Creazione tabella Profilo

CREATE TABLE `Q`.`profilo` (

`idprofilo` VARCHAR(45) NOT NULL,

`pubblico` TINYINT(1) NULL,

PRIMARY KEY (`idprofilo`));

Creazione tabella

CREATE TABLE `Q`.`utente` (

`username` VARCHAR(45) NOT NULL,

`email` VARCHAR(45) NULL,

`password` VARCHAR(45) NULL,

`nome` VARCHAR(45) NULL,

`cognome` VARCHAR(45) NULL,

`dataNascita` DATE NULL,

`idProfilo` VARCHAR(45) NULL,

PRIMARY KEY (`username`),

INDEX `idProfilo\_fk\_idx` (`idProfilo` ASC) VISIBLE,

CONSTRAINT `idProfilo\_fk`

FOREIGN KEY (`idProfilo`)

REFERENCES `Q`.`profilo` (`idprofilo`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

DROP TRIGGER IF EXISTS `Q`.`utente\_BEFORE\_INSERT`;

TRIGGER PER LUNGHEZZA PASSWORD

DELIMITER $$

USE `Q`$$

CREATE DEFINER=`root`@`localhost` TRIGGER `utente\_BEFORE\_INSERT` BEFORE INSERT ON `utente` FOR EACH ROW BEGIN

declare minPwdLength int unsigned;

declare maxPwdLength int unsigned;

declare pwdLength int unsigned;

set minPwdLength = 8;

set maxPwdLength = 16;

set pwdLength = (select length(new.password));

if (pwdLength < minPwdLength) then

signal sqlstate '45000'

set message\_text = 'Password breve (meno di 8 caratteri).';

end if;

if (pwdLength > maxPwdLength) then

signal sqlstate '45000'

set message\_text = 'Password lunga (più di 16 caratteri).';

end if;

END$$

DELIMITER ;

Creazione tabella AMICO

CREATE TABLE `Q`.`amico` (

`username1` VARCHAR(45) NOT NULL,

`username2` VARCHAR(45) NOT NULL,

`DataInizioAmicizia` DATE NULL,

PRIMARY KEY (`username1`, `username2`),

INDEX `username2\_fk\_idx` (`username2` ASC) VISIBLE,

CONSTRAINT `username1\_fk`

FOREIGN KEY (`username1`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `username2\_fk`

FOREIGN KEY (`username2`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

DROP TRIGGER IF EXISTS `Q`.`Amico\_BEFORE\_INSERT`;

DELIMITER $$

USE `Q`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `Q`.`Amico\_BEFORE\_INSERT` BEFORE INSERT ON `Amico` FOR EACH ROW

BEGIN

if (new.username1 = new.username2) then

signal sqlstate '45000'

set message\_text = 'Stesso username.';

end if;

SE NOI STIAMO CERCANDO DI INSERIRE B-A ED ESISTE B-A DEVE DARE ERRORE

if (exists ( select \* from amico a where new.username1 = a.username2 and new.username2 = a.username1) ) then

signal sqlstate '45000'

set message\_text = 'Coppia già presente.';

end if;

END$$

DELIMITER ;

CREATE TABLE `Q`.`Segue` (

`username1` VARCHAR(45) NOT NULL,

`username2` VARCHAR(45) NOT NULL,

`DataInizioFollow` DATE NULL,

PRIMARY KEY (`username1`, `username2`),

INDEX `usernamefollow2\_fk\_idx` (`username2` ASC) VISIBLE,

CONSTRAINT `usernamefollow1\_fk`

FOREIGN KEY (`username1`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `usernamefollow2\_fk`

FOREIGN KEY (`username2`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

DROP TRIGGER IF EXISTS `Q`.`Segue\_BEFORE\_INSERT`;

DELIMITER $$

USE `Q`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `Q`.`Segue\_BEFORE\_INSERT` BEFORE INSERT ON `Segue` FOR EACH ROW

BEGIN

if (new.username1 = new.username2) then

signal sqlstate '45000'

set message\_text = 'Stesso username.';

end if;

END$$

DELIMITER ;

CREATE TABLE `Q`.`post` (

`usernameutente` VARCHAR(45) NOT NULL,

`Data` DATE NOT NULL,

`Orario` TIME NOT NULL,

`testo` VARCHAR(45) NULL,

PRIMARY KEY (`usernameutente`, `Data`, `Orario`),

CONSTRAINT `usernameutentepost\_fk`

FOREIGN KEY (`usernameutente`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

CREATE TABLE `Q`.`sitrova` (

`usernameutentepost` VARCHAR(45) NOT NULL,

`DataPost` DATE NOT NULL,

`OrarioPost` TIME NOT NULL,

`idProfilo` VARCHAR(45) NOT NULL,

PRIMARY KEY (`usernameutentepost`, `DataPost`, `OrarioPost`, `idProfilo`),

INDEX `idProfilo\_fk\_idx` (`idProfilo` ASC) VISIBLE,

CONSTRAINT `post\_fk`

FOREIGN KEY (`usernameutentepost` , `DataPost` , `OrarioPost`)

REFERENCES `Q`.`post` (`usernameutente` , `Data` , `Orario`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `idProfiloSitrova\_fk`

FOREIGN KEY (`idProfilo`)

REFERENCES `Q`.`profilo` (`idprofilo`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

CREATE TABLE `Q`.`tag` (

`usernameUtentePost` VARCHAR(45) NOT NULL,

`DataPost` DATE NOT NULL,

`OrarioPost` TIME NOT NULL,

`usernameUtenteTaggato` VARCHAR(45) NOT NULL,

`Data` DATE NULL,

`Orario` TIME NULL,

PRIMARY KEY (`usernameUtentePost`, `DataPost`, `OrarioPost`, `usernameUtenteTaggato`),

INDEX `usernameUtenteTaggato\_idx` (`usernameUtenteTaggato` ASC) VISIBLE,

CONSTRAINT `Post\_tag\_fk`

FOREIGN KEY (`usernameUtentePost` , `DataPost` , `OrarioPost`)

REFERENCES `Q`.`post` (`usernameutente` , `Data` , `Orario`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `usernameUtenteTaggato`

FOREIGN KEY (`usernameUtenteTaggato`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

CREATE TABLE `Q`.`commento` (

`usernameUtentePost` VARCHAR(45) NOT NULL,

`dataPost` DATE NOT NULL,

`orarioPost` TIME NOT NULL,

`usernameUtenteCommentante` VARCHAR(45) NOT NULL,

`data` DATE NOT NULL,

`orario` TIME NOT NULL,

`testo` VARCHAR(1000) NULL,

PRIMARY KEY (`usernameUtentePost`, `dataPost`, `orarioPost`, `usernameUtenteCommentante`, `data`, `orario`),

CONSTRAINT `postCommento\_fk`

FOREIGN KEY (`usernameUtentePost` , `dataPost` , `orarioPost`)

REFERENCES `Q`.`post` (`usernameUtente` , `data` , `orario`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `usernameUtenteCommentante\_fk`

FOREIGN KEY (`usernameUtentePost`)

REFERENCES `q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

CREATE TABLE `Q`.`messaggio` (

`usernameMittente` VARCHAR(45) NOT NULL,

`usernameDestinatario` VARCHAR(45) NOT NULL,

`Data` DATE NOT NULL,

`Orario` TIME NOT NULL,

`testo` VARCHAR(1000) NULL,

`Letto` TINYINT(1) NULL,

PRIMARY KEY (`usernameMittente`, `usernameDestinatario`, `Data`, `Orario`),

INDEX `usernameDestinatario\_fk\_idx` (`usernameDestinatario` ASC) VISIBLE,

CONSTRAINT `usernameMittente\_fk`

FOREIGN KEY (`usernameMittente`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `usernameDestinatario\_fk`

FOREIGN KEY (`usernameDestinatario`)

REFERENCES `Q`.`utente` (`username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

Rendo pubblico il profilo se l’id è l’id dell’utente tizio.

UPDATE PROFILO SET pubblico=TRUE WHERE idProfilo IN (

SELECT U.idProfilo

FROM UTENTE U

WHERE U.nome='Tizio'

);

2)

Nella prima Colonna di amico sto cercando caio nella seconda gli amici.

Select username2 as usr

From Amico A, Utente U

Where A.username1=U.username and U.nome='Caio'

UNION

Nella seconda Colonna di amico sto cercando caio nella prima gli amici.

Select username1 as usr

From Amico A, Utente U

Where A.username2=U.username and U.nome='Caio';

La prima la utilizzo se mi servono gli username degli amici

Se utilizzo solo una tabella utente ottengo le username degli utenti che sono Caio ed anche amici di caio

Select UA.idProfilo

From Amico A, Utente UA,Utente U

Where A.username2=UA.username and A.username1=U.username and U.nome='Caio'

UNION

Select UA.idProfilo

From Amico A, Utente UA, Utente U

Where A.username1=UA.username and A.username2=U.username and U.nome='Caio';

3)

SELECT NumAmici.idProfilo

FROM (

Prendo l’id profilo della persona pubblica la cui username compare nella prima tabella di amico ( A.username1=U. username) insieme al suo numero di amici

select P.idProfilo,count(\*) as nAmici

from Utente U, Profilo P, Amico A

where A.username1=U.username and U.idProfilo=P.idProfilo and P.pubblico=true

Group by P.idProfilo

UNION all

select P.idProfilo,count(\*) as nAmici

from Utente U, Profilo P, Amico A

where A.username2=U.username and U.idProfilo=P.idProfilo and P.pubblico=true

Group by P.idProfilo

Union all

select P.idProfilo, 0

from Utente U, Profilo P

Se non esiste una tupla in AMICO che ha come username 1 o come username2 la username del profilo fissato

where U.idProfilo=P.idProfilo and P.pubblico = true and not exists.

( select \*

From Amico A

where A.username1=U.username or A.username2=U.username) ) as NumAmici

Infine ottengo una tabella virtuale che è composta da idProfilo e numero amici. In numAmici potrebbero esserci due tuple con lo stesso id Profilo

Id1 4. La prima tupla deriva dal fatto che la username di id1 è presente 4 volte in amico come username1

Id1 3 La seconda tupla deriva dal fatto che la username di id1 è presente 3 volte in amico come username2

group by NumAmici.idProfilo

having sum(NumAmici.namici) <50;

4)

Numero commenti dei post dei profili pubblici

create view numCommenti (usernameUtentePost,dataPost,orarioPost,numerocommenti) as

select usernameUtentePost,dataPost,orarioPost,count(\*) as numerocommenti

From Commento C, Utente U, Profilo P

Where C.usernameUtentePost=U.username and U.idProfilo=P.idProfilo and P.pubblico=True

Group By usernameUtentePost,dataPost,orarioPost

UNION ALL

(Prendo in considerazione I post con 0 commenti sempre dei profili pub)

Select P.usernameUtente as usernameUtentePost , P.data as dataPost , P.orario as orarioPost, 0

Se non esiste un tapla in commento che contiene il post pubblico fissato

from Post P, Utente U, Profilo Pr

where P.usernameUtente=U.username and U.idProfilo=Pr.idProfilo and Pr.pubblico=True and

not exists ( select \*

From Commento C

where C.usernameUtentePost=P.usernameUtente and C.dataPost=P.data and C.orarioPost=P.orario);

un post che è preso dalla vista lo restituisco se ha un numero commmenti = al minimo globale

Select NC.usernameUtentePost, NC.dataPost, NC.orarioPost

From NumCommenti NC

Where NC.numerocommenti = ( Select min(numerocommenti)

From NumCommenti );

DROP VIEW NumCommenti;

5)

Qua prendo per ogni post il numero commenti

create view numCommenti (idProfilo,usernameUtentePost,dataPost,orarioPost,numerocommenti) as

select P.idProfilo, C.usernameUtentePost, C.dataPost, C.orarioPost, count(\*) as numerocommenti

From Commento C, Utente U, Profilo P

Where C.usernameUtentePost=U.username and U.idProfilo=P.idProfilo

Group By usernameUtentePost,dataPost,orarioPost

UNION ALL

Se non esiste un tapla in commento che contiene il post fissato(faccio questo perché è probabile che tutti post di un profilo abbiano 0 commenti)

Select Pr.idProfilo, P.usernameUtente as usernameUtentePost , P.data as dataPost , P.orario as orarioPost, 0

from Post P, Utente U, Profilo Pr

where P.usernameUtente=U.username and U.idProfilo=Pr.idProfilo and

not exists ( select \*

From Commento C

where C.usernameUtentePost=P.usernameUtente and C.dataPost=P.data and C.orarioPost=P.orario);

per ogni profilp trovo il massimo numero commenti trai suoi post

create view ProfiloMaxCommenti(idProfilo, numMaxCommenti) as

select idProfilo, max(maxCommenti)

from NumCommenti

Group By idProfilo;

Per ogni profile prendo il post che ha il massimo numero commenti

select idProfilo, usernameUtentePost,dataPost,orarioPost

from numCommenti NC, ProfiloMaxCommenti PMC

where NC.idProfilo=PMC.idProfilo and NC.numeroCommenti=PMC.maxCommenti;

# NC PMC RIS

# a p1 3 a 4 a p2 4

# a p2 4 b 7 b p3 7

# a p5 4 a p5 4

# b p3 7

# b p4 0

drop view numCommenti drop view ProfiloMaxCommenti;

6)

CREATE DEFINER = CURRENT\_USER TRIGGER `Q`.`commento\_BEFORE\_INSERT` BEFORE INSERT ON `commento` FOR EACH ROW

BEGIN

(se l’utente del post= utente post commentante è possibile perché è la stessa persona)

Mi chiedo se utente del post è divrerso dall’utente commentante AND se l’utente del post ha il profilo privato AND verifico se infine l’utentecommente e l’utente del post non siano amici

if (new.usernameUtentePost<>new.usernameUtenteCommentante

and false=(select pubblico from profilo p, utente u where u.idProfilo=p.idProfilo and u.username=New.UsernameUtentePost)

and new.usernameUtenteCommentante not in (select username1 from Amico where username2=new.usernameUtentePost)

and new.usernameUtenteCommentante not in (select username2 from Amico where username1=new.usernameUtentePost) ) then

signal sqlstate '45000'

set message\_text = 'Non sono amici.';

end if;

END

7)

Seleziono id profilo e post

DELIMITER $$

CREATE PROCEDURE `q`.`postAnniPrecedenti` ()

BEGIN

select pr.idProfilo, p.\*

from profilo pr, utente u, post p

where pr.idProfilo=u.idProfilo and u.username=p.usernameUtentePost and

day(now())=day(p.data) and month(now()) = month(p.data) and year(now()) > year(p.data);

END $$

DELIMITER ;

CREATE EVENT `postPassatiGiornalieri`

ON SCHEDULE EVERY 1 DAY

STARTS '2020-08-07 00:00:00'

DO CALL postAnniPrecedenti();