Shedhol7

Mysql

A → mysql -u root -p password: parul

A → show databases

→ names are case-sensitive

Create → create table chocolate
(name char(10),
cost int
);

* check can't be used as column name.

* '0000-00-00'
↑
Invalid date datatype.

Alter // Add columns → alter table chocolate add (weight int);

Alter // Modify datatype → alter table Movie modify mID int(5);

Delete → Delete from Movie where title = 'op';

AVG → select avg (mID) from Movie;

→ COUNT() AVG() SUM()

LIKE → a% => starts with a
_a% => second letter a

Date Data Type → Date YYYY-MM-DD

Default → alter table Movie alter column date set default '1997-05-08';

Delete Table → drop tables Persons;

Drop Column → Alter table Movie drop weight;

Setting Primary Key → Alter Table Movie Add Primary Key (mID);

• Setting foreign key → Alter table rating add foreign key (mID) References Movie(MId);

• Extract date → extract (year from rating.date)

16/Oct/2017 LAB Assignment #5

Highschooler
(ID, name, grade)

Friend (ID1, ID2)

Likes (ID1, ID2)

Q1.

Name
Jordan
Cassandra
Andrew
Alexis
Jessica

⇒ select name from Highschooler where ID in
(select ID1 from Friend where ID2 in
(select ID from Highschooler where name='Gabriel'));

Q2. John | Haley

⇒ select h1.name, h2.name from Highschooler h1 join Highschooler h2 join Likes where h1.ID = Likes.ID1 and h2.ID = Likes.ID2 and h1.grade - h2.grade >= 2;

03

name1	grade1	name2	grade2
Cassandra	9	Gabriel	9
Jessica	11	Kyle	12

⇒ select h1.name as name1, h1.grade as grade1, h2.name as name2, h2.grade as grade2 from Highschooler h1 join Highschooler h2 join (select * from likes where ID1 in (select ID2 from likes) and ID2 in (select ID1 from likes)) as p where h1.ID=p.ID1 and h2.ID=p.ID2 and h1.name < h2.name;

04.

name	grade
Jordan	9
Tiffany	9
Logan	12

⇒ select name, grade from Highschooler where ID not in (select ID1 from likes) and ID not in (select ID from likes);

05.

name	grade	name	grade
John	12	Haley	10
Brittany	10	Kris	10
Alexis	11	Kris	10
Austin	11	Jordan	12

⇒ select h1.name, h1.grade, h2.name, h2.grade from Highschooler h1, Highschooler h2, Likes where Likes.ID2 not in (select ID1 from likes) and Likes.ID1=h1.ID and Likes.ID2 = h2.ID;

2/Nov/2017

Constraints and Triggers
- For relational databases
- SQL standard; systems vary considerably

(Integrity) Constraints (Static concept)
constrain allowable database states

Triggers (Dynamic concept)
monitor database changes, check conditions and initiate actions

Integrity constraints
Impose restrictions on allowable data, beyond those imposed by structure and types.
Examples:
$0.0 < GPA \leq 10.0$
enrollment < 50000
decision : 'y' 'N' null
major = 'CS' ⇒ decision = null
SizeHS < 200 ⇒ not admitted

Why use them?
Data-entry errors (inserts)
Correctness Criteria (updates)
Enforce consistency
Tell system about data - store, query process

Classification
- Non-null
- Key
- Tuple-based
- Referential Integrity (foreign Key)
- Attribute-based
- General Assertions

Declaring and enforcing constraints

Declaration

- With original schema - checked after bulk loading.
  error raised if constraint ~~not fulf~~ fullfilled
- Or later → checked on current database.

Enforcement

- check after every ~~modification~~ transaction
  dangerous.
- deferred constraint checking

Triggers

"Event-Condition-Action-Rules"
When event occurs, check condition; if true, do action

Examples
enrollment > 35000 → reject all applicants
GPA > 3.95 → accepts automatically

Why use them?
Move logic from apps into DBMS.
To enforce constraints
- expressiveness
- constraint "repair" logic

Triggers in SQL
" Event - Condition - Action - Rules "

Create Trigger Name
Before | After | Instead of events
[referencing - variables]
[for each Row]
When (condition)
action

Integrity constraints
Impose restriction on allowable data, beyond those imposed by structure and types.

- Non-null ~~constraints~~ constraints
- Key constraints
- Attribute-based and tuple-based constraints
- General Assertions

Triggers

Triggers
ers Quiz
Triggers Exercises

Bookmark

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:

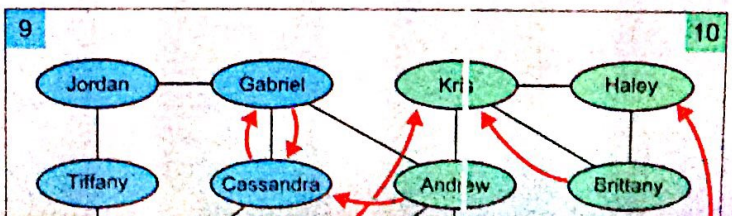
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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.



You may perform these exercises as many times as you like, so we strong y encourage you to keep working with them until you complete the exercises with full credit.

Q1

(1/1 point)

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

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

```
1 create trigger t1
2 after insert on Highschooler
3 for each row
4 when new.name="Friendly"
5 begin
6 insert into Likes select new.ID,ID from Highschooler where id<>new.ID and grade=new.grade;
7 end;
```

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification stat ment(s): *insert into Highschooler values (1000,*

kyle	12	jessica	11

SUBMIT RESET

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 t1
after insert on Highschooler
for each row
when NEW.grade>12 or NEW.grade<9
begin
update Highschooler
set grade=NULL
where ID=NEW.ID;
end
|
create trigger t2
after insert on Highschooler
for each row
when NEW.grade is NULL
begin
```

Correct

Correct

Trigger command(s) were executed.

SUBMIT RESET

Q3

(1/1 point)

Write one or more triggers to maintain symmetry in friend relationships. Specifically, if (A,B) is deleted from Friend, then (B,A) should be deleted too. If (A,B) is inserted into Friend then (B,A) should be inserted too. Don't worry about updates to the Friend table.

- 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 t1
after insert on Friend
begin
insert into Friend select new.ID2,new.ID1;
end
|
create trigger t2
after delete on Friend
for each row
begin
delete from Friend where ID1=OLD.ID2 and ID2=OLD.ID1;
end;
```

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *delete from Friend where ID1 = 1641 and ID2 = 1468;*

Tiffany	9	Alexis	11
Tiffany	9	Jordan	9

SUBMIT RESET

Q4

(1/1 point)

Write a trigger that automatically deletes students when they graduate, i.e., when their grade is updated to exceed 12.

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

```
1 create trigger t1
2 after update on Highschooler
3 for each row
4 when new.grade>12
5 begin
6 delete from Highschooler where ID=new.ID;
7 end;
```

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *update Highschooler set grade = grade + 1*

1381	Tiffany	9

SUBMIT RESET

Q5

(1/1 point)

Write a trigger that automatically deletes students when they graduate, i.e., when their grade is updated to exceed 12 (same as Question 4). In addition, write a trigger so when a student is moved ahead one grade, then so are all of his or her friends.

- 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 t1
after update on Highschooler
for each row
when new.grade>12
begin
delete from Highschooler where ID=new.ID;
end
|
create trigger t2
after update on highschooler
for each row
when new.grade=old.grade+1
begin
update Highschooler set grade=grade+1 where ID in (select ID2 from Friend where ID1=new.ID);
end;
```

Correct

Correct

Trigger command(s) were executed.

SUBMIT RESET

Q6

(1/1 point)

Write a trigger to enforce the following behavior: If A liked B but is updated to A liking C instead, and B and C were friends, make B and C no longer friends. Don't forget to delete the friendship in both directions, and make sure the trigger only runs when the "liked" (ID2) person is changed but the "liking" (ID1) person is not changed.

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

```
1 create trigger t1
2 after update on Likes
3 for each row
4 when new.ID1=old.ID1 and new.ID2!=old.ID2
5 begin
6 delete from Friend where ID1=new.ID2 and ID2=old.ID2;
7 delete from Friend where ID2=new.ID2 and ID1=old.ID2;
8 end;
9
10
```

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *update Likes set ID2 = 1501 where ID1 = 1911, update Likes set ID2 = 1316 where ID1 = 1501, update Likes set ID2 = 1304 where ID1 = 1934, update Likes set ID1 = 1061*

6/Nov/2017

1) db2 Start
2) db2 -td @
3) list database directory @ list tables @
4) Connect to R
5) describe Table table name @

CREATE TRIGGER trig
BEFORE / AFTER INSERT / UPDATE / DELETE
ON table

How to import files in db2

db2start
db2 -tvf path / social.sql
db2 -td@

Information and Database Management Systems (IDBMS)
Lab Assignment #7

Consider the social networking database used in Assignment 5 and perform the fo lowing:

1. Write a trigger that makes new students named 'Friendly' automatically lik e 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'.
2. Write one or more triggers to manage the grade attribute of new Highs hoolers. If the inserted tuple has a value less than 9 or greater than 12, change the valu to NULL. On the other hand, if the inserted tuple has a null value for grade, change it to).
3. Write one or more triggers to manage the grade attribute of new Highs hoolers. If the inserted tuple has a value less than 9 or greater than 12, change the valu to NULL. On the other hand, if the inserted tuple has a null value for grade, change it to).
4. Write a trigger that automatically deletes students when they graduate, .e., when their grade is updated to exceed 12.
5. Write a trigger that automatically deletes students when they graduate, .e., when their grade is updated to exceed 12 (same as Question 4). In addition, write a tr gger so when a student is moved ahead one grade, then so are all of his or her friends.

```
CREATE TRIGGER FRIENDLY AFTER INSERT ON HIGHSCHOOLER REFERENCING NEW AS N FOR EACH ROW WHEN
(N.NAME='Friendly') BEGIN INSERT INTO LIKES SELECT N.ID,ID FROM HIGHSCHOOLER WHERE N.GRADE=GRADE; END @


CREATE TRIGGER GRADECHANGE AFTER INSERT ON HIGHSCHOOLER REFERENCING NEW AS N FOR EACH ROW WHEN (N.GRADE
IS NULL) BEGIN UPDATE HIGHSCHOOLER SET GRADE=9 WHERE ID=N.ID; END @


CREATE TRIGGER GRADECHANGE2 AFTER INSERT ON HIGHSCHOOLER REFERENCING NEW AS N FOR EACH ROW WHEN
(N.GRADE<9 OR N.GRADE>12) BEGIN UPDATE HIGHSCHOOLER SET GRADE=NULL WHERE ID=N.ID; END @


CREATE TRIGGER FAREWELL AFTER UPDATE ON HIGHSCHOOLER REFERENCING NEW AS N FOR EACH ROW WHEN (N.GRADE>12)
BEGIN DELETE FROM HIGHSCHOOLER WHERE ID=N.ID; END @


CREATE TRIGGER PASS AFTER UPDATE ON HIGHSCHOOLER REFERENCING NEW AS N OLD AS O FOR EACH ROW WHEN
(N.GRADE=O.GRADE+1) BEGIN UPDATE HIGHSCHOOLER SET GRADE=GRADE+1 FROM FRIEND WHERE ID1=N.ID AND ID2=ID ; END
@
```

Information and Database Management Systems (IDBMS)
Lab Assignment #6

Consider the modernized version of Microsoft's *Northwind* database given in Assignment 3. Perform the following:

1. Create a view "ProductSupplier" that shows the product names along with their supplier names.
2. Using the view "ProductSupplier", list the names of suppliers who supply the product names start with 'C'.
3. Create a view called "TotalOrders" that shows the total number of orders placed and total amount spent by each customer after his or her name.
4. Using the view "TotalOrders", list the names of customers who spend more than the average expenditure placed by all customers.
5. Create a view "SupplierCountry" that shows the number of suppliers in each country.
6. Create a view "Discount" that shows the supplier names, product names supplied by them and the discount amount given on these products.
7. Using the view "Discount", find the supplier name who provide the second highest discount.

19/Oct/2017

Assignment-6

Sr Seram Nahar

~~Customer~~ CUSTOMER

id*	FirstName	LastName	City	Country	Phone

~~OrderItem~~ ORDERITEM

id*	OrderId	ProductId	UnitPrice	Quantity

~~Order~~ ORDERS

id*	OrderDate	OrderNumber	CustomerId	TotalAmount

~~Product~~ PRODUCT

id*	ProductName	SupplierId	UnitPrice	Package	IsDiscontinued

SUPPLIER

id*	CompanyName	ContactTitle	City	Country	Phone	Fax

Q1.

```
create view ProductSupplier AS
Select s.CompanyName, p.ProductName
From PRODUCT p, SUPPLIER s, ORDERITEM o
Where p.id = o.OrderId and s.id = o.ProductId;
```

Q2

```
Select *
From ProductSupplier
Where ProductName
like 'C%.';
```

13.
```
Create view TotalOrders AS
select concat(c.FirstName, " ", c.LastName),
       Sum(o.TotalAmount) as TotalAmount, count(o.Customerid) as
from CUSTOMER c, ORDERS o                          | NoOrders
Where o.Customerid=c.id
Group by o.Customerid;
```

14.
```
select /*/ from TotalOrders
With
  Select *
  From TotalOrders
  where TotalAmount > (select avg(TotalAmount)
                       from TotalOrders);
```

15.
```
Create view SupplierCountry as
Select Country, count(Country)
from SUPPLIER
Group by Country;
```

16.
```
create view Discount As
select p.UnitPrice-o.UnitPrice as discount,
       p.ProductName, S.CompanyName
From SUPPLIER s, ORDERITEM o, PRODUCT P
where s.id=p.Supplierid and p.id=o.Productid;
```

Q7.
```
select distinct CompanyName
from Discount
where discount = (select max(discount) from
                  from Discount
                  where discount < (select max(discount)
                                    from Discount )
                  );
```