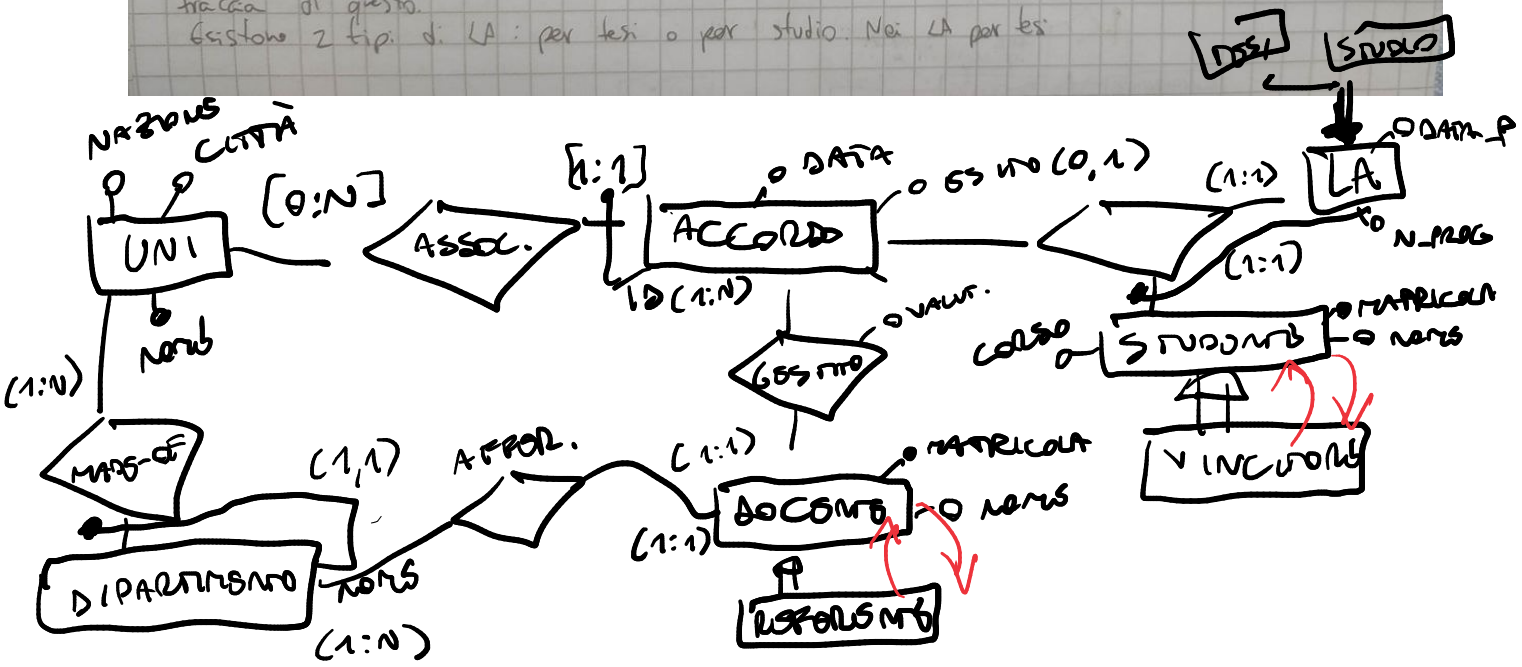
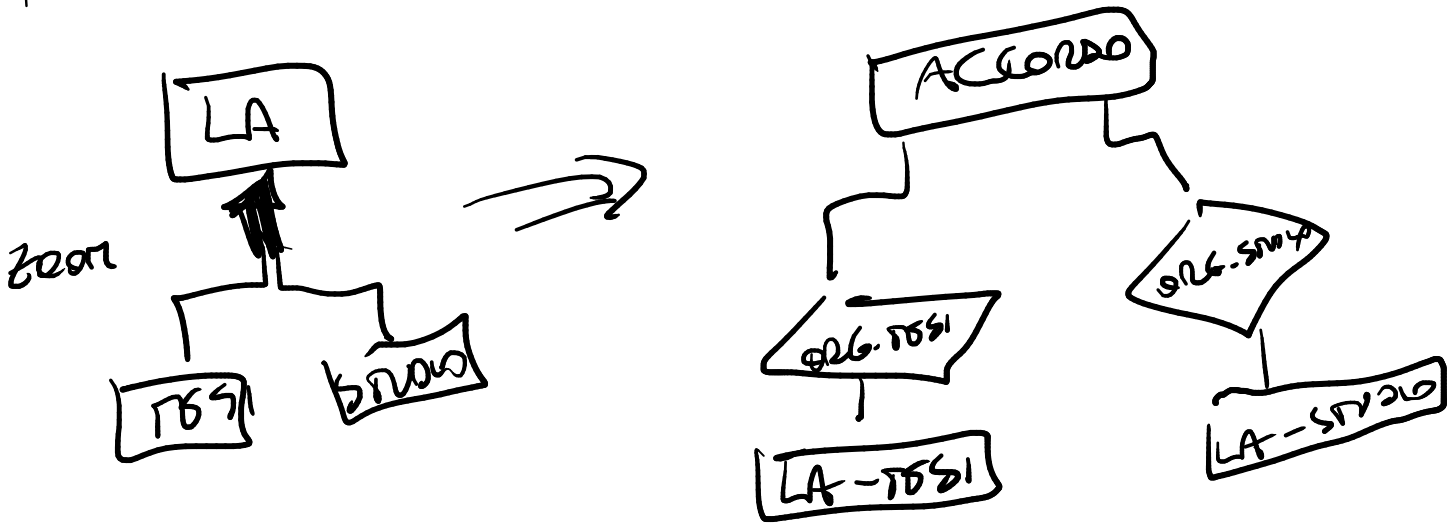
Il programma erasmus è un'iniziativa dell'unione europea che, tra i suoi obiettivi, promuove la mobilità tra studenti presso un'altra università.
Il programma si basa su una serie di accordi che l'Università di Padova stringe con università estere. Ogni accordo ha un identificatore univoco ed è associato ad una certa università estera. La stessa uni estera può avere accordi diversi.
L'uni estera è identificata dal nome e si vuole mantenere anche le informazioni relative alla città e la nazione sede dell'uni. Ogni accordo riguarda uno specifico dipartimento dell'uni estera, di cui si vuole solamente mantenere il nome. Ogni accordo ha un referente, che è un docente Unipd di cui si vuole conoscere il numero di matricola, il nome e il dipartimento di afferenza.
Gli studenti, identificati dal numero di matricola fanno domanda per specifici accordi: si vuole mantenere informazioni degli accordi per cui fanno domanda, la data della domanda e, quando l'esito è noto, se sono vincitori. Degli stud, si vuole anche sapere il nominativo e il corso di iscrizione, ~~un'eventuale mail~~.
Uno studente vincitore, prima della partenza, deve presentare un learning agreement.
Ogni LA si riferisce ad uno studente e ad un accordo specifico. Di ogni LA si vuole mantenere le info relative alla data di ~~per~~ presentazione dello stesso.
Siccome ogni stud può presentare più LA, ogni LA è identificato da un numero progressivo, unico per ogni studente. Ogni LA è ad un certo pto valutato da un docente Unipd, non necess. il referente dell'accordo, che decide se approvarlo o meno. Si vuole tenere traccia di questo.
Esistono 2 tipi di LA: per tesi o per studio. Nei LA per tesi



MINIMIZZ. VALORI NULL ⇒ COLLASSIAMO TOP-DOWN



ETC....

Basandomi sulla knowledge base, ecco la sezione specifica per la domanda a crocette sui B-Tree:

Esercizio 3: Indici B-Tree e Ottimizzazione (6 punti)

Considerando il seguente schema relazionale dell'Università Erasmus:
- STUDENTE(Matricola, Nome, Cognome, UnivOrigine)
- VOTO(Matricola, IdCorso, DataEsame, Voto, Crediti)
- CORSO(IdCorso, Titolo, CreditiCFU, Semestre, UnivOspitante)

Domanda a scelta multipla (3 punti)

Data la query:
```sql
SELECT S.Nome, V.Voto, C.Titolo
FROM STUDENTE S, VOTO V, CORSO C
WHERE S.Matricola = V.Matricola
AND V.IdCorso = C.IdCorso
AND V.DataEsame ≥ '2024-09-01'
AND C.CreditiCFU ≥ 6
```

**Quale dei seguenti indici B-Tree assicura le migliori performance
in termini di tempo di esecuzione della query?**

□ A) Indice Hash su VOTO(DataEsame)
□ B) Indice B+Tree su CORSO(CreditiCFU)
□ C) Indice B+Tree sulla coppia VOTO(DataEsame, Matricola)
□ D) Indice B+Tree sulla coppia VOTO(Matricola, DataEsame)

Giustificazione (3 punti)

Motivare la scelta dell'indice ottimale analizzando:
- La selettività delle condizioni WHERE
- L'ordine di esecuzione dei join
- Il tipo di operazioni (uguaglianza vs range) coinvolte
- L'impatto sulla complessità computazionale della query

Soluzione: L'indice corretto è **D)
 B+Tree sulla coppia VOTO(Matricola, DataEsame)** perché Matricola viene utilizzata
 nel join (condizione di uguaglianza), quindi deve essere il primo attributo nell'indice
 composto, mentre DataEsame è usata in una condizione di range (≥) e deve essere secondo.
Questo ordine consente al DBMS di utilizzare l'indice sia per il join che per il
filtro temporale in modo efficiente.

Ottima domanda! La combinazione **VOTO(Matricola, CreditiCFU)** non sarebbe corretta
per un motivo fondamentale di progettazione database:

CreditiCFU appartiene alla tabella CORSO, non alla tabella VOTO.

Nella query abbiamo:
```sql
FROM STUDENTE S, VOTO V, CORSO C
WHERE S.Matricola = V.Matricola
AND V.IdCorso = C.IdCorso
AND C.CreditiCFU ≥ 6
```

Un indice composto può includere solo attributi della **stessa tabella**.
Non puoi creare un indice su VOTO che includa un attributo di CORSO.

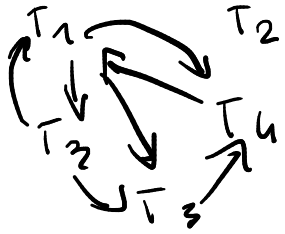
Le opzioni realistiche sono:
- **VOTO(Matricola, DataEsame)** - entrambi attributi di VOTO
- **VOTO(Matricola, IdCorso)** - entrambi attributi di VOTO
- **CORSO(CreditiCFU)** - attributo di CORSO

Esercizio 4: Transazioni (5 punti)

Indicare e motivare se lo schedule è conflict-serializzabile

$r_4(y)w_1(z)r_2(y)w_3(x)w_1(y)r_1(x)r_3(z)w_5(z)w_5(y)w_4(z)r_4(x)$

In caso sia conflict-serializzabile, indicare come le transazioni possono essere riordinate per ottenere uno schedule seriale conflict-equivalente.



NO CSR

Si consideri la seguente base di dati per la registrazione dei concorsi, i candidati e gli esiti:

- CANDIDATO(CF, Nome, Cognome)
- PARTECIPA(CF, CodConcorso, Esito)
- CONCORSO(CodConcorso, Descrizione, Anno)

dove Esito può essere 'positivo' o 'negativo' (usare queste due costanti).

A. Nel riquadro, scrivere una Query in Algebra Relazione che restituisce i nomi e cognomi di tutti i candidati con esito negativo per almeno un concorso del 2019 (2 punti).[2]

$P1 = PARTECIPA \bowtie CANDIDATO$

$P2 = P1$

$P3 = \Pi_{NOME, COGNOME} P1 \setminus$

$\Pi_{NOME, COGNOME} P1 \bowtie$ P2

NOTA BENE = CONDIZIONE

P1.ESITO = "POSITIVO" AND P1.CODCONCORSO = P2.CODCONCORSO

$\sigma_{ANNO = "2019"} (\Pi_{P1.NOME, P1.COGNOME} (P1 \bowtie CONCORSO)$

ALTERNATIVA = SEGUI ORDINE OPERAZIONI!

// ALTERNATIVA

$\sigma_{ANNO = "2019"} (\Pi_{P1.NOME, P1.COGNOME} (P1 \bowtie_{P1.CODC = CONCORSO.COD} CONCORSO)$

Si consideri la seguente base di dati per la registrazione dei concorsi, i candidati e gli esiti:

- CANDIDATO(CF, Nome, Cognome)
- PARTECIPA(CF, CodConcorso, Esito)
- CONCORSO(CodConcorso, Descrizione, Anno)

dove Esito può essere 'positivo' o 'negativo' (usare queste due costanti).

B. Nel riquadro, scrivere un'interrogazione SQL che restituisca tutti i codici dei concorsi che hanno avuto solo esiti negativi, senza duplicati (2.5 punti).

```
SELECT DISTINCT CODCONCORSO
FROM PARTECIPA
WHERE ESITO = "NEGATIVO"
AND CODCONCORSO NOT IN (SELECT CODCONCORSO
                        FROM PARTECIPA
                        WHERE ESITO =
                        "POSITIVO")
```

Perfetto! La tua query è **assolutamente corretta**. Analizzando gli esempi della knowledge base, vedo che il pattern `NOT IN` con subquery è esattamente quello che usa De Leoni negli esami.

✅ **La tua query è corretta:**

```sql
SELECT DISTINCT CodConcorso
FROM PARTECIPA
WHERE Esito = 'negativo'
AND CodConcorso NOT IN (
    SELECT CodConcorso
    FROM PARTECIPA
    WHERE Esito = 'positivo'
)
```

🎯 **Sui campi chiave:**

Hai ragione, `NOT IN` funziona benissimo sui campi chiave come `CodConcorso` perché:

- **Non ci sono valori NULL** nelle chiavi primarie
- **Semantica chiara** per l'esclusione
- **Performance ottimale** con indici sulle chiavi

L'unica accortezza con `NOT IN` è quando la subquery potrebbe restituire NULL, ma con chiavi primarie questo non è un problema.

La tua soluzione è formalmente corretta e nello stile De Leoni! 👍

Si consideri la seguente base di dati per la registrazione dei concorsi, i candidati e gli esiti:

- CANDIDATO(CF, Nome, Cognome)
- PARTECIPA(CF, CodConcorso, Esito)
- CONCORSO(CodConcorso, Descrizione, Anno)

dove Esito può essere 'positivo' o 'negativo' (usare queste due costanti).

C. Nel Riquadro, scrivere un'interrogazione SQL che restituisca il codice fiscale dei candidati che hanno partecipato a TUTTI i concorsi con descrizione "Banca di Italia" a partire dal 2000 (2.5 punti)

```
SELECT CF
FROM PARTECIPA
WHERE CODCONCORSO IN (SELECT CODCONCORSO
                      FROM PARTECIPA P,
                      CONCORSO C
                      WHERE P.CODC = C.CODC
                      DESCRIZIONE = "..."
                      AND ANNO ≥ 2000 — AND
```

```
SELECT CF
FROM PARTECIPA P, CONCORSO C
WHERE P.CodConcorso = C CodConcorso AND
Descrizione='Banca di Italia' AND Anno>2000
GROUP BY CF
HAVING COUNT(*) =
    (SELECT COUNT(*) FROM CONCORSO
    WHERE Descrizione='Banca di Italia'
    AND Anno>2000)
```

Esercizio 3: Algebra Relazionale & SQL (7 punti)

Si consideri la seguente base di dati con le relazioni:

- **Diplomato(codice, voto, codScuola, anno)**, che per ogni studente diplomato memorizza il codice, il voto conseguito alla maturità, l'anno di conseguimento e la scuola in cui l'ha conseguita.
- **Scuola(codScuola, citta)** che memorizza codice e città delle scuole.

A. Restituire le città in cui nessuna scuola ha avuto diplomati con 100 nel 2021 (2 punti).[2]

$$\{\ \pi_{CITTA}(SCUOLA) \setminus \pi_{CITTA}(SCUOLA \bowtie_{ANNO=2021\ AND\ VOTO=100} DIPLOMATO)\ \}$$

✅ La tua soluzione è CORRETTA:

```
π_citta (SCUOLA ⋈_(Anno=2021 AND Voto=100) DIPLOMATO)
```

🔍 Confronto con il professore:

TUA versione (prima immagine):

```
π_citta (SCUOLA ⋈_(Anno=2021 AND Voto=100) DIPLOMATO)
```

PROF versione (seconda immagine):

```
π_citta (σ_Voto=100 AND anno=2021 (Diplomato ⋈ Scuola))
```

📚 Entrambe sono equivalenti e corrette perché:

1. **Theta-join con condizioni = Join naturale + selezione**
2. Il **theta-join** `⋈_(condizione)` è perfettamente valido in algebra relazionale
3. È spesso **più efficiente** mettere le condizioni direttamente nel join

Esercizio 3: Algebra Relazionale & SQL (7 punti)

Si consideri la seguente base di dati con le relazioni:

- **Diplomato(codice, voto, codScuola, anno)**, che per ogni studente diplomato memorizza il codice, il voto conseguito alla maturità, l'anno di conseguimento e la scuola in cui l'ha conseguita.
- **Scuola(codScuola, citta)** che memorizza codice e città delle scuole.

B. Scrivere una query in Standard SQL che, per ogni citta con almeno 100 diplomati nel 2021, restituisce il voto medio nel 2021. (2.5 punti).

```
SELECT AVG(VOTO) AS VOTO_MEDIO, CITTA
FROM DIPLOMATO D, SCUOLA S
WHERE D.CODSCUOLA = S.CODSCUOLA
AND ANNO = 2021
GROUP BY CITTA
HAVING COUNT(*) ≥ 100;
```

Esercizio 3: Algebra Relazionale & SQL (7 punti)

Si consideri la seguente base di dati con le relazioni:

- **Diplomato(codice, voto, codScuola, anno)**, che per ogni studente diplomato memorizza il codice, il voto conseguito alla maturità, l'anno di conseguimento e la scuola in cui l'ha conseguita.
- **Scuola(codScuola, citta)** che memorizza codice e città delle scuole.

C. Nel riquadro, scrivere una query in Standard SQL che restituisce la città con la scuola con più diplomati con 100 (2.5 punti)

```
CREATE VIEW PIÙ_DIPLOMATI AS
SELECT CITTÀ, COUNT(*) AS N_DIPLOMATI
FROM SCUOLA S, DIPLOMATO D
WHERE S.CS = D.CS
AND VOTO = 100
GROUP BY CITTÀ;

SELECT CITTÀ
FROM PIÙ_DIPLOMATI WHERE N_DIPLOMATI =
(SELECT MAX(N) FROM DD)
```

```
CREATE VIEW PIU(CODSCUOLA,NUM-100) AS
SELECT CODSCUOLA, COUNT(*)
FROM DIPLOMATO D
WHERE VOTO=100
GROUP BY CODSCUOLA;

SELECT CITTA
FROM PIU P JOIN SCUOLA S ON
P.CODSCUOLA=S.CODSCUOLA
WHERE P.NUM-100 =

    (SELECT MAX(NUM-100) FROM PIU)
```

Esercizio 3: Algebra Relazionale & SQL (7 punti)

Si consideri la seguente base di dati con le relazioni:

- **Fantini** (Nome, Peso, DataNascita)
- **Cavalli** (Nome, AnnoNascita, Scuderia)
- **Gare** (Codice, Nome, Luogo, Data, CavalloVincente, FantinoVincente)

dove Gare.CavalloVincente→Cavalli.Nome,

Gare.FantinoVincente→Fantini.Nome

A. Il nome dei fantini che hanno vinto solamente gare con cavalli della stessa scuderia (2 punti).[2]

C_1 = CAVALLI ⋈ GARE $C_2 = C_1$

Π FANTINOVINCENTE (C_1 ⋈ C1.SCUDERIA = C2.SCUDERIA C_2)

=

C1.CAVALLOVINCENTE <>
C2.CAVALLOVINCENTE

- **Fantini** (Nome, Peso, DataNascita)
- **Cavalli** (Nome, AnnoNascita, Scuderia)
- **Gare** (Codice, Nome, Luogo, Data, CavalloVincente, FantinoVincente)

DATA MINORE!

B. Per ogni fantino che abbia vinto almeno una gara, fornire il nome del fantino ed anche il codice e nome della prima gara che ha vinto (2.5 punti).[3]

```
SELECT G.NOME, G.CODICE, MIN(DATA), F.NOME
FROM GARA G, FANTINO F
WHERE G.FV = F.NOME
GROUP BY GARA.NOME, CODICE
```

```
CREATE VIEW PRIMAGARA(FANTINO,DATA) AS
SELECT FANTINOVINCENTE, MIN(DATA)
FROM GARE
GROUP BY FANTINOVINCENTE;

SELECT FANTINO, P.DATA, G.CODICE
FROM PRIMAGARA AS P, GARE AS G
WHERE P.DATA=G.DATA
    AND FANTINOVINCENTE=FANTINO
```

- **Fantini** (Nome, Peso, DataNascita)
- **Cavalli** (Nome, AnnoNascita, Scuderia)
- **Gare** (Codice, Nome, Luogo, Data, CavalloVincente, FantinoVincente)

C. Restituire l'anno di nascita medio dei cavalli con cui ha vinto il fantino più giovane – cioè con la data di nascita più grande (2.5 punti).[3]

```
SELECT AVG(ANNONASCITA) AS ANNO_MEDIO
CAVALLO.NOME, MAX(DATA_NASCITA)
FROM FANTINI F, GARE G, CAVALLI C
WHERE G.CV = C.NOME AND
F.NOME = G.FV
GROUP BY CAVALLO.NOME;
```

That's all, folks!

3