UNIVERSITÀ DEGLI STUDI DI TRIESTE

ARTIFICIAL INTELLIGENCE & DATA ANALYTICS

Terza esercitazione di Basi di Dati

Università degli studi di Trieste

Giovanni **Pinna**

07 maggio 2024

Contatti

GIOVANNI.PINNA@phd.units.it

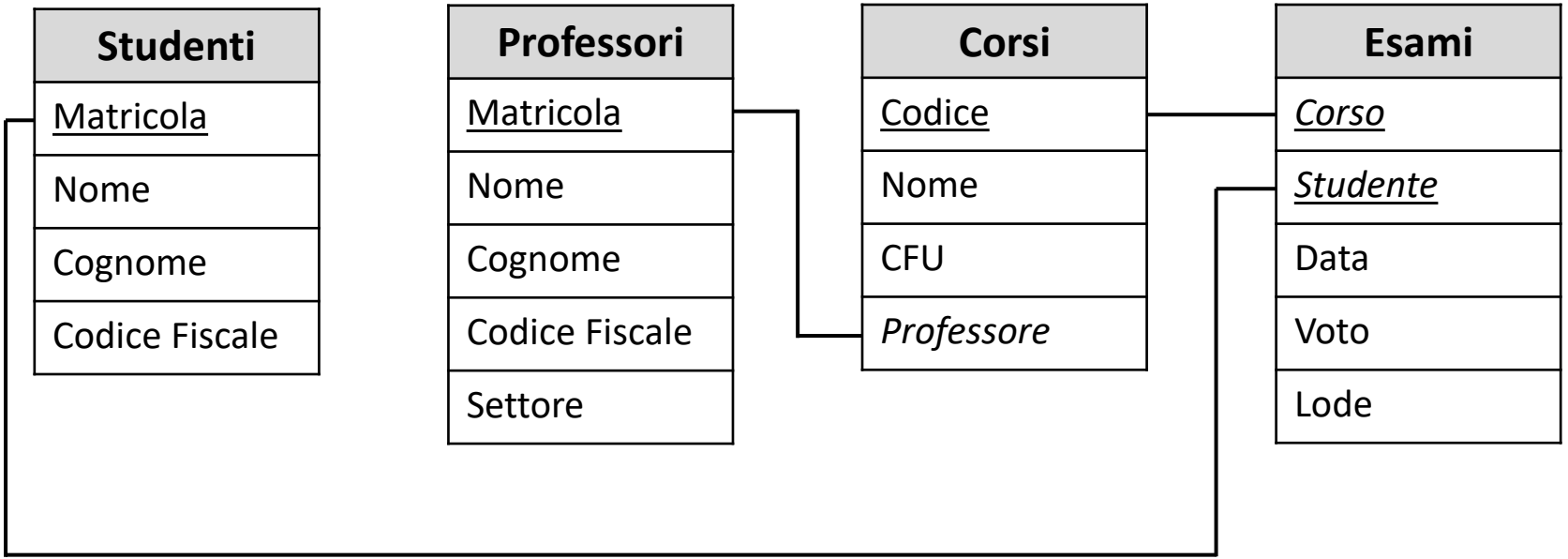
Edificio: C3, Ufficio: C3.232
(contattatemi prima via mail)

https://github.com/giovannipinna96/DB_2024_Exercises

Sommario

1. Riassunto della seconda esercitazione
2. Transazioni
3. Stored procedures
4. User defined function
5. Triggers

Database (molto semplice) dell'Università



Cosa abbiamo visto nelle precedenti esercitazioni

1. Creazione del DB
2. Popolamento del DB
3. Prepared statements
4. Query su viste

Per partire tutti dallo stesso punto

File ***uni_db.sql****:*
- Elimina il DB se esiste già
- Crea un nuovo DB
- Lo popola

File ***query_prima_esercitazione.sql****:*
- Contiene il codice per modificare il DB (colonna *«Genere»*)
- Contiene le query della prima esercitazione

File ***query_seconda_esercitazione.sql****:*
- Contiene le query della seconda esercitazione
- Contiene le query si viste della seconda esercitazione

Per partire tutti dallo stesso punto

Se ti sei perso l'esercitazione o le esercitazioni precedenti?

- esegui **uni_db.sql**
  + query_prima_esercitazione.sql
  + query_seconda_esercitazione.sql

Se hai già creato il DB la volta scorsa

- USE uni_db;

Files della lezione di oggi

Nella cartella *lez_3* della repo Github troverete i seguenti file:

- ***query_terza_esercitazione.sql****:* contiene tutto quello che faremo oggi.
- ***transazione.sql:*** contiene il punto di partenza per l'esercizio sulle transazioni.
- ***udf.sql:*** contiene il punto di partenza per l'esercizio sulle UDF.
- ***trigger.sql:*** contiene il punto di partenza per l'esercizio sul trigger.

Per scrivere codice SQL

Per chi non ha MySQL o MySQL Workbench…
Esistono questi editor online:

- https://onecompiler.com/studio
- https://www.jdoodle.com/execute-sql-online/
- …

Svantaggi:

- potrebbe non supportare alcune operazioni
- è volatile, ovvero non ricorda quanto eseguito in precedenza (è necessario creare, popolare ed interrogare in un’unica esecuzione!)

Morale: può andare bene come soluzione temporanea.

Transazione

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

Transazione

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

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

Transazione

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

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

```sql
-- Qual è il corso scoperto
SELECT *
FROM corsi c
WHERE professore IS NULL;
```

Transazione

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

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

```sql
-- Qual è il professore meno impegnato
SELECT 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;
```

Transazione

```sql
-- Creazione della transizione
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 Procedures

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 Procedures

```sql
-- Stored procedure che scriva in una variabile passata il monte ore
-- di un dato docente. Lanciare un errore se il docente non esiste.
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

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

User defined function 1

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

```
1  -- Defined function che restituisca il corso di laurea di un dato studente.
2  DELIMITER $$
3  CREATE FUNCTION cdl(matricola CHAR(9))
4  RETURNS CHAR(4) DETERMINISTIC
5  BEGIN
6      RETURN SUBSTRING(matricola, 1, 4);
7  END $$
8  DELIMITER;
```

User defined function 2

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 2 (help function)

Defined function che restituisca la media ponderata di un dato studente.

```
1  -- Defined function che restituisce la media ponderata di un dato studente
2  DELIMITER $$
3  CREATE FUNCTION media_ponderata(matricola CHAR(9))
4  RETURNS float DETERMINISTIC
5  BEGIN
6      DECLARE mp float ;
7      SELECT SUM(c.cfu * e.voto)/ SUM(c.cfu)
8      INTO mp
9      FROM esami e INNER JOIN corsi s ON e.corso = s.codice
10     WHERE e.studente = matricola;
11     RETURN(mp);
12 END $$
13 DELIMITER;
```

User defined function 2

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 2

```
1  -- Defined function che restituisca il rank di uno studente nel suo corso di laurea
2  -- in base alla sua media ponderata.
3  DELIMITER $$
4  CREATE FUNCTION rank_cdl(matricola CHAR(9))
5  RETURNS INT DETERMINISTIC
6  BEGIN
7      DECLARE r INT ;
8      SELECT COUNT(*)
9      INTO r
10     FROM studenti s
11     WHERE cdl(s.matricola) = cdl(matricola) AND
12     media_ponderata(s.matricola) >= media_ponderata(matricola);
13     RETURN(r);
14 END $$
15 DELIMITER;
```

Trigger 1

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

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

Suggerimento: creare prima una tabella per registrare le assunzioni.

Trigger 1

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

Suggerimento: creare prima una tabella per registrare le assunzioni.

```
1  -- Trigger 1
2  -- Creazione della tabella assunzioni
3  CREATE TABLE assunzioni (
4  matricola INT(4) PRIMARY KEY,
5  data_assunzione DATE
6  );
```

Trigger 1

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

```
1  -- Trigger 1
2  DELIMITER $$
3  CREATE TRIGGER trg_data_assunzione
4  AFTER INSERT ON professori
5  FOR EACH ROW BEGIN
6      INSERT INTO assunzioni VALUES (NEW.matricola, CURDATE());
7  END $$
8  DELIMITER;
```

Grazie !