**Vincolo emailFormat**

-- Implementazione del vincolo emailFormat

ALTER TABLE Email ADD CONSTRAINT emailFormat CHECK ( email LIKE '\_%@\_%.\_\_%')

**Vincoli passwordLen e usernameLen**

-- Implementazione dei vincoli passwordLen e usernameLen

ALTER TABLE R\_User

ADD CONSTRAINT passwordLen CHECK (LENGTH(password)>7),

ADD CONSTRAINT usernameLen CHECK (LENGTH(nickname)>2)

**Vincolo uniqueMainAddress**

-- Implementazione del vincolo uniqueMainAddress

CREATE FUNCTION uniqueMainAddress()

RETURNS TRIGGER AS $uniqueMainAddress$

BEGIN

IF NEW.main = true AND NEW.contactID IN (SELECT contactID FROM AssignedAddress WHERE main = true) THEN

RAISE EXCEPTION 'A main e-mail for this contact already exists!';

END IF;

RETURN NEW;

END;

$uniqueMainAddress$ LANGUAGE plpgsql;

CREATE TRIGGER uniqueMainAddress

BEFORE INSERT OR UPDATE ON AssignedAddress

FOR EACH ROW

EXECUTE PROCEDURE uniqueMainAddress();

**Vincolo checkContactNumbers**

-- Implementazione del vincolo checkContactNumbers

CREATE FUNCTION checkContactNumbers()

RETURNS TRIGGER AS $checkContactNumbers$

DECLARE

ConID integer := OLD.contactID;

BEGIN

IF NOT EXISTS (SELECT \*

FROM AssignedPhone as AP, PhoneNumber as PN

WHERE AP.phoneNumber = PN.phoneNumber and ConID = AP.contactID and PN.phoneType = 'MOBILE')

OR NOT EXISTS (SELECT \*

FROM AssignedPhone as AP, PhoneNumber as PN

WHERE AP.phoneNumber = PN.phoneNumber and ConID = AP.contactID and PN.phoneType = 'LANDLINE')

THEN

RAISE EXCEPTION 'A contact must have at least a landline number and a mobile number';

END IF;

RETURN NEW;

END;

$checkContactNumbers$ LANGUAGE plpgsql;

CREATE TRIGGER checkContactNumbers

AFTER DELETE OR UPDATE ON AssignedPhone

FOR EACH ROW

EXECUTE PROCEDURE checkContactNumbers();

**Vincolo distinctEmail**

-- Implementazione del vincolo distinctEmail

ALTER TABLE Email

ADD CONSTRAINT distinctEmail UNIQUE (email, contactID)

**Vincolo checkCallType**

-- Implementazione del vincolo checkCallType

ALTER TABLE PhoneCall

ADD CONSTRAINT checkCallType CHECK (callType in ('SENT', 'ENTERED', 'MISSED'))

**Trigger AutoDeleteGroup**

CREATE FUNCTION AutoDeleteGroup()

RETURNS TRIGGER AS $AutoDeleteGroup$

BEGIN

IF NOT EXISTS (SELECT \* FROM Participant as Par WHERE OLD.groupID = Par.groupID)

THEN

DELETE FROM R\_Group as rg

WHERE OLD.groupID = rg.groupID;

END IF;

RETURN NEW;

END;

$AutoDeleteGroup$ LANGUAGE plpgsql;

CREATE TRIGGER AutoDeleteGroup

AFTER DELETE ON Participant

FOR EACH ROW

EXECUTE PROCEDURE AutoDeleteGroup();

**Trigger AutoDeleteInfo**

**Creare procedura che viene chiamata alla rimozione di un contatto**

CREATE FUNCTION AutoDeletePhone()

RETURNS TRIGGER AS $AutoDeletePhone$

BEGIN

IF NOT EXISTS (SELECT \* FROM AssignedPhone as AP WHERE OLD.phoneNumber = AP.phoneNumber)

THEN

DELETE FROM PhoneNumber as PN

WHERE PN.phoneNumber = OLD.phoneNumber;

END IF;

RETURN NEW;

END;

$AutoDeletePhone$ LANGUAGE plpgsql;

CREATE TRIGGER AutoDeletePhone

AFTER DELETE ON AssignedPhone

FOR EACH ROW

EXECUTE PROCEDURE AutoDeletePhone();

CREATE FUNCTION AutoDeleteAddress()

RETURNS TRIGGER AS $AutoDeleteAddress$

BEGIN

IF NOT EXISTS (SELECT \* FROM AssignedAddress as AA WHERE OLD.addressStr = AA.addressStr and OLD.addressZip = AA.addressZip)

THEN

DELETE FROM Address

WHERE Address.street = OLD.addressStr and Address.zipcode = OLD.addressZip;

END IF;

RETURN NEW;

END;

$AutoDeleteAddress$ LANGUAGE plpgsql;

CREATE TRIGGER AutoDeleteAddress

AFTER DELETE ON AssignedAddress

FOR EACH ROW

EXECUTE PROCEDURE AutoDeleteAddress();

**Trigger SetGroupDate**

CREATE FUNCTION SetGroupDate()

RETURNS TRIGGER AS $SetGroupDate$

BEGIN

UPDATE R\_Group as RG

SET NEW.creationDate = CURRENT\_DATE

WHERE NEW.groupID = RG.groupID;

RETURN NEW;

END;

$SetGroupDate$ LANGUAGE plpgsql;

CREATE TRIGGER SetGroupDate

AFTER INSERT ON R\_Group

FOR EACH ROW

EXECUTE PROCEDURE SetGroupDate();

**Procedura UpdCallContactName**

-- La seguente procedura riceve in ingresso un numero telefonico e un utente, cerca il nome e il cognome dei contatti associati ad essi, e li scrive nella colonna contactName di PhoneCall dove il numero telefonico corrisponde. Se non ci sono contatti associati al numero telefonico, scrive “Unknown”

CREATE PROCEDURE UpdCallContactName(PNumber char(10), usr R\_User.email%type)

LANGUAGE plpgsql AS $UpdCallContactName$

DECLARE

resultString varchar(1024) := '';

namesur CURSOR FOR (SELECT contactName, surname

FROM PhoneNumber as PN, AssignedPhone as AP, Contact as Con

WHERE PN.phoneNumber = AP.phoneNumber and AP.contactID = Con.contactID and PN.phoneNumber = PNumber and Con.r\_user = usr);

BEGIN

FOR nome IN namesur LOOP

resultString := resultString || nome.contactName || ' ' || nome.surname || ', ';

END LOOP;

resultString := rtrim(resultString, ', ');

IF resultString = '' THEN

resultString := 'Unknown';

END IF;

UPDATE PhoneCall as PC

SET contactName = resultString

WHERE PC.phoneNumber = PNumber;

END;

$UpdCallContactName$

-- Chiamata della procedura all’inserimento di una nuova riga nella tabella PhoneCall

CREATE FUNCTION UpdPhoneCall()

RETURNS TRIGGER AS $UpdPhoneCall$

BEGIN

CALL UpdCallContactName(NEW.phoneNumber, NEW.r\_user);

RETURN NEW;

END;

$UpdPhoneCall$ LANGUAGE plpgsql;

CREATE TRIGGER UpdPhoneCall

AFTER INSERT ON PhoneCall

FOR EACH ROW

EXECUTE PROCEDURE UpdPhoneCall();

-- Chiamata della procedura all’assegnazione di un numero di telefono ad un contatto

CREATE FUNCTION UpdCallFromAP()

RETURNS TRIGGER AS $UpdCallFromAP$

DECLARE

currentUser varchar(10);

BEGIN

SELECT Con.r\_user into currentUser

FROM Contact as Con

WHERE NEW.contactID = Con.contactID;

CALL UpdCallContactName(NEW.phoneNumber, currentUser);

RETURN NEW;

END;

$UpdCallFromAP$ LANGUAGE plpgsql;

CREATE TRIGGER UpdCallFromAP

AFTER INSERT or DELETE ON AssignedPhone

FOR EACH ROW

EXECUTE PROCEDURE UpdCallFromAP();