Basi di Dati

Esercitazione #4

Tutor

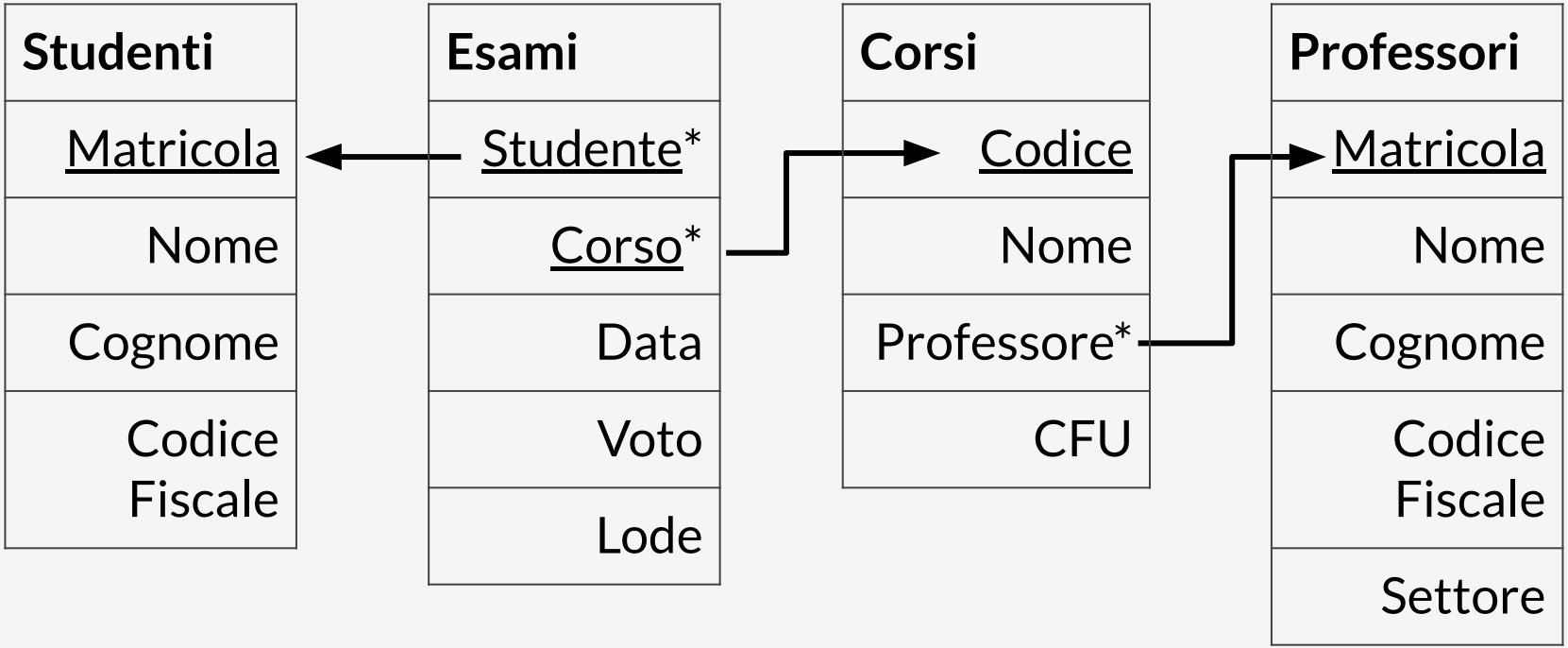
Mauro Farina

- Dottorando in ingegneria industriale e dell'informazione
- 1° anno
- Cybersecurity e misurazioni di Internet
- mauro.farina@phd.units.it

Materiale su GitHub: github.com/mauro-farina/database-exercises

Feedback: https://forms.gle/yAYRAvC3iHarHJyL6

"Modello logico"



Database

unidb.sql → https://github.com/mauro-farina/database-exercises

- 301 studenti
- 52 professori
- 31 corsi
- 2203 esami

Per caricare lo script…

1. Copia-incolla, oppure
2. Da MySQL Workbench: File > Open SQLScript

Esercizio Extra 8: EE8

Sul DB *classicmodels*, quanti ordini ha effettuato ciascun cliente (*customer*) in ogni anno (*orderDate*)? E quanto ha speso? Ordinare i risultati in ordine alfabetico rispetto al nome del cliente (*customerName*)

customerName	orderYear	numOrders	total
Alpha Cognac	2003	15	48051.04
Alpha Cognac	2005	5	12432.32
Amica Models & Co.	2004	26	82223.23
Anna's Decorations, Ltd	2003	27	80101.92
Anna's Decorations, Ltd	2005	19	56932.30
Atelier graphique	2003	4	14571.44

Esercizio Extra 9: EE9

Sul DB *classicmodels*, elencare i prodotti che (i) appartengono a una *productLine* contenente più di 20 prodotti e (ii) aventi un *buyPrice* maggiore del *buyPrice* medio della productLine a cui appartiene

Query sul Database

Esercizio 11

Quali studenti hanno migliorato almeno una volta la loro media fra un anno (**solare**) e l'altro?

1. Creare la Vista **`media_per_anno`**
2. Selezionare gli studenti che migliorano la media fra un anno e l'altro

Esercizio 11: soluzione (1)

1. Creare la Vista **`media_per_anno`**

matricola	nome	cognome	anno	media
EC2100025	Luca	Bianco	2019	27.0000
EC2100025	Luca	Bianco	2020	24.2000
EC2100025	Luca	Bianco	2021	25.5000
EC2100142	Matteo	Catalano	2013	26.3478
EC2100142	Matteo	Catalano	2016	23.9375
EC2100142	Matteo	Catalano	2017	21.0000

Esercizio 11: soluzione (2)

2. Selezionare gli studenti che migliorano la media fra un anno e l'altro

```
SELECT DISTINCT m1.matricola, m1.nome, m1.cognome
FROM media_per_anno m1
WHERE m1.media > (
    SELECT m2.media
    FROM media_per_anno m2
    WHERE m2.matricola=m1.matricola
       AND m2.anno < m1.anno
    ORDER BY m2.anno LIMIT 1
);
```

Esercizio 12

Scrivere una **transazione** che assegni al professore col minor carico di lavoro l'unico corso scoperto

- Corso scoperto = esiste, ma non ha un professore assegnato
- Consideriamo solo i professori che **già** insegnano almeno un corso

Esercizio 12: soluzione

```
START TRANSACTION;
SELECT matricola INTO @prof
FROM professori p
INNER JOIN corsi c ON c.professore = p.matricola
GROUP BY c.professore
ORDER BY sum(cfu) ASC LIMIT 1;
UPDATE corsi
SET professore = @prof
WHERE professore IS NULL ;
COMMIT;
```

Esercizio 13

Creare una **stored procedure** (`sp_ore_prof`) che restituisca in una variabile passata (ore, int) il monte ore di lezione di un dato professore (matricola, int).

Se questo non esiste, bisogna lanciare un errore.

- Nota: 1 CFU = 8h di lezione

Esercizio 13: soluzione

```
DELIMITER $$
CREATE PROCEDURE sp_ore_prof(IN prof INT, OUT ore INT)
BEGIN
    SELECT sum(cfu * 8) INTO ore FROM corsi c
    WHERE professore = prof;
    IF ore IS NULL THEN
        SIGNAL SQLSTATE "02000"
        SET MESSAGE_TEXT = "Prof. non esistente";
    END IF;
END $$
DELIMITER ;
```

Esercizio 14

Scrivere la **UDF** *rank_cdl* che restituisca il rank di uno studente nel suo corso di laurea (cdl) in base alla sua media ponderata

- "quanti studenti dello stesso cdl hanno media ponderata maggiore?"

Per convenienza, useremo due UDF già pronte:

1. *cdl*: matricola CHAR(9) → corso di laurea CHAR(4)
2. *media_p*: matricola CHAR(9) → media_ponderata FLOAT

Codice: github.com/mauro-farina/database-exercises → UDFs.sql

Esercizio 14: soluzione

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

Esercizio Extra 10: EE10

Sul DB *classicmodels*, elencare i clienti (*customerName*) che in almeno un anno hanno acquistato l'articolo più venduto in quell'anno

- Suggerimento: definire la UDF *most_sold_of_year*

Esercizio 15

Supponiamo si possano rifare gli esami: scrivere un **trigger** che verifichi che il nuovo voto non sia più basso del precedente. In tal caso, lancia un messaggio di errore

Esercizio 15: soluzione

```
DELIMITER $$
CREATE TRIGGER trg_no_voto_peggiore
BEFORE UPDATE ON esami
FOR EACH ROW BEGIN
    IF NEW.voto < OLD.voto OR NEW.lode < OLD.lode THEN
        SIGNAL sqlstate "45000"
        SET message_text = "Voto peggiore non ammesso!";
    END IF ;
END $$
DELIMITER ;
```

Esercizio Extra 11: EE11

Scrivere un **trigger** che, nel momento in cui viene inserito un corso scoperto (cioè senza un professore associato), lo assegna ad un prof. che non ha corsi (non importa a quale)

Esercizio 16

Creare la tabella **crediti**(studente VARCHAR(9), tot_cfu INT)

- studente è sia PK che FK con riferimento a **studente**(matricola)

Popolare la tabella **crediti** tramite una stored procedure

- Suggerimento: usare un cursore

Esercizio 16: soluzione (1/4)

Creare la tabella **crediti**(studente VARCHAR(9), tot_cfu INT)

- studente è sia PK che FK con riferimento a **studente**(matricola)

```
CREATE TABLE crediti(
    studente varchar(9) primary key,
    tot_cfu int,
    foreign key (studente) references
studenti(matricola)
);
```

Esercizio 16: soluzione (2/4)

Popolare la tabella **crediti** tramite una stored procedure

```
DELIMITER $$
CREATE PROCEDURE assegna_crediti()
BEGIN

    DECLARE stud VARCHAR(9);
    DECLARE tot_cfu INT;
    DECLARE fin BOOLEAN DEFAULT FALSE;

    [ ... ]
```

Esercizio 16: soluzione (3/4)

```
DECLARE curs CURSOR FOR
SELECT e.studente, sum(c.cfu)
FROM esami e
INNER JOIN corsi c ON e.corso = c.codice
GROUP BY e.studente;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET fin = TRUE;

[ ... ]
```

Esercizio 16: soluzione (4/4)

```
    OPEN curs;
    FETCH curs INTO stud, tot_cfu;
    WHILE fin IS FALSE DO
          INSERT INTO crediti VALUES (stud, tot_cfu);
          FETCH curs INTO stud, tot_cfu;
    END WHILE;
    CLOSE curs;

END $$
DELIMITER ;
```