USA	9	16	46000
Kongo	bc	Japan	8	14	32000
North Carolina	bb	USA	9	16	37000
Pennsylvania	bb	USA	12	14	29158
Renown	bc	Gt. Britain	6	15	32000
Revenge	bb	USA	12	14	32000
test1	bb	USA	8	15	33000
test2	bb	USA	8	15	35000
test3	bb	USA	8	15	35000
Yamato	bb	Japan	9	18	65000
NULL	NULL	NULL	NULL	NULL	NULL

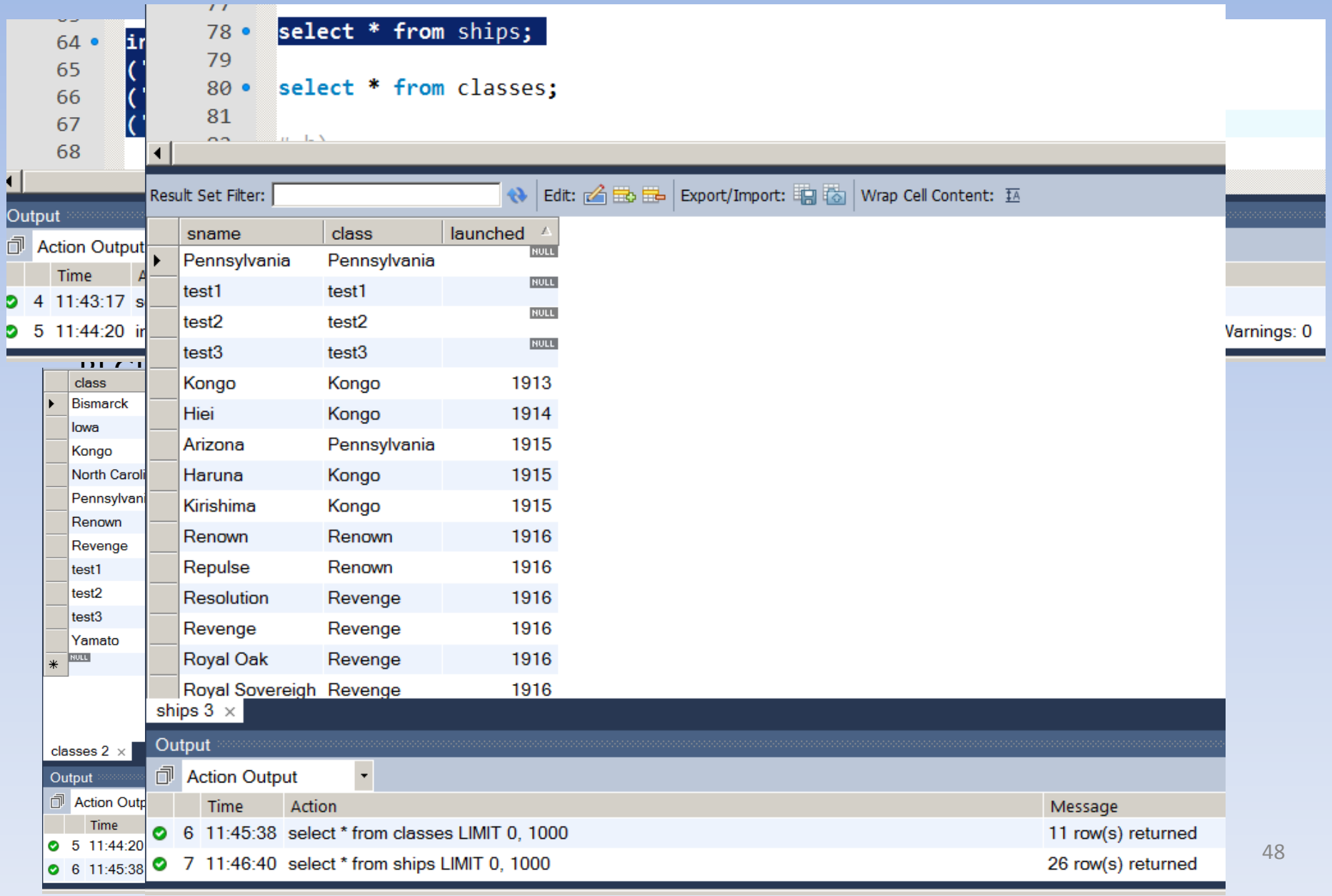
classes 2

Output

Action Output

	Time	Action	Message
5	11:44:20	insert into classes values ('test1', 'bb', 'USA', 8, 15, 33000), ('test2', 'bb', 'USA', 8, 15, 360...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
6	11:45:38	select * from classes LIMIT 0, 1000	11 row(s) returned

Exercise 7.5.3a



Question

[Q2] Consider the following trigger over a table R(a,b), specified using the trigger language of the SQL standard:

```
CREATE TRIGGER Rins
AFTER INSERT ON R
REFERENCING NEW ROW AS new
FOR EACH ROW
INSERT INTO R(a,b)
  (SELECT DISTINCT R.a, new.b
   FROM R
   WHERE R.b = new.a)
  EXCEPT
  (SELECT DISTINCT a,b FROM R)
```

Suppose table R is empty initially. We issue three commands to insert tuples into R: first we insert (1,2), then we insert (2,3), then we insert (3,4). After some of these inserts, trigger **Rins** may insert further tuples into R, which may activate the trigger recursively. After all the inserts are done, which of these tuples is NOT in table R?

(2,4)

(1,4)

(3,1)

(1,3)

Question

[Q2] Consider the following trigger over a table R(a,b), specified using the trigger language of the SQL standard:

```
CREATE TRIGGER Rins
AFTER INSERT ON R
REFERENCING NEW ROW AS new
FOR EACH ROW
INSERT INTO R(a,b)
  (SELECT DISTINCT R.a, new.b
   FROM R
   WHERE R.b = new.a)
  EXCEPT
  (SELECT DISTINCT a,b FROM R)
```

Suppose table R is empty initially. We issue three commands to insert tuples into R: first we insert (1,2), then we insert (2,3), then we insert (3,4). After some of these inserts, trigger **Rins** may insert further tuples into R, which may activate the trigger recursively. After all the inserts are done, which of these tuples is NOT in table R?

- ○ (2,4)
- ○ (1,4)
- ◉ (3,1) ✔
- ○ (1,3)

Question

Students at your hometown high school have decided to organize their social network using databases. So far, they have collected information about sixteen students in four grades, 9-12. Here's the schema:

Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

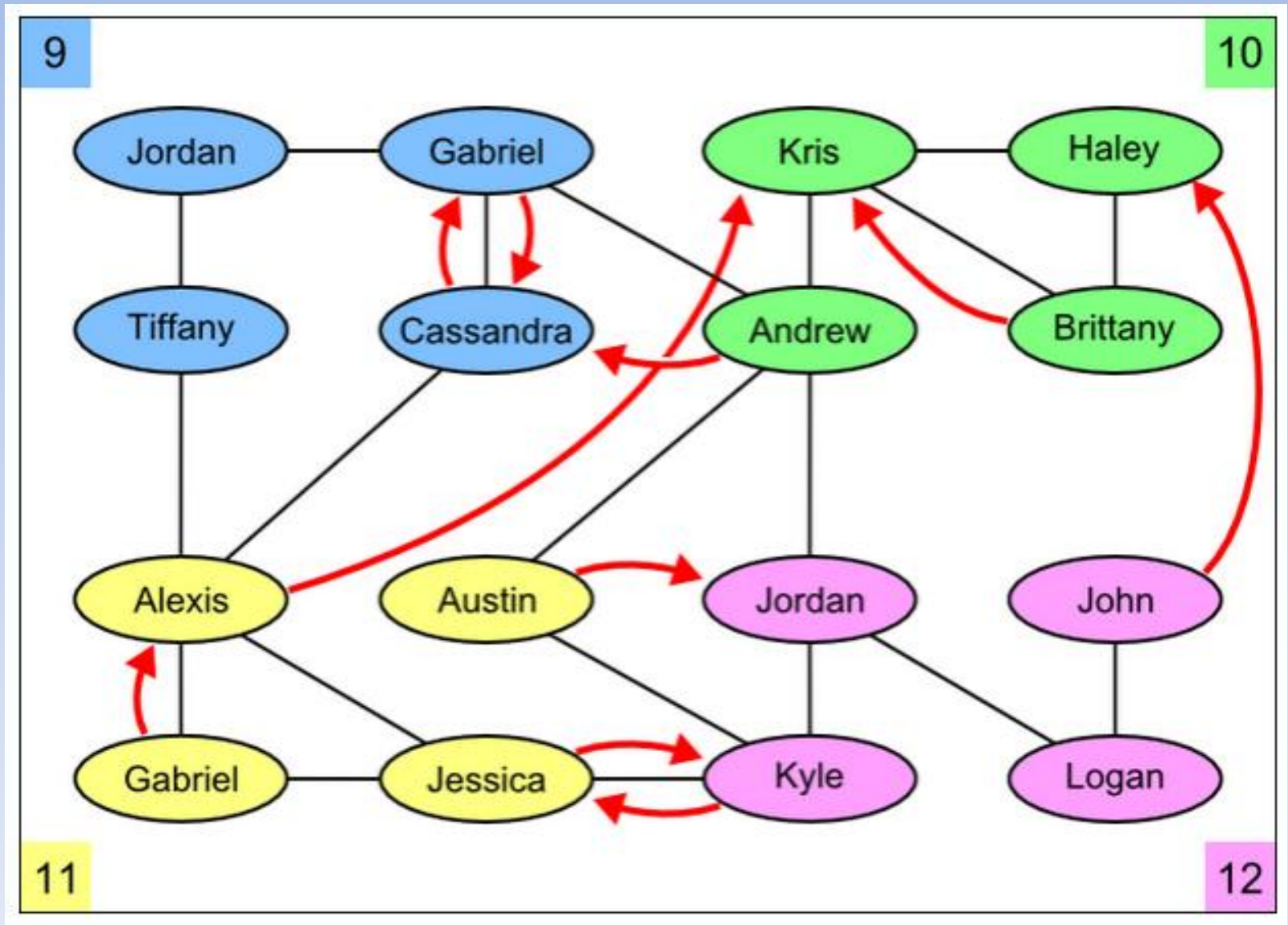
Friend (ID1, ID2)
English: The student with *ID1* is friends with the student with *ID2*. Friendship is mutual, so if (123, 456) is in the Friend table, so is (456, 123).

Likes (ID1, ID2)
English: The student with *ID1* likes the student with *ID2*. Liking someone is not necessarily mutual, so if (123, 456) is in the Likes table, there is no guarantee that (456, 123) is also present.

Your triggers will run over a small data set conforming to the schema. View the database. (You can also download the schema and data.)

For your convenience, here is a graph showing the various connections between the people in our database. 9th graders are blue, 10th graders are green, 11th graders are yellow, and 12th graders are purple. Undirected black edges indicate friendships, and directed red edges indicate that one person likes another person.

Question


9
10
Jordan
Gabriel
Kris
Haley
Tiffany
Cassandra
Andrew
Brittany
Alexis
Austin
Jordan
John
Gabriel
Jessica
Kyle
Logan
11
12

Question

Q1 (1/1 point)

Write a trigger that makes new students named 'Friendly' automatically like everyone else in their grade. That is, after the trigger runs, we should have ('Friendly', A) in the Likes table for every other Highschooler A in the same grade as 'Friendly'.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.

Question

Q1 (1/1 point)

Write a trigger that makes new students named 'Friendly' automatically like everyone else in their grade. That is, after the trigger runs, we should have ('Friendly', A) in the Likes table for every other Highschooler A in the same grade as 'Friendly'.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.

```
CREATE TRIGGER q1 after insert ON highschooler
     FOR EACH ROW when new.name='Friendly'
     BEGIN
        insert into Likes select new.id, id from highschooler where id<>new.id and
        new.grade = grade;
     END
```

Question

To check your trigger(s), we first ran the following data modification statement(s): *insert into Highschooler values (1000, 'Friendly', 9);*
insert into Highschooler values (2000, 'Friendly', 11);
insert into Highschooler values (3000, 'Unfriendly', 10).

We then ran the following query: *select H1.name, H1.grade, H2.name, H2.grade from Likes L, Highschooler H1, Highschooler H2 where L.ID1 = H1.ID and L.ID2 = H2.ID order by H1.name, H1.grade, H2.name, H2.grade*

Alexis	11	Kris	10
Andrew	10	Cassandra	9
Austin	11	Jordan	12
Brittany	10	Kris	10
Cassandra	9	Gabriel	9
Friendly	9	Cassandra	9
Friendly	9	Gabriel	9
Friendly	9	Jordan	9
Friendly	9	Tiffany	9
Friendly	11	Alexis	11
Friendly	11	Austin	11
Friendly	11	Gabriel	11
Friendly	11	Jessica	11
Gabriel	9	Cassandra	9
Gabriel	11	Alexis	11
Jessica	11	Kyle	12
John	12	Haley	10
Kyle	12	Jessica	11

Question

Q2 (1/1 point)

Write one or more triggers to manage the grade attribute of new Highschoolers. If the inserted tuple has a value less than 9 or greater than 12, change the value to NULL. On the other hand, if the inserted tuple has a null value for grade, change it to 9.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.
- To create more than one trigger, separate the triggers with a vertical bar (|).

Question

- Two Parts:
  - Trigger Condition to find right situation.
  - Update Query to make the actual change.

Question

- Trigger 1: “If the inserted tuple has a value less than 9 or greater than 12, change the value to NULL.”
  - After Insert

CREATE TRIGGER q1 after insert ON highschooler

Question

- Trigger 1: “If the inserted tuple has a value less than 9 or greater than 12, change the value to NULL.”
  - After Insert
  - For Each Row When…

```
CREATE TRIGGER q1 after insert ON highschooler
    FOR EACH ROW when new.grade<9 or new.grade > 12
    BEGIN
        …make changes…
    END
```

Question

- Trigger 1: “If the inserted tuple has a value less than 9 or greater than 12, change the value to NULL.”
  - After Insert
  - For Each Row When…
  - Update Query:

```
CREATE TRIGGER q1 after insert ON highschooler
    FOR EACH ROW when new.grade<9 or new.grade > 12
    BEGIN
        update highschooler set grade = null where id=new.id;
    END
```

Question

- Trigger 2: “On the other hand, if the inserted tuple has a null value for grade, change it to 9.”
  - After Insert
  - For Each Row When…
  - Update Query:

```
CREATE TRIGGER q1 after insert ON highschooler
   FOR EACH ROW when new.grade<9 or new.grade > 12
   BEGIN
     update highschooler set grade = null where id=new.id;
   END
|
CREATE TRIGGER q2 after insert ON highschooler
   FOR EACH ROW when ____________________
   BEGIN
     update highschooler set _________ where _____________;
   END
```

Question

Q2 (1/1 point)

Write one or more triggers to manage the grade attribute of new Highschoolers. If the inserted tuple has a value less than 9 or greater than 12, change the value to NULL. On the other hand, if the inserted tuple has a null value for grade, change it to 9.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.
- To create more than one trigger, separate the triggers with a vertical bar (|).

```
CREATE TRIGGER q1 after insert ON highschooler
     FOR EACH ROW when new.grade<9 or new.grade > 12
     BEGIN
        update highschooler set grade = null where id=new.id;
     END
|
CREATE TRIGGER q2 after insert ON highschooler
     FOR EACH ROW when new.grade is null
     BEGIN
        update highschooler set grade = 9 where id =new.id;
     END
```

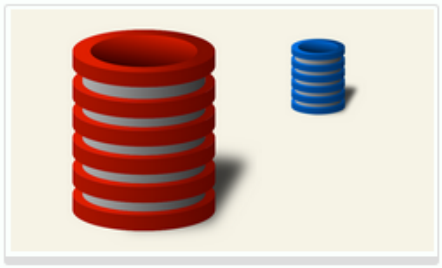
Question

To check your trigger(s), we first ran the following data modification statement(s): *insert into Highschooler values (2121, 'Caitlin', null);*
insert into Highschooler values (2122, 'Don', null);
insert into Highschooler values (2123, 'Elaine', 7);
insert into Highschooler values (2124, 'Frank', 20);
insert into Highschooler values (2125, 'Gale', 10)
.

We then ran the following query: *select * from Highschooler order by ID*

1025	John	12
1101	Haley	10
1247	Alexis	11
1304	Jordan	12
1316	Austin	11
1381	Tiffany	9
1468	Kris	10
1501	Jessica	11
1510	Jordan	9
1641	Brittany	10
1661	Logan	12
1689	Gabriel	9
1709	Cassandra	9
1782	Andrew	10
1911	Gabriel	11
1934	Kyle	12
2121	Caitlin	9
2122	Don	9
2123	Elaine	<NULL>
2124	Frank	<NULL>
2125	Gale	10

Views:
Instead-Of Triggers



Databases

Course Started - Jun 09, 2014 at 15:00 UTC

DB12 Views and Authorization

Your final grade: **90%**.

Download Your Statement (PDF)

View Course

Email Settings *Unenroll*

Question

You've started a new movie-rating website, and you've been collecting data on reviewers' ratings of various movies. Here's the schema:

Movie (mID, title, year, director)
English: There is a movie with ID number *mID*, a *title*, a release *year*, and a *director*.

Reviewer (rID, name)
English: The reviewer with ID number *rID* has a certain *name*.

Rating (rID, mID, stars, ratingDate)
English: The reviewer *rID* gave the movie *mID* a number of *stars* rating (1-5) on a certain *ratingDate*.

Question

In addition to the base tables, you've created three views:

View **LateRating** contains movie ratings after January 20, 2011. The view contains the movie ID, movie title, number of stars, and rating date.

```
create view LateRating as
  select distinct R.mID, title, stars, ratingDate
  from Rating R, Movie M
  where R.mID = M.mID
  and ratingDate > '2011-01-20'
```

View **HighlyRated** contains movies with at least one rating above 3 stars. The view contains the movie ID and movie title.

```
create view HighlyRated as
  select mID, title
  from Movie
  where mID in (select mID from Rating where stars > 3)
```

View **NoRating** contains movies with no ratings in the database. The view contains the movie ID and movie title.

```
create view NoRating as
  select mID, title
  from Movie
  where mID not in (select mID from Rating)
```

Question

Q1 (1/1 point)

Write an instead-of trigger that enables updates to the title attribute of view **LateRating**.

Policy: Updates to attribute title in LateRating should update Movie.title for the corresponding movie. (You may assume attribute mID is a key for table Movie.) Make sure the mID attribute of view LateRating has not also been updated -- if it has been updated, don't make any changes. Don't worry about updates to stars or ratingDate.

- Remember you need to use an instead-of trigger for view modifications as supported by SQLite.

Question

- Trigger Condition: Write an instead-of trigger that enables updates to the stars attribute of view **LateRating**.
  - Updates to attribute stars in LateRating should update Rating.stars for the corresponding movie rating.
    - You may assume attributes [mID,ratingDate] together are a key for table Rating.
  - Make sure the mID and ratingDate attributes of view LateRating have not also been updated
    - if either one has been updated, don't make any changes.
    - Don't worry about updates to title.

Question

- Trigger: Write an instead-of trigger that enables updates to the stars attribute of view **LateRating**.
  - Instead Of Update on View
  - New Update Query on base tables

```
create trigger t1 instead of update of stars on LateRating
begin
  update _______ set _____ = ______  where ___________________
end
```

Question

Q1 (1/1 point)

Write an instead-of trigger that enables updates to the title attribute of view **LateRating**.

Policy: Updates to attribute title in LateRating should update Movie.title for the corresponding movie. (You may assume attribute mID is a key for table Movie.) Make sure the mID attribute of view LateRating has not also been updated -- if it has been updated, don't make any changes. Don't worry about updates to stars or ratingDate.

- Remember you need to use an instead-of trigger for view modifications as supported by SQLite.

```
1 create trigger t instead of update of title on laterating
2 begin
3    update movie set title = new.title where mid = new.mid;
4 end
```

Question

To check your trigger, we first ran the following data modification statement(s): *insert into NoRating values (104, 'E.T.'); insert into NoRating values (110, 'Avatar').*

We then queried the view: *select * from NoRating*

View Result:

102	Star Wars
104	E.T.
105	Titanic

We then ran the following query: *select * from Rating order by mID*

Your Query Result:

201	101	2	2011-01-22
201	101	4	2011-01-27
202	106	4	<NULL>
203	103	2	2011-01-20
203	108	2	2011-01-30
203	108	4	2011-01-12
204	101	3	2011-01-09
205	103	3	2011-01-27
205	108	4	<NULL>
206	106	5	2011-01-19
206	107	3	2011-01-15
207	107	5	2011-01-20