Terza Esercitazione di Basi di Dati

Lorenzo Bonin
Università degli Studi di Trieste

21 aprile 2023

Overview

1. Riassunto dell'esercitazione precedente
2. Transazioni
3. Stored procedures
4. User defined functions
5. Triggers

Database dell'Università

Studenti

Matricola	Nome	Cognome	Codice Fiscale

Professori

Matricola	Nome	Cognome	Codice Fiscale	Settore

Corsi

Codice	Nome	CFU	Professore [Matricola]

Esami

Corso [Codice]	Studente [Matricola]	Data	Voto	Lode

Cosa abbiamo visto fino ad ora

- Creazione e popolamento del DB
- Query sul DB creato
- Prepared statements
- Query su viste

Per partire tutti dallo stesso punto...

File **uni_db.sql**

- Elimina il DB se esiste già
- Ne crea uno nuovo
- Lo popola

File **query1.sql** e **query2.sql**

- Contengono il codice delle precedenti esercitazioni

Ti sei perso l'esercitazione precedente? $\rightarrow$ esegui **uni_db.sql** e **query1.sql**.
Hai già creato il DB la scorsa volta? $\rightarrow$ `USE uni_db;`

Transazione

Task

Scrivere una transazione che assegni al professore meno impegnato l'unico corso scoperto.

Transazione

Task

Scrivere una transazione che assegni al professore meno impegnato l'unico corso scoperto.

- *Hint*: scrivere prima delle query di prova: qual è il corso scoperto? Qual è il professore meno impegnato?

Transazione

Task

Scrivere una transazione che assegni al professore meno impegnato l'unico corso scoperto.

- *Hint*: scrivere prima delle query di prova: qual è il corso scoperto? Qual è il professore meno impegnato?

```
SELECT *
FROM corsi c
WHERE professore IS NULL;
```

Transazione

Qual è il professore meno impegnato?

Transazione

Qual è il professore meno impegnato?

```
SELECT matricola, sum(cfu) as cfu_tot
FROM professori p
INNER JOIN corsi c
ON c.professore = bp.matricola
GROUP BY professore
ORDER BY cfu_tot asc
LIMIT 1;
```

Transazione

```
START TRANSACTION;

SELECT @prof := matricola, sum(cfu) as cfu_tot
FROM professori p
INNER JOIN corsi c
ON c.professore = p.matricola
GROUP BY professore
ORDER BY cfu_tot asc
LIMIT 1;

UPDATE corsi
SET professore = @prof
WHERE professore IS NULL;

COMMIT;
```

Stored procedure 1

Task

Scrivere una stored procedure che restituisca la media aritmetica e la media ponderata di tutti gli studenti.

Stored procedure 1

```
DELIMITER $$
CREATE PROCEDURE CalcoloMedie()
BEGIN
    SELECT s.matricola, s.nome, s.cognome,
    SUM(e.voto * c.cfu)/SUM(c.cfu) as mp,
    AVG(e.voto) as ma
    FROM studenti s INNER JOIN esami e
    ON s.matricola = e.studente
    INNER JOIN corsi c ON e.corso = c.codice
    GROUP BY e.studente;
END $$
DELIMITER ;
```

Stored procedure 2

Task

Scrivere una stored procedure che scriva in una variabile passata il monte di ore di un dato docente. Lanciare un errore se il docente non esiste.

Stored procedure 2

```
DELIMITER $$
CREATE PROCEDURE MonteOre(IN docente INT, OUT ore INT)
BEGIN
    SELECT SUM(cfu * 8)
    INTO ore
    FROM corsi c
    WHERE professore = docente;
    IF ore IS NULL THEN
        SIGNAL SQLSTATE "02000"
        SET MESSAGE_TEXT = "Docente not found!";
    END IF;
END $$
DELIMITER ;
```

User defined function 1

Task

Scrivere una user defined function che restituisca il corso di laurea di un dato studente.

User defined function 1

```
DELIMITER $$
CREATE FUNCTION cdl(matricola char(9))
RETURNS CHAR(4) DETERMINISTIC
BEGIN
    RETURN SUBSTRING(matricola, 1, 4);
END $$
DELIMITER ;
```

User defined function 2

Task

Scrivere una user defined function che restituisca la media ponderata di un dato studente.

User defined function 2

```
DELIMITER $$
CREATE FUNCTION media_ponderata(matricola char(9))
RETURNS float DETERMINISTIC
BEGIN
    DECLARE mp float;
    SELECT SUM(c.cfu * e.voto )/ SUM(c.cfu)
    INTO mp
    FROM esami e INNER JOIN corsi s
    ON e.corso = s.codice
    WHERE e.studente = matricola;
    RETURN(mp);
END $$
DELIMITER ;
```

User defined function 3

Task

Scrivere una user defined function che restituisca il rank di uno studente nel suo corso di laurea in base alla sua media ponderata.

User defined function 3

```
DELIMITER $$
CREATE FUNCTION rank_cdl(matricola char(9))
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE r INT;
    SELECT COUNT(*)
    INTO r
    FROM studenti s
    WHERE cdl(s.matricola) = cdl(matricola) AND
    media_ponderata(s.matricola) >= media_ponderata(
       matricola);
    RETURN(r);
END $$
DELIMITER ;
```

Trigger 1

Task

Scrivere un trigger per tenere traccia delle assunzioni. Supporre che l'inserimento di un docente nel DB coincida con la sua data di assunzione.

Trigger 1

Task

Scrivere un trigger per tenere traccia delle assunzioni. Supporre che l'inserimento di un docente nel DB coincida con la sua data di assunzione.

- *Hint*: creare prima una tabella per registrare le assunzioni.

Trigger 1

Task

Scrivere un trigger per tenere traccia delle assunzioni. Supporre che l'inserimento di un docente nel DB coincida con la sua data di assunzione.

- *Hint*: creare prima una tabella per registrare le assunzioni.

```
CREATE TABLE assunzioni (
    matricola INT (4) PRIMARY KEY ,
    data_assunzione DATE
);
```

Trigger 1

```
DELIMITER $$
CREATE TRIGGER trg_data_assunzione
AFTER INSERT ON professori
FOR EACH ROW BEGIN
    INSERT INTO assunzioni VALUES(matricola, CURDATE());
END $$
DELIMITER ;
```

Trigger 2

Task

Scrivere un trigger che, nel momento in cui viene inserito un corso senza un professore assegnato, lo assegna ad un qualsiasi professore che non ha corsi.

Trigger 2

```
CREATE TRIGGER trg_corso_scoperto
BEFORE INSERT ON corsi
FOR EACH ROW BEGIN
    IF NEW.professore IS NULL THEN
        SELECT matricola INTO @prof
        FROM professori
        WHERE matricola NOT IN(
            SELECT DISTINCT professore
            FROM corsi
            WHERE professore IS NOT NULL
        )
        LIMIT 1;
        SET NEW.professore = @prof;
    END IF;
END $$
```