

**Esercizio 9** Sia dato il seguente schema relazionale che descrive l'insieme degli esami obbligatori e che è possibile scegliere in un anno (I, II, III) in un indirizzo di un corso di laurea triennale:

*CORSO(Cod-C, Nome, CFU, SSD)*

*INDIRIZZO(Cod-I, Titolo)*

*COMPOSIZIONE(Cod-I, Cod-C, Tipo, Anno)*

dove l'attributo tipo assume i possibili valori 'Obb' (Obbligatorio) oppure 'Scelta' ed Anno assume i possibili valori I, II e III.

1. Determinare i titoli degli indirizzi che prevedono lo stesso insieme di esami obbligatori dell'indirizzo 'Sistemi'.
2. Determinare i titoli dei corsi che in almeno un indirizzo possono essere collocati indifferentemente in ciascuno dei tre anni.
3. Determinare l'indirizzo (o gli indirizzi) che ha (hanno) il maggior numero di CFU (complessivamente) legato a insegnamenti obbligatori.
4. Determinare i titoli dei corsi che sono presenti in tutti gli indirizzi e sono o sempre obbligatori o sempre facoltativi (in ciascun indirizzo).

$$1) \quad \text{IND-SIS} \leftarrow \sigma_{\text{Titolo} = 'SISTEMI'} \left( \text{INDIRIZZO} \bowtie \text{COMPOSIZIONE} \right) \quad \text{TIPO} = 'OBBL'$$

$$\overline{\text{INDIRIZZO}} \leftarrow \prod_{\text{Titolo} \in 'SISTEMI'} \left( \text{INDIRIZZO} \bowtie \text{COMPOSIZIONE} \bowtie \text{IND-SIS} \right) \quad \text{COMPOSIZIONE} = \text{COP-C}$$

$$2) \quad \text{ANNO3} \leftarrow \prod_{\text{COD-13}, \text{COD-13} \in 3} \left( \text{COMPOSIZIONE} \right) \quad \text{ANNO} = '3'$$

$$\text{ANNO2} \leftarrow \prod_{\text{COD-12}, \text{COD-12} \in 2} \left( \text{COMPOSIZIONE} \right) \quad \text{ANNO} = '2'$$

$$\text{ANNO3} \leftarrow \prod_{\text{COD-13}, \text{COD-13} \in 3} \left( \text{COMPOSIZIONE} \right) \quad \text{ANNO} = '3'$$

$$R \leftarrow \prod_{\text{COP-C3}} \left( \text{ANNO3} \bowtie \text{ANNO2} \bowtie \text{ANNO3} \right) \quad \begin{array}{l} 1_3 = 1_2 \text{ AND } 2_3 = 2_2 \\ 1_2 = 1_3 \text{ AND } 2_2 = 2_3 \end{array}$$

$$\overline{\text{INDIRIZZO}} \leftarrow \text{COP-C3}$$

$$3) \quad \text{CORSO} \leftarrow \rho_{\text{Cod-C}} \text{ F SUM(CFU)} \left( \text{INDIRIZZO} \bowtie \left( \text{COMPOSIZIONE} \bowtie \text{CORSO} \right) \quad \text{TIPO} = 'OBBL' \right)$$

3)  $S_{CF} = \rho \left( \text{compos. F SORH(CFU)} \left( \text{IND: Q1220 or } G(\text{composition}) \rightarrow \text{corso} \right) \right)$   
 $\sigma_{CF} = \sqrt{\rho_0} = \sqrt{6BB'}$

$$4) NO\_T1PO \leftarrow \neg \left( \left( \begin{array}{c} p(COMP0.S2.DNE) \wedge COMP0.S2.ON \\ \downarrow \\ CI.CODE,T,A \end{array} \right) \rightarrow \neg \text{COMP} = \text{CODE} \right)$$

$T \hookrightarrow T1PO$  AND  $CI \hookrightarrow \text{COMP}$

$\Delta C$   $\propto p(\text{cond} - c \sum_{i=1}^n \text{COUNT}(c_i))$  ( $S_1 \propto \text{COMPOSITION}^3$ )  
 CORSO, N-INPI(22)

$r_1 \propto p \in \text{count}(*) \text{ (WDIR,220)}$   
TOT INP.

$$R \leftarrow (T_C \bowtie T_I) \quad \Pi(R \bowtie \text{corsi})$$

$M_{IND}(1,2,1) = TOTIND$

**Esercizio 10** Sia dato il seguente schema relazionale che descrive i risultati di alcune stagioni di incontri calcistici:

**STADIO**(Nome, Città, Capienza)

*PARTITA*(NomeStadio, Data, Squadra1, Squadra2, Goal1, Goal2)

**SQUADRA**(*NomeSquadra.Città*)

1. Determinare le città dove le squadre milanesi non hanno mai vinto.
  2. Determinare la città dove la squadra udinese ha vinto il maggior numero di partite.
  3. Determinare la città dove la squadra udinese ha subito il maggior numero di goal in una partita.

5) MILAN → - P (SQUADRA)  
CITTÀ = MILANO

Per città si intende l'entità degli Stati).

VITCASP 2 - 6 ( PARTITA DI MILAN  
SQUADRA 1: NORE SQUADRA )  
GOALS ? GOALS

VITT-FUORI - 6 (PARTITA CON MILAN  
SQUADRA 2: NOBISQUADRA)  
GOAL 2 > GOAL 1

$$\text{NO} \leftarrow \overline{\Pi} (\text{VITTCASA} \vee \text{VITTFUORI})$$

NONESQUADRA CITTÀ

$$SI \leftarrow (\text{SQUADRA} \wedge \text{NO})$$

$$\overline{\Pi} ((SI \wedge \text{PARTITA}) \wedge \text{PARTITA} \wedge \text{STADIO})$$

NONESQUADRA = SQUADRA  $\wedge$  NONESQUADRA = SQUADRA  
STADIO = CITTÀ

2) UD, NOME\_SQ

$$\rho \left( \zeta(\text{SQUADRA}) \right)$$

NOMESQUADRA = 'UDINESE'  
CITTÀ

$$V1 \leftarrow \rho \left( \text{NOMESTADIO}, \text{CITTÀ} \sum_{\text{COUNT}(*)} \left( \begin{array}{c} G (\text{PARTITA} \wedge \text{UDINESE} \wedge \text{STADIO}) \\ \text{GOAL1} > \text{GOAL2} \end{array} \right) \right)$$

$$V2 \leftarrow \rho \left( \text{NOMESTADIO}, \text{CITTÀ} \sum_{\text{COUNT}(*)} \left( \begin{array}{c} G (\text{PARTITA} \wedge \text{UDINESE} \wedge \text{STADIO}) \\ \text{GOAL2} > \text{GOAL1} \end{array} \right) \right)$$

$$\overline{\Pi} \left( \text{NOMESTADIO}, \text{CITTÀ} \sum_{\text{MAX}(V1, V2)} (V0 \wedge V2) \right)$$

CITTÀ

3)

$$\rho \left( \zeta(\text{SQUADRA}) \right)$$

NOMESQUADRA = 'UDINESE'  
CITTÀ

$$R1G \leftarrow \rho \left( \text{NOMESTADIO} \sum_{\text{MAX(GOAL)}} \left( \begin{array}{c} G (\text{PARTITA} \wedge \text{UDINESE}) \\ \text{SQUADRA} = \text{NOMESQUADRA} \end{array} \right) \right)$$

GOAL  $\leftarrow$  MAX(GOAL2)

$$R2G \leftarrow \rho \left( \text{NOMESTADIO} \sum_{\text{MAX(GOAL)}} \left( \begin{array}{c} G (\text{PARTITA} \wedge \text{UDINESE}) \\ \text{SQUADRA} = \text{NOMESQUADRA} \end{array} \right) \right)$$

GOAL  $\leftarrow$  MAX(GOAL1)

$$R \leftarrow \text{NOMESTADIO}, \text{MAX(GOAL)} (R1G \vee R2G)$$

$$\overline{\Pi} \left( R \wedge \text{STADIO} \right)$$

CITTÀ

STADIO(Nome, Città, Capienza)

PARTITA(NomeStadio, Data, Ora, Squadra1, Squadra2, Goal1, Goal2)

SQUADRA(Nazionale, Continente)

1. Determinare il nome degli stadi dove non gioca nessuna nazionale africana.
2. Determinare la capienza complessiva degli stadi in cui si giocano partite di nazionali africane (si sommino le capienze associate a ciascuna gara).
3. Determinare la città (o più città) in cui la nazionale italiana gioca il maggior numero di partite;

1)  $\text{AFRICANAS} - \sum (\text{SQUADRA})$   
CONTINENTE = AFRIKA

$$\rho_A \leftarrow \prod \left( \text{PARTITA} \Delta \text{AFRICANAS} \right)$$

SQUADRA1 = NAZIONE OR SQUADRA2 = NAZIONE  
 NOSTRADIO = PARTITA, OPA, SQUADRA1.SQUADRA2, GOAL1, GOAL2

$$\prod_{\text{NOME STADIO}} (\text{PARTITA} \setminus \rho_A)$$

2) SFRUTTIAMO LE VISTE PRECEDENTI

$$\text{NOSTRADIO} \sum_{\text{SQUADRA}} (\text{CAPACITA}) \quad \left( \rho_A \Delta \text{STADIO} \right)$$

NOSTRADIO = STADIO

3) ITALIA -  $\sum (\text{SQUADRA})$   
NAZIONE = ITALIA  
CONTINENTE

$$\rho(\text{CITTA'}) \sum_{\text{CITTA'}} \text{COUNT}(\#) \left( \text{PARTITA} \Delta \text{ITALIA} \quad \times \text{STADIO} \right)$$

SQUADRA1 = NAZIONE OR SQUADRA2 = NAZIONE  
 NOSTRADIO = NOME

$$\prod_{\text{CITTA'}} (\text{CITTA'} \sum_{\text{PARTITA}} (\text{PARTITA} \setminus (\text{CITTA'} \times (\text{NOSTRADIO} \setminus \text{CITTA'})))$$

4. determinare le squadre che incontrano solo squadre dello stesso continente.
5. determinare le squadre che giocano esattamente negli stessi stadi della nazionale italiana.

4)

$$\rho_{\text{SQUADRA}} \left( \left( \rho \left( \begin{array}{l} \text{PARTITA} \Delta \text{SQUADRA} \\ \text{SQUADRA1} = \text{NAZIONE} \end{array} \right) \right) \Delta \text{SQUADRA} \right)$$

CONTINENTE  
 AND SQUADRA2 = NAZIONE

NAZIONE > NAZIONE

$$CP \leftarrow \rho(NAZIONE, COUNT(*)) (PARTITA \bowtie SQUADRA)$$

N-PARTITE ↗  
 SQUADRA = NAZIONE  
 OR SQUADRA = NAZIONE

$$C2 \leftarrow \bigcap \left( \left( \rho \left( \begin{array}{l} PARTITA \bowtie SQUADRA \\ SQUADRA = NAZIONE \end{array} \right) \right) \bowtie SQUADRA \right)$$

CON = CONTINGENTE, NAZIONE  
 AND SQUADRA = NAZIONE

NAZ > NAZIONE

$$\begin{matrix} NP_1 \\ R \\ \rho \left( NAZ \underset{NAZIONE, N-PARTITE}{\sim} COUNT(*) \right) \end{matrix}$$

$$\begin{matrix} NP_2 \\ R \\ \rho \left( NAZ \underset{NAZIONE, N-PARTITE}{\sim} COUNT(*) \right) \end{matrix}$$

$$NAZIONI\_C \leftarrow \rho \left( NAZIONE^2 \underset{N-PARTITE}{\sum} SUM(N\_PARTITE)(NP_1 \cup NP_2) \right)$$

$$\prod_{NP_1 \cup NP_2} (NAZIONI\_C \bowtie CP) \text{ b/w } \text{N-REPATRIO E IN SVOLTA NAZIONE C'E SUL NUMERO DI PARTITE}$$

5)

STADI\_NAZ\_ITALIA\_PTA ~

$$\prod_{NS} \rho \left( ITALIA \bowtie PARTITA \right)$$

NS & NON-STADI

$$NO \leftarrow \prod_{NAZIONE, CONTINGENTE} \left( (PART_{-N-1} \bowtie STADIO) \setminus (PART_{-N-1} \bowtie STADI_NAZ_ITALIA_PTA) \right)$$

NON-STADIO = NO

PART\_{-N-1}

ITALIA  $\bowtie$  (SQUADRA)  
NAZIONE = ITALIA

$$\left( \bigcap (SQUADRA) \underset{NAZIONE \bowtie ITALIA}{\bowtie} PARTITA \right)$$

SQUADRA = NAZIONE  
OR SQUADRA = NAZIONE

$$SI \leftarrow \prod_{NAZIONE} (SQUADRA \setminus NO)$$

**Esercizio 12** Sia dato il seguente schema relazionale:

AUTOMOBILE(Targa, Cilindrata, Modello, Casa, Nazione, Tasse)  
 PROPRIETARIO(CodiceF, Nome, Targa)

1. Determinare il nome delle persone che posseggono solo automobili della stessa casa costruttrice.
2. Determinare il nome delle persone che posseggono solo automobili tedesche di almeno due case costruttrici diverse.
3. Determinare le tasse che ogni persona deve pagare per le automobili possedute.
4. Determinare la nazione (o le nazioni) in cui è prodotto il maggior numero di automobili tra quelle registrate nella base di dati
5. Determinare le persone che possiedono solo automobili di cilindrata minima (tra quelle registrate nella base di dati).

1)  $\text{no} \leftarrow \overline{\Pi}_{\text{CODICEF}, \text{NOME}} \left( \overline{\sigma}_{\substack{\text{TARGA} \\ \text{CASA}}} \left( \overline{\sigma}_{\substack{\text{TARGA} \\ \text{CASA}, \text{NITA}}} \left( \text{PROPRIETARIO} \bowtie \text{AUTOMOBILE} \right) \right) \bowtie \text{AUTOMOBILE} \right)$

$$\overline{\Pi}_{\text{NOME}} \left( \text{PROPRIETARIO} \setminus \left( \text{PROPRIETARIO} \bowtie \text{NO} \right) \right)$$

2) A1 A2  
 $\overline{\Pi}_{\text{CODICEF}, \text{TARGA}, \text{CASA}} \left( \overline{\sigma}_{\substack{\text{NATION} = 'GERMANIA'}} \left( \text{PROPRIETARIO} \bowtie \text{AUTOMOBILE} \right) \right)$

R  $\leftarrow \overline{\Pi}_{\text{CODICEF}} \left( \overline{\sigma}_{\substack{\text{CODICEF} = \text{CODICEF} \\ \text{TARGA} \leftrightarrow \text{A2.TARGA} \text{ AND } \text{A1.CASA} \leftrightarrow \text{A2.CASA}}} \left( \overline{\sigma}_{\substack{\text{A1} \wedge \text{A2} \\ \text{CODICEF}}} \right) \right)$

$$\overline{\Pi}_{\text{NOME}} (R \bowtie \text{PROPRIETARIO})$$

3)  $\text{SUM}(\text{TASSE}) \leq \text{SUR}(\text{TASSE}) \left( \text{PROPRIETARIO} \bowtie \text{AUTOMOBILE} \right)$   
 $\text{SUM}(\text{TASSE})$

4)  $N1 \leftarrow \overline{\sigma}_{\substack{\text{NATION} \in \text{COUNT}(\text{*}) \\ \text{N-AUTO}}} \left( \text{AUTOMOBILE} \right)$

$\overline{\Pi} (\text{AZIENDA} \in \forall A \times (N \text{- AUTO}) \text{ (AS)})$

AZIENDA

5)  $\overline{\Pi} \left( \begin{array}{l} C_{\text{MIN}} \\ \leq N \text{ (CILINDRATA)} \end{array} \right) \text{ (AUTOMOBILE)}$

CILINDRATA

$\overline{\Pi} (\text{PROPRIETARIO DI AUTOMOBILE DI } C_{\text{MIN}})$

NONO

Esercizio 13 Sia dato il seguente schema relazionale:

CORSO(Codice, Titolo, CFU, SSD)

ATTIVAZIONE(CodiceCorso, Gruppo, AnnoA, Periodo, Titolare)

VINCOLO(Cod-Corso, Cod-Vincolante, Tipo)

dove la relazione VINCOLO esprime i vincoli di 'Propedeuticità' e 'Prerequisito' tra i corsi.

- Determinare i titoli dei corsi che hanno le stesse propedeuticità e gli stessi prerequisiti del corso 'Basi di Dati'.
- Determinare i titoli dei corsi attivati nell'anno accademico corrente con numero di CFU minimo.
- Determinare i titoli di corsi che hanno nell'anno accademico corrente il massimo numero di attivazioni.
- Determinare i titoli dei corsi che negli anni tra il 1995 e il 2000 non sono stati attivati almeno una volta.
- Determinare i titoli dei corsi che negli anni tra il 1995 e il 2000 sono stati attivati almeno in due anni distinti.
- Trovare l'insieme dei corsi attivati in un anno che hanno corsi propedeutici non attivati in quello stesso anno.
- Scrivere (se possibile) una espressione che dato un corso fornisca la chiusura transitiva di tutte le sue propedeuticità (propedeutici dei propedeutici etc.)

1)  $\text{CBD} \circ \overline{\Pi} (G(\text{corso}))$   
TITOLO = 'BASI DI DATI'

VS  $\neg \overline{\Pi} \left( G \left( \begin{array}{l} \text{VINCOLO} \bowtie \text{CBD} \\ \text{CODCORSO} = \text{CODICE} \\ \text{TIPPO} = \text{'PROPEDUTICITA'} \end{array} \right) \right)$   
NULL, COD-VINCOLANTE, TIPPO

$\overline{\Pi} \left( G \left( \begin{array}{l} \text{VINCOLO} \bowtie \text{CBD} \\ \text{COD-VINCOLANTE} = \text{CODICE} \\ \text{TIPPO} = \text{'PREREQUISITO'} \end{array} \right) \right)$   
COD-CORSO, NULL, TIPPO

VP1  
 $\rho$  (CODICE  $\in \text{COUNT}(\ast)$ )  
N-PROP

$\left( \overline{\Pi} \left( G \left( \begin{array}{l} \text{VINCOLO} \bowtie \text{corso} \\ \text{CODCORSO} = \text{CODICE} \\ \text{TIPPO} = \text{'PROPEDUTICITA'} \end{array} \right) \right) \setminus V1 \right)$   
CODICE, COD-VINCOLANTE, TIPPO

VP2  $\neg (\neg (VINCOLO \bowtie \text{corso}))$

VP2

$$\rho \left( \text{CODICE} \underset{\text{N\_PROB}}{\overset{\text{TO}}{\exists}} \text{COUNT}(\#) \right) \left( \prod_{\text{CODICE}, \text{COD-CORSO}, \text{TIPPO}} \left( \bigcap_{\text{VINCULO} \propto \text{CORSO}} \left( \begin{array}{l} \text{COD-VINCULANTE} = \text{CODICE} \\ \text{TIPPO} = \text{'PREZARVITO'} \end{array} \right) \right) \setminus \text{V2} \right)$$

$$\prod_{\text{CODICE}} \left( \bigcap_{\text{VINCULO} \propto \text{CORSO}} \left( \begin{array}{l} \text{VINCULO} \propto \text{CORSO} \\ \text{COD-VINCULANTE} = \text{CODICE} \\ \text{TIPPO} = \text{'PREZARVITO'} \end{array} \right) \right) \setminus \text{V2}$$

2)

$$CFU-TIN \Leftarrow \rho \left( \sum_{CFU-TIN} CFU \left( \text{CORSO} \right) \right)$$

$$\prod_{\text{TITOLO}} \left( \bigcap_{\text{ANNOA} = \text{YEAR(SUSSDATO)}} \left( \begin{array}{l} \text{CORSO} \propto \text{ATTIVAZIONE} \\ \text{CODICE} = \text{CODICE-CORSO} \\ \text{CFU} = \text{CFU-TIN} \end{array} \right) \right) \Delta CFU-TIN$$

3)

$$\prod_{\text{TITOLO}} \left( \bigcap_{\text{NATTIV}} \left( \begin{array}{l} \text{CORSO} \propto \text{ATTIVAZIONE} \\ \text{CODICE} = \text{CODICE-CORSO} \\ \text{ANNOA} = \text{YEAR(SUSSDATO)} \end{array} \right) \right)$$

$$\prod_{\text{TITOLO}} \left( \text{TITOLO} \text{ F MAX(NATTIV)}(F3) \right)$$

4)

$$NO \Leftarrow \prod_{\text{CODICE}, \text{TITOLO}, \text{CFU}, \text{SSD}} \left( \bigcap_{\text{ANNOA} > 1994 \text{ AND } \text{ANNOA} < 2001} \left( \begin{array}{l} \text{CORSO} \propto \text{ATTIVAZIONE} \\ \text{CODICE} = \text{CODICE-CORSO} \end{array} \right) \right)$$

$$\prod_{\text{TITOLO}} (\text{CORSO}, \text{NO})$$

5)

$$\prod_{\text{TITOLO}} \left( \bigcap_{\text{ANNOA} < \text{ANNOB}} \left( \begin{array}{l} A2 \propto A1 \\ \text{A1.CODICE} = \text{A2.CODICE} \end{array} \right) \right)$$

$$A1 \Leftarrow \left( \bigcap_{\text{ANNOA} > 1994 \text{ AND } \text{ANNOA} < 2001} \left( \begin{array}{l} \text{CORSO} \propto \text{ATTIVAZIONE} \\ \text{CODICE} = \text{CODICE-CORSO} \end{array} \right) \right)$$

6)  $\overline{\Pi} \left( \delta \left( \left( \begin{array}{l} \text{CORPO} \bowtie \text{ATTIVAZIONE} \\ \text{CODICE} = \text{CODICECORPO} \end{array} \right) \bowtie \text{VINCOLO} \right) \right)$ , CORSI VINC PROPI

$\text{TIPO} = 'PROPRÉTÉ'$   
 $\text{COD-CORSO}, \text{COD-VINCOLANTE}, \text{ANNO}$

NO O -  $\overline{\Pi} \left( \begin{array}{l} \text{CORSI VINC PROPI} \bowtie \text{ATTIVAZIONE} \\ \text{ANNOA} = \text{ATT-ANNOA} \text{ AND } \text{COD-VINCOLANTE} = \text{CODICECORPO} \end{array} \right)$

$\text{COD-CORSO},$

SI  $\left( \overline{\Pi} \left( \begin{array}{l} \text{CORSI VINC PROPI} \\ \text{COD-CORSO} \end{array} \right) \right) \setminus NO \quad \overline{\Pi} \left( \begin{array}{l} SI \bowtie \text{corso} \\ \text{TITOLO} \end{array} \right)$

▽  $\left( \delta \left( \begin{array}{l} \text{VINCOLO} \bowtie V \\ \text{COD-CORSO}, \text{CV} \text{ AND } \text{TIPO} = T \end{array} \right) \right)$   
 $\text{TIPO} = 'PROPRÉTÉ UTICITA'$

$V \in \rho(\text{VINCOLO})$   
 $CC, CV, T$

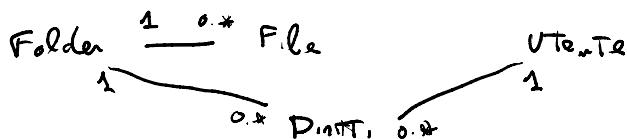
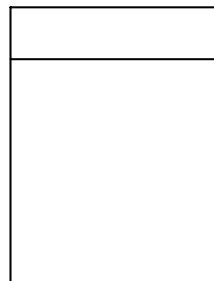
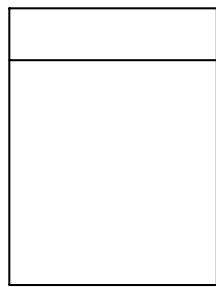
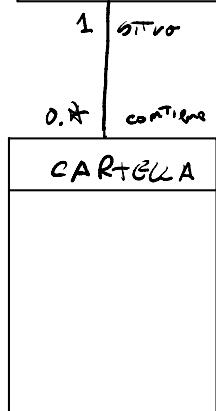
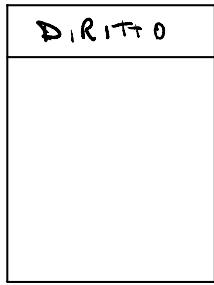
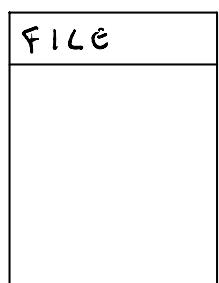
NON È POSSIBILE PARLO ALL'INFINTO

Scriva una interrogazione in algebra relazionale che se valutata fornisce nome, cognome, cruciverba completi

Diritti

File

Cartelle



$\text{FOLDER}(\text{CodFO}, \text{Nome}, \text{Path}, \text{Dimensione}, \text{FolderContenente})$   
 $\text{FILE}(\text{CodFI}, \text{Nome}, \text{CodFO}, \text{Path}, \text{DataC}, \text{DataM}, \text{Dimensione})$   
 $\text{UTENTE}(\text{CodU}, \text{Nome}, \text{Cognome}, \text{DataN})$   
 $\text{DIRITTI}(\text{CodFO}, \text{CodU}, \text{Operazione})$   
 $\text{LOG}(\text{CodOp}, \text{Utente}, \text{Operazione}, \text{File}, \text{Time})$

**Esercizio 1**

**Esercizio 01 (Punti 7)** Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce il nome e il pathname del file sul quale nel corso dell'anno 2015 sono state fatte il maggior numero di modifiche in scrittura (Si usi la funzione YEAR per estrarre l'anno dal timestamp).

$\text{FICROLOR} \Delta - 6 \text{ OPERAZIONE} = 'W' \text{ AND } \text{YEAR}(\text{TIME}) = 2015 \text{ (LOG)}$

$\text{CONTA} \leftarrow \sum_{\text{codFile}} (\text{FILE} \sum_{\text{numero-operazioni}} \text{COUNT}(\text{FILE})) \text{ (FICROLOR-LOG)}$

$\text{MAS} \Delta - \text{codFile} \sum_{\text{numero-operazioni}} \text{RAK}(\text{numero-operazioni}) \text{ (CONTA)}$

$\overline{\prod}_{\text{Nome}, \text{PATH}} \left( \text{MAS} \Delta \text{FILE} \right) \text{ codFile} = \text{codFI}$

**Esercizio 02** (Punti 9) Si implementino nel modo più adeguato i seguenti vincoli:

1. I file che stanno nello stesso folder devono avere nomi diversi.
2. La dimensione di un folder deve essere pari alla somma delle dimensioni di tutti i file e di tutti i folder contenuti.
3. Quando viene inserito un file nella tabella file il suo campo path viene aggiornato dopo l'inserimento in modo che risulti essere la concatenazione del path della cartella contenente e del nome del file (col tradizionale separatore).

1) Poniamo nella Tabella File, due colonna unique, ovvero Nome e CodFO. Così due file che hanno nome uguale e sono in due cartelle diverse non si creano conflitti..

ALTER TABLE FILE ADD CONSTRAINT UC\_FILE\_FOLDER UNIQUE (Nome, CodFO)

2)

```
ALTER TABLE FOLDER
ADD CONSTRAINT CK_SIZE_FOLDER
CHECK F.DIMENSIONE =
(SELECT SUM(DIMENSIONE) FROM FILE WHERE CODFO = F.CODFO) +
(SELECT SUM(DIMENSIONE) FROM FOLDER WHERE FOLDERCONTENENTE = F.CODFO))
```

3)

```
CREATE TRIGGER PATH_FILE
AFTER INSERT ON FILE
FOR EACH ROW
BEGIN

    UPDATE FILE FI
    SET FI.PATH = ((SELECT PATH FROM FOLDER WHERE CODFO = FI.CODFO) || '/' || NEW.NOME)
    WHERE FI.CODFI = NEW.CODFI

END
```

**Esercizio 03** (Punti 9) Si supponga che sia definita una tabella CONENUTO(CodFO, CodFOContenuto, profondita)

dove CodFOContenuto è un folder contenuto in CodFO e profondita indica la distanza di contenimento nell'albero del file system (ad es. folder in folder ha profondità 1 folder in folder in folder ha profondità 2 etc). Si scriva una procedura che riceve in ingresso il codice di un folder (padre). La procedura inserisce una riga nella tabella CONENUTO per ogni folder contenuto nel folder padre indicando la relativa profondità.

```
CREATE OR REPLACE PROCEDURE ADD_CONTENUTO
(CODICE_FOLDER IN FOLDER.CODFO%TYPE)

PROFONDITA INTEGER := 0;
CURRENT_FOLDER FOLDER.CODFO%TYPE := CODICE_FOLDER;

BEGIN

SELECT

END
```

```

CREATE OR REPLACE PROCEDURE ADD_CONTENUTO
(CODICE_FOLDER IN FOLDER.CODFO%TYPE)

PROFONDITA INTEGER := 0;
CURRENT_FOLDER FOLDER.CODFO%TYPE := CODICE_FOLDER;

CURSOR C1 IS
SELECT CODFO
FROM FOLDER WHERE FOLDERCONTENENTE = CODICE_FOLDER

BEGIN

SELECT PROFONDITA + 1 INTO PROFONDITA FROM CONTENUTO WHERE CODFOCONTENUTO = CODICE_FOLDER;

OPEN C1;
LOOP
FETCH C1 INTO CURRENT_FOLDER;
EXIT WHEN C1%NOTFOUND;

INSERT INTO CONTENUTO VALUES(CODICE_FOLDER, CURRENT_FOLDER, PROFONDITA);
END LOOP;
CLOSE C1;

END;

```

**Esercizio 04 (Punti 7)** Utilizzando SQL dinamico si scriva una procedura che riceve in ingresso una stringa di caratteri contenente una sequenza di codici di file separati da ;. La procedura restituisce una sequenza di codici utente (senza duplicazioni) per utenti che hanno modificato almeno un file tra quelli indicati nella stringa di ingresso.

```

CREATE OR REPLACE PROCEDURE P1
(STRINGA VARCHAR(1000))

CURRENT_CODICE VARCHAR2(10) ;

COMANDO VARCHAR2(2000) := 'SELECT DISTINCT UTENTE FROM LOG
WHERE OPERAZIONE = ''W'' AND FILE IN (';

BEGIN

WHILE INSTR (STRINGA, ',') >= 1
LOOP
POS1 := INSTR(STRINGA, ',');
POS2 := INSTR(STRINGA, ',', 2);
CURRENT_CODICE := SUBSTR(STRINGA, 1, INSTR( POS1 - 1));
IF POS2 >= 1
THEN
STRINGA := SUBSTR(STRINGA, POS2 + 1)
ELSE
STRINGA := "";
END IF

COMANDO := COMANDO || CURRENT_CODICE || ',';

END LOOP;

COMANDO := SUBSTR(COMANDO, LENGTH(COMANDO) - 1);
COMANDO := COMANDO || ')';

EXECUTE IMMEDIATE COMANDO;

END

```

Roma 2020

**Esercizio 01** (Punti 8) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce l'identificativo di utenti autori di post che sono stati commentati SOLO da amici (mai da utenti non inclusi tra gli amici).

$\text{Post}(IdPost, Autore, Data, Ora, Testo)$   
 $\text{Commento}(IdPost, Ordine, Testo, Autore, Data, Ora)$   
 $\text{Key}(\text{parola}, \text{Tema})$   
 $\text{Presenza}(IdPost, \text{Tema})$   
 $\text{Sottoscrizione}(IdUtente, \text{Tema}, \text{Data})$   
 $\text{Notifica}(IdPost, IdUtente, \text{Data}, \text{Ora})$   
 $\text{Amici}(IdUtente1, IdUtente2)$

$$\begin{aligned}
 & \text{AM1} \quad s \leftarrow \text{commento} \setminus \text{AM1} \\
 & \quad \text{Autore} = \text{IDUTENTE2} \\
 & \quad \text{ID POST} = \text{ID POST} \\
 & \quad \text{AM1} \quad s \leftarrow \left( \overline{\Pi} \left( \begin{array}{l} \text{Post} \setminus \text{AMICI} \\ \text{Autore} = \text{IDUTENTE2} \\ \text{AMICO} = \text{IDUTENTE2} \end{array} \right) \right) \cup \left( \overline{\Pi} \left( \begin{array}{l} \text{Post} \setminus \text{AMICI} \\ \text{Autore} = \text{IDUTENTE2} \\ \text{AMICO} = \text{ID POST} \end{array} \right) \right) \\
 & \quad \text{NO} \leftarrow \overline{\Pi} \left( \begin{array}{l} \text{Post} \setminus (\text{commento} \setminus \text{com_amici}) \\ \text{ID POST} = \text{ID POST} \end{array} \right) \\
 & \quad \text{ID POST, Autore, Data, Ora, Testo} \\
 & \quad \text{SI} \quad s \leftarrow \text{Post} \setminus \text{NO} \\
 & \quad \overline{\Pi}_{\text{AUTORE}}(\text{SI})
 \end{aligned}$$

**Esercizio 02** (Punti 8) Si scriva una vista che per ogni utente ed ogni mese fornisca un riepilogo dell'attività dell'utente del tipo

(Utente, anno, mese, N\_post, N\_nocom, N\_com\_amici, N\_com\_nonamici). (N\_post il numero di post dell'utente nel mese, N\_com\_amici il numero di commenti di amici, N\_com\_nonamici il numero di commenti di utenti non amici, N\_nocom numero di post senza commenti)

```

CREATE VIEW RIEPILOGO
AS
SELECT P.AUTORE AS UTENTE, P.YEAR(DATA) AS ANNO, P.MONTH(DATA) AS MESE,
FROM POST P
LEFT JOIN (SELECT COUNT(*) AS N_POST FROM POST WHERE AUTORE = P.AUTORE AND YEAR(DATA) = P.YEAR(DATA) AND MONTH(DATA) = P.MONTH(DATA))
LEFT JOIN
(SELECT COUNT(*) AS N_NOCOM FROM (SELECT COUNT(IDPOST) FROM POST NATURAL JOIN COMMENTO WHERE AUTORE = P.AUTORE AND YEAR(DATA) = P.YEAR(DATA) AND MONTH(DATA) = P.MONTH(DATA)) HAVING COUNT(IDPOST) = 0)
LEFT JOIN
(SELECT SUM(N_COM) AS N_COM_AMICI FROM (
SELECT COUNT(IDPOST) AS N_COM FROM POST NATURAL JOIN COMMENTO C JOIN AMICI ON (C.AUTORE = IDUTENTE1 AND AUTORE = IDUTENTE2) OR (C.AUTORE = IDUTENTE2 AND AUTORE = IDUTENTE1) WHERE AUTORE = P.AUTORE AND YEAR(DATA) = P.YEAR(DATA) AND MONTH(DATA) = P.MONTH(DATA) ))
LEFT JOIN
(SELECT SUM(N_COM) AS N_COM_NONAMICI FROM (
SELECT COUNT(IDPOST) AS N_COM FROM POST NATURAL JOIN COMMENTO C LEFT JOIN AMICI WHERE IDUTENTE1 IS NULL AND IDUTENTE2 IS NULL AND AUTORE = P.AUTORE AND YEAR(DATA) = P.YEAR(DATA) AND MONTH(DATA) = P.MONTH(DATA) ))
GROUP BY (P.AUTORE, P.YEAR(DATA), P.MONTH(DATA))

```

**Esercizio 03** (Punti 8) Si scriva un trigger che, quando viene inserito un nuovo post, controlla se tra le parole del testo sono presenti delle parole chiave (parole presenti nella tabella Key). Se una parola chiave è presente deve essere inviata una notifica (aggiunta una riga nella tabella Notifica) a tutti gli utenti interessati al tema collegato. Una notifica deve essere inviata anche a tutti gli amici dell'autore del post.

```

CREATE TRIGGER CK1
AFTER INSERT ON POST
FOR EACH ROW

DECLARE

TESTO VARCHAR(1000):= NEW.TESTO;
WORD VARCHAR(100);
TEM VARCHAR(100);
UTEN VARCHAR(100);

CURSOR C1 IS
SELECT IDUTENTE FROM SOTTOSCRIZIONE WHERE TEMA LIKE TEM;

CURSOR A1 IS
SELECT IDUTENTE1 FROM AMICI JOIN POST ON AUTORE = IDUTENTE2;

CURSOR A2 IS
SELECT IDUTENTE2 FROM AMICI JOIN POST ON AUTORE = IDUTENTE1;

BEGIN

WHILE TESTO <> ''
LOOP

WORD := SUBSTR(TESTO, 1, INSTR(' ', 1));
TESTO := SUBSTR(TESTO, INSTR(' ', 1));

IF EXIST (SELECT * FROM KEY WHERE PAROLA LIKE WORD)
THEN
SELECT TEMA INTO TEM FROM KEY WHERE PAROLA LIKE WORD;

OPEN C1;
LOOP
FETCH C1 INTO UTEN;

INSERT INTO NOTIFICA VALUES(NEW.IDPOST, UTEN, SYSDATE(), SYSTIME());

END LOOP;
CLOSE C1;

OPEN A1;
LOOP
FETCH A1 INTO UTEN;

INSERT INTO NOTIFICA VALUES(NEW.IDPOST, UTEN, SYSDATE(), SYSTIME());

END LOOP;
CLOSE A1;

OPEN A2;
LOOP
FETCH A2 INTO UTEN;

INSERT INTO NOTIFICA VALUES(NEW.IDPOST, UTEN, SYSDATE(), SYSTIME());

END LOOP;
CLOSE A2;

END IF

END LOOP

```

END

**Esercizio 04** (Punti 8) Si scriva una funzione che prende in ingresso il codice di un post e un parametro k intero e che restituisce in una stringa separati da virgole i codici dei k post più simili al post dato. Il grado di somiglianza tra due post è dato dal numero di parole chiave che hanno in comune.

```
CREATE OR REPLACE FUNCTION F1 ( CODICEPOST POST.IDPOST%TYPE, K INTEGER )
RETURN VARCHAR(200)
```

```
DECLARE
```

```
PAROL VARCHAR(100);
```

```
PRESenza PR JOIN KEY K ON PR.TEMA = K.TEMA
```

```
CURSOR C1 IS
SELECT K.PAROLA
FROM PRESenza PR JOIN KEY K ON PR.TEMA = K.TEMA
WHERE PR.IDPOST = CODICEPOST;
```

```
BEGIN
```

```
OPEN C1;
```

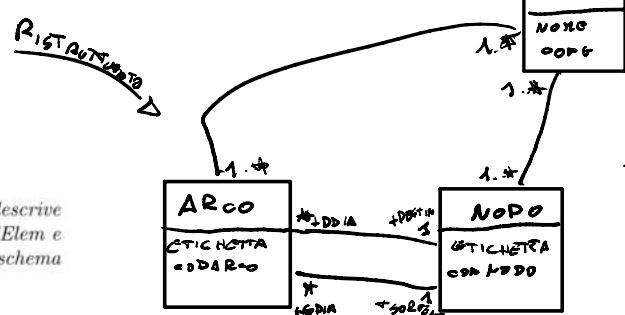
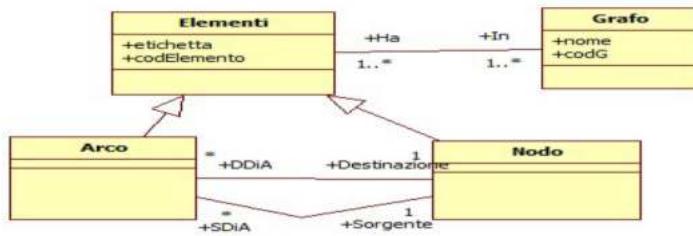
```
LOOP
```

```
FETCH C1 INTO PAROL
```

```
END LOOP;
CLOSE C1;
```

```
END
```

X  
Parole chiave del Post Principale



**Esercizio 01** (Punti 9, 30 minuti) Si consideri la bozza di Class Diagram in figura che descrive grafi composti di nodi e di archi (un arco ha un nodo sorgente e uno destinazione). CodElem e CodG sono attributi chiave. La specializzazione è totale e disgiunta. Si scriva prima lo schema logico per il class diagram e si definiscano poi le tabelle.

GRAFO(C NOME, COD G )

ARCO( CODARCO , ETICHETTA, CODICE-NODO-DDIA, CODICE-NODO-SDIA )

NODO(COD NODO, ETICHETTA)

ASS-G-A ( CODARCO, COD G )

ASS-G-N(CODNODO, COD G )

## ASS-G-N(CODNODO, CODG)

```
CREATE TABLE GRAFO (
    NOME VARCHAR(100) NOT NULL,
    CODG INT NOT NULL,
    CONSTRAINT PK1 PRIMARY KEY(CODG)
);

CREATE TABLE ARCO (
    CODARCO INT NOT NULL,
    ETICHETTA VARCHAR(200) NOT NULL,
    CODICE_NODO_DDIA INT NOT NULL,
    CODICE_NODO_SDIA INT NOT NULL,
    CONSTRAINT PK1 PRIMARY KEY(CODARCO),
    CONSTRAINT FK_ARCO_NODO_DDIA FOREIGN KEY (CODICE_NODO_DDIA) REFERENCES
    NODO(CODNODO),
    CONSTRAINT FK_ARCO_NODO_SDIA FOREIGN KEY (CODICE_NODO_SDIA) REFERENCES
    NODO(CODNODO)
);

CREATE TABLE NODO (
    CODNODO INT NOT NULL,
    ETICHETTA VARCHAR(200) NOT NULL,
    CONSTRAINT PK1 PRIMARY KEY(CODNODO)
);

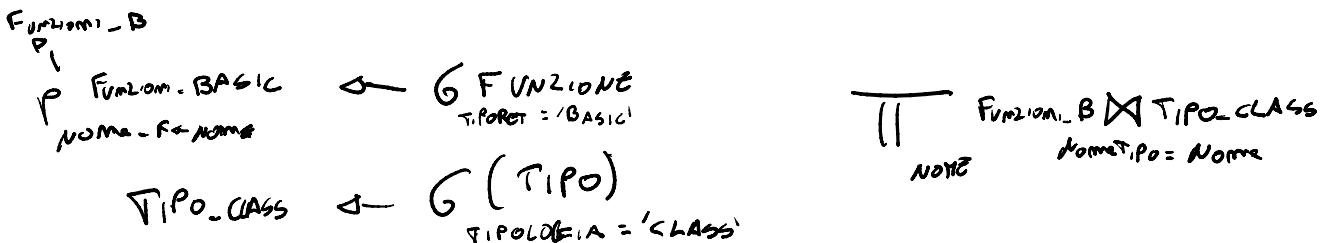
CREATE TABLE ASS_G_A (
    CODARCO INT NOT NULL,
    CODG INT NOT NULL,
    CONSTRAINT FK_A FOREIGN KEY (CODARCO) REFERENCES ARCO(CODARCO),
    CONSTRAINT FK_G FOREIGN KEY (CODG) REFERENCES GRAFO(CODG)
);

CREATE TABLE ASS_G_N (
    CODNODO INT NOT NULL,
    CODG INT NOT NULL,
    CONSTRAINT FK_N FOREIGN KEY (CODNODO) REFERENCES NODO(CODNODO),
    CONSTRAINT FK_G FOREIGN KEY (CODG) REFERENCES GRAFO(CODG)
);
```



$TIPO(Nome, Tipologia, DataDef, DataVar, TipoArr, Dim, SupClass)$   
 $ATTRIBUTO(NomeTipo, NomeAtt, TipoAtt, ValDef, Posizione)$   
 $FUNZIONE(CodF, NomeTipo, DataDef, DataVar, TipoRet, Nome, Posizione)$   
 $PARAM(CodF, nome, TipoPar, Posizione)$

**Esercizio 01** (Punti 7) Si scriva una interrogazione in algebra relazionale che, se valutata, restituisca il Nome dei tipi di tipologia Class con associati SOLO funzioni il cui valore di ritorno è di tipo BASIC.



**Esercizio 02** (Punti 8) Si scriva in SQL una interrogazione che restituisca COPPIE di nomi di tipi di tipologia Class che hanno la stessa struttura di attributi (vale a dire, per ogni posizione il nome e tipo degli attributi sono gli stessi) e che hanno almeno una funzione di nome diverso.

```

SELECT T.NOME, A.NOMEATT
FROM TIPO T
JOIN ATTRIBUTO A ON A.NOMETIPO = T.NOME
JOIN FUNZIONE F ON F.NOMETIPO = T.NOME
WHERE T.TIPOLOGIA LIKE 'CLASS'
AND A.NOMEATT = A.TIPOATT
AND F.NOME <> A.NOMEATT
    
```

**Esercizio 03** (Punti 7) Si implementino nel modo più opportuno i seguenti vincoli:

- Quando viene inserita una nuova funzione si deve aggiornare anche la DataVar del tipo class associato se la DataDef della funzione è successiva alla DataVar del tipo (il nuovo valore sarà quello della DataDef della funzione).
- Ci possono essere associazioni di funzioni ad un tipo solo se la tipologia del tipo è CLASS;
- Due parametri diversi associati alla stessa funzione non possono avere lo stesso valore di posizione.

3)

```

CREATE ASSERTION CK_POSIZIONE
CHECK NOT EXIST (
SELECT * FROM PARAM P1 NATURAL JOIN PARAM P2
WHERE P1.CODF = P2.CODF AND P1.POSIZIONE = P2.POSIZIONE)
    
```

2)

```

CREATE ASSERTION CK_ASSOCIAZIONE_FUNZIONE
CHECK NOT EXIST (
SELECT * FROM FUNZIONE F JOIN TIPO T ON F.NOMETIPO = T.NOME
WHERE T.TIPOLOGIA NOT LIKE 'CLASS')
    
```

1)

```
CREATE TRIGGER INS_FUNZIONE
AFTER INSERT ON FUNZIONE

BEGIN

IF (NEW.DATADEF > (SELECT T.DATAVAR FROM TIPO T WHERE T.NOME = NEW.NOMETIPO))
THEN

UPDATE TIPO
SET DATAVAR = NEW.DATADEF
WHERE NOME = NEW.NOMETIPO;

END IF

END
```

**Esercizio 04** (*Punti 8*) Si scriva una funzione PLSQL che riceve come parametro par1 il nome di un tipo di tipologia classe e parametro par2 il nome di un tipo. La funzione restituisce il numero di funzioni associate a sottoclassi AD OGNI LIVELLO DI PROFONDITA' di par1 (nodi dell'albero delle classi radicato nel nodo par1 dato) che abbiano par2 come tipo di ritorno. Per svolgere la ricerca si assuma di avere una struttura TMP già definita esternamente alla struttura (si scelga la struttura della tabella TMP secondo le proprie necessità).

```
CREATE FUNCTION F1
(PAR1 TIPO.NOME%TYPE, PAR2 TIPO.NOME%TYPE)
RETURN INT

N_FUNZIONI INT := 0;

BEGIN

-- consideriamo TMP tabella esterna che contiene tutti i livelli di profondità delle classi, I
PARAMETRI SONO UGUALI A TIPO

SELECT COUNT(*) INTO N_FUNZIONI
FROM TMP
JOIN FUNZIONE F ON F.NOMETIPO = TMP.NOME
WHERE TMP.NOME = PAR1
AND F.TIPORET = PAR2;

RETURN N_FUNZIONI
END
```

**Esercizio 05** (*Punti 6*) Utilizzando SQL DINAMICO Si scriva una procedura che riceve in ingresso il nome di una tabella e una stringa di nomi di attributi separati dal carattere separatore #. La funzione ha l'effetto di aggiungere alla tabella passata per parametro un vincolo di chiave primaria che coinvolge tutti gli attributi passati nella stringa.

```
CREATE PROCEDURE P1
(TABELLA UNSER_TABLE%TABLE_NAME, STRINGA VARCHAR(1000))

COMANDO VARCHAR(1000) := 'ALTER TABLE ' || TABELLA || ' ADD CONSTRAINT PK PRIMARY KEY (';
S VARCHAR(1000) := STRINGA;
```

```

ATTRIBUTO VARCHAR(200) := '';
BEGIN
WHILE S <> ''
LOOP

ATTRIBUTO := SUBSTR(S, 1, INSTR(S, '#', 1));
S := SUBSTR(S, INSTR(S, '#', 1));

COMANDO := COMANDO || ATTRIBUTO || ',';
END LOOP

COMANDO := SUBSTR(COMANDO, 1, LENGTH(COMANDO) - 1);
COMANDO := COMANDO || ';'';

EXECUTE IMMEDIATE COMANDO;

END

```

A

**Esercizio 01** (Punti 7) Si scriva una interrogazione in algebra relazionale che, se valutata, restituisca il Nome dei tipi di tipologia Class con associati SOLO attributi di tipo BASIC

$$TP \leftarrow \{ (TPO) \mid TPO_{tipologia} = 'CLASS' \}$$

$$ATT \leftarrow \{ (ATTRIBUTO) \mid TPO_{ATTRIBUTO} = 'BASIC' \}$$

$$\overline{\prod}_{NOME\_TPO} TP \bowtie ATT \quad NOME = NOME\_TPO$$

**Esercizio 02** (Punti 8) Si scriva in SQL una interrogazione che restituisca COPPIE di nomi di funzioni associate a classi diverse ma che hanno la stessa firma (vale a dire, stesso nome, stesso tipo di ritorno, per ogni posizione il nome e tipo del parametro sono gli stessi).

```

SELECT F1.NOME, F2.NOME
FROM FUNZIONE F1
JOIN FUNZIONE F2 ON F1.NOME = F2.NOME AND F1.TIPORET = F2.TIPORET
JOIN PARAM P ON F1.POSIZIONE = P.POSIZIONE AND F2.POSIZIONE = P.POSIZIONE

WHERE F1.CODF <> F2.CODF
AND F1.NOMETIPO <> F2.NOMETIPO
AND F1.POSIZIONE = F2.POSIZIONE
AND P.NOME = P.TIPOPAR

```

**Esercizio 03** (Punti 7) Si implementino nel modo più opportuno i seguenti vincoli:

1. Quando viene variata la DataVar di una funzione si deve aggiornare anche la DataVar del tipo class associato se la DataVar della funzione è successiva alla DataVar del tipo.
2. Ci possono essere associazioni di attributi ad un tipo solo se la tipologia del tipo è CLASS ;
3. Due funzioni diverse associate allo stesso tipo non possono avere lo stesso valore di posizione.

3)

```
CREATE ASSERTION CK_POSIZIONE_F
CHECK NOT EXIST (
SELECT * FROM FUNZIONE F1
JOIN FUNZIONE F2 ON F1.NOMETIPO = F2.NOMETIPO
WHERE F1.CODF <> F2.CODF
AND F1.POSIZIONE = F2.POSIZIONE
)
```

2)

```
CREATE ASSERTION CK_TIPOATTRIBUTI
CHECK NOT EXIST (
SELECT * FROM ATTRIBUTO A
JOIN TIPO T ON A.NOMETIPO = T.NOME
WHERE T.TIPOLOGIA NOT LIKE 'CLASS'
)
```

1)

```
CREATE TRIGGER UPD_FUNZ
AFTER UPDATE OF DATAVAR ON FUNZIONE
BEGIN
IF NEW.DATAVAR > (SELECT T.DATAVAR FROM TIPO T WHERE T.NOME = OLD.NOMETIPO)
THEN
UPDATE TIPO
SET DATAVAR = NEW.DATAVAR
WHERE NOME = OLD.NOMETIPO;
END IF
END
```

**Esercizio 04** (Punti 8) Si scriva una funzione PLSQL che riceve come parametro il nome di un tipo di tipologia classe. La funzione restituisce il numero di funzioni associate a sottoclassi AD OGNI LIVELLO DI PROFONDITA' del tipo dato (nodi dell'albero delle classi radicato nel nodo dato). Per svolgere la ricerca si assuma di avere una struttura TMP già definita esternamente alla struttura (si scelga la struttura della tabella TMP secondo le proprie necessità).

```
CREATE FUNCTION F1
(NAME TIPO.NOME%TYPE)
RETURN INT
VAR INT := 0;
BEGIN
```

```
-- LA STRUTTURA TMP CONTIENE LE INFORMAZIONI DELL'ALBERO
```

```
SELECT COUNT(*) INTO VAR
FROM TMP JOIN FUNZIONE F ON TMP.NOME = F.NOMETIPO
WHERE TMP.NOME = NAME;
```

```
RETURN VAR
END
```

**Esercizio 05** (*Punti 6*) Utilizzando SQL DINAMICO Si scriva una procedura che riceve in ingresso il nome di una tabella e una stringa di nomi di attributi separati dal carattere separatore #. La funzione ha l'effetto di aggiungere alla tabella passata per parametro un (unico) vincolo di unicità che coinvolge tutti gli attributi passati nella stringa.

```
CREATE PROCEDURE P1
(TABELLA UNSER_TABLE%TABLE_NAME, STRINGA VARCHAR(1000))
```

```
ATTRIBUTO VARCHAR(200) := '';
S VARCHAR(1000) := STRINGA;
COMANDO := 'ALTER TABLE ' || TABELLA || ' ADD CONSTRAINT U1 UNIQUE( ';
```

```
BEGIN
WHILE S <> ''
LOOP
```

```
ATTRIBUTO := SUBSTR(S, 1, INSTR(S, 1, '#'));
S := SUBSTR(S, INSTR(S, 1, '#') + 1);
```

```
COMANDO := COMANDO || ATTRIBUTO || ',';
```

```
END LOOP
```

```
COMANDO := SUBSTR(COMANDO, 1, LENGTH(COMANDO) - 1);
COMANDO := COMANDO || ');';
```

```
EXECUTE IMMEDIATE COMANDO;
```

```
END
```

*STUDENTE(Matricola, Nome, Cognome)  
 PIANI(CodPiani, Matricola, Data)  
 COMPOSIZIONE(CodPiano, CodEsame, Anno)  
 ESAME(CodEsame, Titolo, CFU, Tipo)  
 ANNI(CodEsame, Anno)  
 PROPED(CodEsame, CodEsameProp).*

**Esercizio 01** (punti 8) Si scriva una espressione in algebra relazionale che, se valutata, fornisca il nome e cognome e la matricola degli studenti che hanno un piano di studi dove tutti gli insegnamenti presenti non hanno richiesta di propedeuticità.

$\overline{\exists} (\text{STUDENTE} \bowtie \text{PIANI} \bowtie \text{COMPOSIZIONE} \bowtie (\text{ESAMI} \setminus (\text{ESAMI} \bowtie \text{PROPED})))$   
NOME, COGNOME, MATRICOLA

**Esercizio 02** (punti 8) Definire nel modo più semplice i seguenti vincoli:

1. lo stesso esame non deve essere associato più di una volta allo stesso piano di studi;
2. nella tabella PROPED ad un esame possono essere associati solo esami effettivamente presenti nella tabella ESAMI;
3. se un esame è presente in un piano di studi devono essere presenti anche tutti gli esami propedeutici;
4. la somma dei CFU presenti in ogni anno (I, II e III) deve essere esattamente 60 CFU.

1)

ALTER TABLE COMPOSIZIONE  
 ADD CONSTRAINT UK1 UNIQUE (CODPIANO, CODESAME)

2)

ALTER TABLE PROPED  
 ADD CONSTRAINT CK1 CHECK EXIST (SELECT \* FROM ESAME E WHERE E.CODESAME = CODESAME)  
 ADD CONSTRAINT CK2 CHECK EXIST (SELECT \* FROM ESAME E WHERE E.CODESAME = CODESAMEPROP)

4)

CREATE ASSERTION A1  
 CHECK (SELECT SUM(E.CFU) FROM ESAME E NATURAL JOIN ANNI A) = 60

3)

CREATE ASSERTION A2  
 CHECK  
 (SELECT COUNT(\*) FROM COMPOSIZIONE C  
 NATURAL JOIN PROPED) -  
 (SELECT COUNT(\*) FROM PROPED P  
 JOIN COMPOSIZIONE C ON C.CODESAME = P.CODESAMEPROP)  
 = 0

**Esercizio 03** (Punti 8) Si scriva una trigger che viene attivato quando viene inserito un nuovo studente. Il trigger crea un piano di studi di default per lo studente. Supponendo che CodPiano in PIANI sia un intero, il codice del piano dello studente è dato dall'intero successivo del massimo CodPiano presente nella tabella. La data è quella corrente ottenibile con la funzione SYSDATE. Per la composizione del piano di studi si usino tutti gli esami obbligatori e tutti gli esami di default. Se un esame può essere collocato in anni diversi, nel piano di studi viene collocato nel primo anno possibile.

CREATE TRIGGER T1  
 AFTER INSERT ON STUDENTE  
 BEGIN

INDICEPIANO INT := (SELECT MAX(CODPIANI) FROM PIANI) + 1;

```

INSERT INTO PIANI P VALUES(
INDICEPIANO,
NEW.MATRICOLA,
SYSDATE()));

INser INTO COMPOSIZIONE C
(SELECT INDICEPIANO, E.CODESAME, MIN(A.ANNO) FROM ESAME E
NATURAL JOIN ANNI A
WHERE (E.TIPO LIKE 'OBBLIGATORIO' OR E.TIPO LIKE 'DEFAULT')
GROUP BY(E.CODESAME));

END

```

**Esercizio 04** (*Punti 8*) Si scriva una funzione in **SQL dinamico** che permette la costruzione di interrogazioni parametriche su tabelle. La funzione riceve in ingresso tre parametri: (a) il nome di una tabella; (b) il nome di un attributo della tabella da selezionare (si supponga che il tipo dell'attributo sia VARCHAR); (c) una lista di coppie nomeattributo-valore in cui gli elementi sono separati da virgolette (es. 'nome,pippo,cognome,pluto,citta,napoli' etc.). La funzione costruisce la selezione sulla tabella (primo parametro) selezionando l'attributo di selezione (secondo parametro) imponendo che per ogni coppia nomeattributo-valore valga nomeattributo=valore. Il risultato della funzione è una stringa che concatena i valori restituiti dalla selezione in ordine crescente (i valori sono separati da virgolette).

```

CREATE FUNCTION F1
(A UNSER_TABLE%TABLE_NAME, B VARCHAR(100), C VARCHAR(1000))
RETURN VARCHAR(1000)

COMANDO VARCHAR(1000) := 'SELECT ' || B || ' FROM ' || A || ' WHERE ';
ATTRIBUTO VARCHAR(100);
VALORE VARCHAR(100);
RIGA VARCHAR(100);
RISULTATO VARCHAR(1000) := '';

STRINGA VARCHAR(1000) := C;

CURSOR C1 REF SYSCURSOR;

BEGIN

WHILE STRINGA <> ''
LOOP

ATTRIBUTO := SUBSTR(STRINGA, 1, INSTR(STRINGA, 1, ','));
VALORE := SUBSTR(STRINGA, INSTR(STRINGA, 1, ',') + 2, INSTR(STRINGA, 1, ',', 2));
STRINGA := SUBSTR(STRINGA, INSTR(STRINGA, 1, ',', 2) + 2);

COMANDO := COMANDO || ATTRIBUTO || '=' || VALORE || ' AND ';

END LOOP

COMANDO := SUBSTR(COMANDO, 1, LENGTH(COMANDO) - 4);

COMANDO := COMANDO || ' ORDER BY ' || B || ' ASC;';

OPEN C1 USING COMANDO
WHILE C1%FOUND
LOOP
FETCH C1 INTO RIGA
RISULTATO := RISULTATO || RIGA || ',';

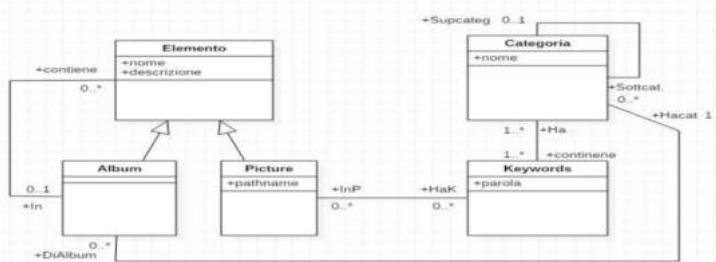

```

END LOOP

RISULTATO = SUBSTR(RISULTATO, 1, LENGTH(RISULTATO) - 1);

RETURN RISULTATO;

END



- Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (6 punti);
- Si forniscano gli schemi relazionali per il class diagram ristrutturato (6 punti);
- Si definisca usando SQL le tabelle (4 punti);
- Nella definizione delle tabelle si scrivano esplicitamente i vincoli (a) elementi nello stesso album hanno nomi diversi; (b) un album non può essere contenuto in se stesso. (4 punti)

ALBUM (CODALBUM, NAME, DESCRIZIONE, INALBUM, IDCATEGORIA)

PICTURE (CODPICTURE, NAME, DESCRIZIONE, PATHNAME, INALBUM)

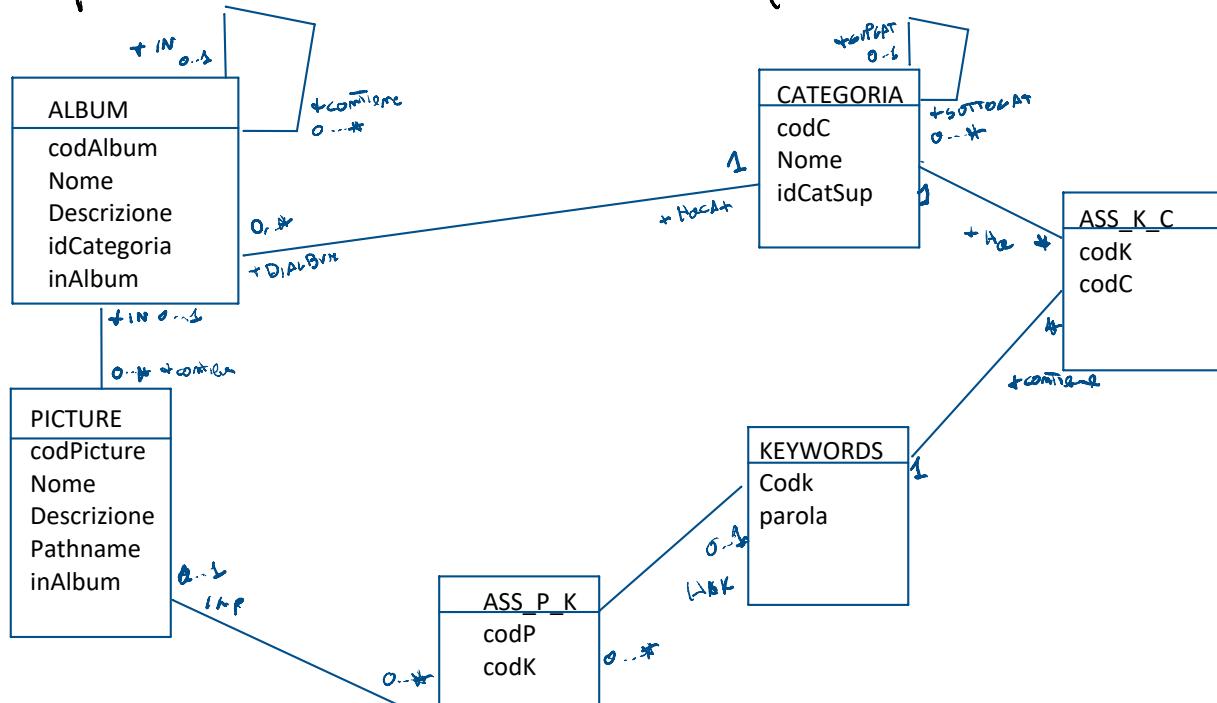
ASS\_P\_K (CODP, CODK)

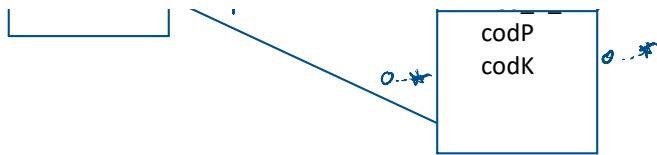
KEYWORDS (CODK, PAROLA)

ASS\_C\_K (CODC, CODC)

CATEGORIA (CODC, NAME, IDCATSUP)

INALBUM è facoltativo sia in ALBUM che in PICTURE. In ASS\_P\_K entrambi le chiavi esterne sono facoltative. In CATEGORIA IDCATSUP C'è facoltativo.





```

CREATE TABLE ALBUM (
CODALBUM INT NOT NULL,
NOME VARCHAR(100) NOT NULL,
DESCRIZIONE VARCHAR(1000) NOT NULL,
INALBUM INT NULL,
IDCATEGORIA INT NOT NULL,
CONSTRAINT PK1 PRIMARY KEY(CODALBUM),
CONSTRAINT FK_ALBUM FOREIGN KEY (INALBUM) REFERENCES ALBUM(CODALBUM),
CONSTRAINT U1 UNIQUE (NOME, INALBUM),
CONSTRAINT CK1 CHECK CODALBUM <> INALBUM
);

```

```

CREATE TABLE PICTURE (
CODPICTURE INT NOT NULL,
NOME VARCHAR(100) NOT NULL,
DESCRIZIONE VARCHAR(1000) NOT NULL,
PATHNAME VARCHAR(200) NOT NULL,
INALBUM INT NULL,
CONSTRAINT PK1 PRIMARY KEY(CODPICTURE),
CONSTRAINT FK_P_ALBUM FOREIGN KEY (INALBUM) REFERENCES ALBUM(CODALBUM),
CONSTRAINT U1 UNIQUE (NOME, INALBUM)
);

```

```

CREATE TABLE KEYWORDS (
CODK INT NOT NULL,
PAROLA VARCHAR(100) NOT NULL,
CONSTRAINT PK PRIMARY KEY (CODK)
);

```

```

CREATE TABLE CATEGORIA (
CODC INT NOT NULL,
NOME VARCHAR(100) NOT NULL,
IDCATSUP INT NULL,
CONSTRAINT PK PRIMARY KEY (CODC),
CONSTRAINT FK_CAT_CAT FOREIGN KEY (IDCATSUP) REFERENCES CATEGORIA(CODC)
);

```

```

CREATE TABLE ASS_P_K (
CODP INT NOT NULL,
CODK INT NOT NULL,
CONSTRAINT FK_P FOREIGN KEY (CODP) REFERENCES PICTURE(CODPICTURE),
CONSTRAINT FK_K FOREIGN KEY (CODK) REFERENCES KEYWORDS(CODK)
);

```

```

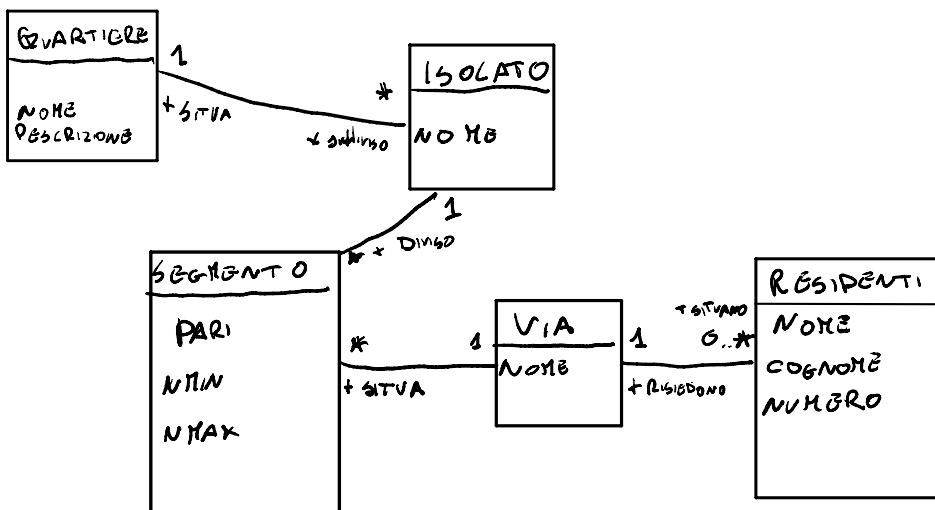
CREATE TABLE ASS_C_K (
CODC INT NOT NULL,
CODK INT NOT NULL,
CONSTRAINT FK_C FOREIGN KEY (CODC) REFERENCES CATEGORIA(CODC),
CONSTRAINT FK_K FOREIGN KEY (CODK) REFERENCES KEYWORDS(CODK)
);

```

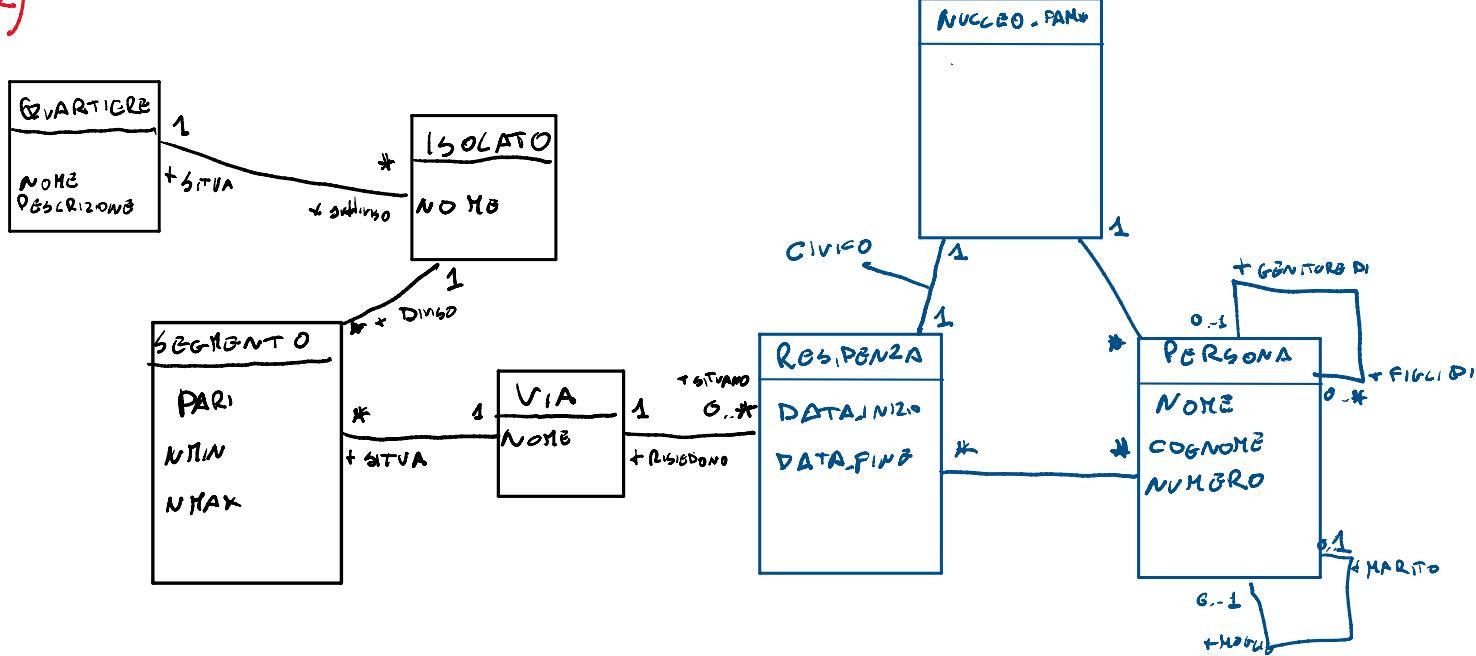
`QUARTIERE(CodQ, Nome, Descrizione), ISOLATO(CodI, CodQ, Nome)`  
`VIA(CodV, Nome), SEGMENTO(CodS, CodV, CodI, Pari, NMin, NMax)`  
`RESIDENTI(CF, Nome, Cognome, CodV, Numero).`

1. Si fornisca un Class Diagram di progettazione (reverse engineering) per lo schema relazionale (5 punti);
2. Si estenda il Class diagram per tenere traccia: dei cambiamenti di residenza di una persona (la residenza ha un inizio, una fine e una persona può avere più residenze); dei nuclei familiari (insiemi di persone che vivono insieme nella stessa unità abitativa associata ad un numero civico); dei legami marito moglie, e genitore figlio dei residenti (5 punti).
3. Per il class diagram fornito, si dia un esempio di vincolo di dominio, uno di ennupla, uno intrarelazionale ed uno interrelazionale. (3 punti).

1)



2)



3)

VINCOLO PI DOMINIO

ALTER TABLE SEGMENTO  
ADD CONSTRAINT CK1 CHECK NMIN > 0;

VINCOLO PI DOMINIO

ALTER TABLE SEGMENTO  
ADD CONSTRAINT CK2 CHECK NMAX > NMIN;

VINCOLO INTRA RELAZIONALI

ALTER TABLE SEGMENTO  
ADD CONSTRAINT PK1 PRIMARY KEY (CODS)

VINCOLO INTERRELAZIONALI

ALTER TABLE SEGMENTO  
ADD CONSTRAINT FK1 FOREIGN KEY (CODV) REFERENCES VIA (CODV)  
ADD CONSTRAINT FK2 FOREIGN KEY (CODI) REFERENCES ISOLATO (CODI);

*Foto(CodF, NomeFile, Proprietario, Dimensione, Descrizione, Tempo)  
 TAG(CodF, parola)  
 Key(parola)  
 Classificazione(Parola1, Parola2, Relazione)*

**Esercizio 01** (Punti 8) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce per ogni utente l'identificativo della foto aggiunta più recentemente.

$\prod_{\text{CODE}} G \left( \begin{array}{l} \text{Proprietario} \leq \text{MAX(Tempo)}(\text{Foto}) \\ \text{Proprietario} = \text{Proprietario AND Tempo} = \text{MAX(Temp)} \end{array} \right)$

**Esercizio 02** (punti 8) Si scriva una interrogazione SQL che fornisca coppie di foto della forma  $(f_1, f_2)$  tali che l'insieme di parole chiave associate a  $f_2$  sia un sottoinsieme di quelle associate a  $f_1$ .

```
SELECT F1.*, F2.*  
FROM FOTO F1 NATURAL JOIN FOTO F2  
WHERE  
    (SELECT COUNT(*) FROM TAG T1  
     WHERE T1.CODF = F1.CODF  
     AND T1.PAROLA IN (SELECT PAROLA FROM TAG WHERE CODF = F2.CODF))  
    )  
    =  
    (SELECT COUNT(*) FROM TAG T2 WHERE T2.CODF = F2.CODF);
```

**Esercizio 03** (Punti 8) Si scriva un trigger che quando viene inserita una nuova foto controlla se tra le parole della descrizione sono presenti delle parole chiave (parole presenti nella tabella Key). Se una parola chiave è presente deve essere inserito un tag corrispondente nella tabella TAG.

```
CREATE TRIGGER T1
AFTER INSERT ON FOTO

DESCR VARCHAR(1000) := NEW.DESCRIZIONE;
P VARCHAR(200) := '';

BEGIN

WHILE DESCR <> ''
LOOP

P := SUBSTR(DESCRIPTOR, 1, INSTR(DESCRIPTOR, 1, ' '));
DESCRIPTOR := SUBSTR(DESCRIPTOR, INSTR(DESCRIPTOR, 1, ' ') + 2);

IF EXIST (SELECT * FROM KEY WHERE PAROLA = P)
THEN

INSERT INTO TAG VALUES(NEW.CODF, P);

END IF

END LOOP

END
```

**Esercizio 04** (Punti 8) Si scriva una funzione che prende in ingresso il codice di una foto e un parametro k intero e che restituisce in una stringa separati da virgole i codici delle k foto più simili. La misura di somiglianza tra due foto  $f_1$  e  $f_2$  è dato da  $\frac{|tag(f_1) \cap tag(f_2)|}{\max\{|tag(f_1)|, |tag(f_2)|\}}$  dove  $tag(f)$  indica l'insieme delle parole chiave associate all'immagine f.

```
CREATE FUNCTION
( IDFOTO FOTO.CODF%TYPE, K INT )
RETURN VARCHAR(1000)
```

```
CURSOR C1
IS (
SELECT F.CODF FROM FOTO F
WHERE F.CODF <> IDFOTO
);
```

```
CURSOR C2
IS (
SELECT CODF FROM TEMP
ORDER BY (MISURA) DESC
);
```

```
CFOTO FOTO.CODF%TYPE;
NOMINAT INT;
DENOMIN INT;
MISURA FLOAT;
I INT;
RISULTATO VARCHAR(1000) := ";
```

```
BEGIN
```

```
CREATE TABLE TEMP
(CODF FOTO.CODF%TYPE NOT NULL PRIMARY KEY,
MISURA FLOAT NOT NULL
CONSTRAINT U1 UNIQUE(CODF));
```

```
OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO CFOTO;
```

```
SELECT COUNT(*) INTO NOMINAT
FROM
(SELECT T1.PAROLA FROM TAG T1 ON T1.CODF = IDFOTO)
INTERSECT
(SELECT T2.PAROLA FROM TAG T2 ON T2.CODF = CFOTO)
```

```
SELECT MAX(*) INTO DENOMIN
FROM
(SELECT COUNT(*) FROM TAG T1 ON T1.CODF = IDFOTO)
UNION
(SELECT COUNT(*) FROM TAG T2 ON T2.CODF = CFOTO)
```

```
MISURA := NOMINAT / DENOMIN;
```

```
INSERT INTO TEMP VALUES(CFOTO, MISURA);
```

```
END LOOP
CLOSE C1;
```

```
I := 0;
OPEN C2
WHILE I < K
LOOP
FETCH C2 INTO CFOTO;
```

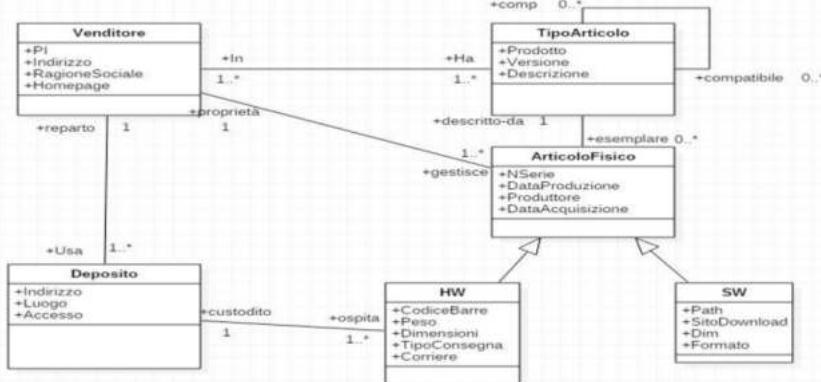
```
RISULTATO := RISULTATO || CFOTO || ',';
```

$| := | + 1;$

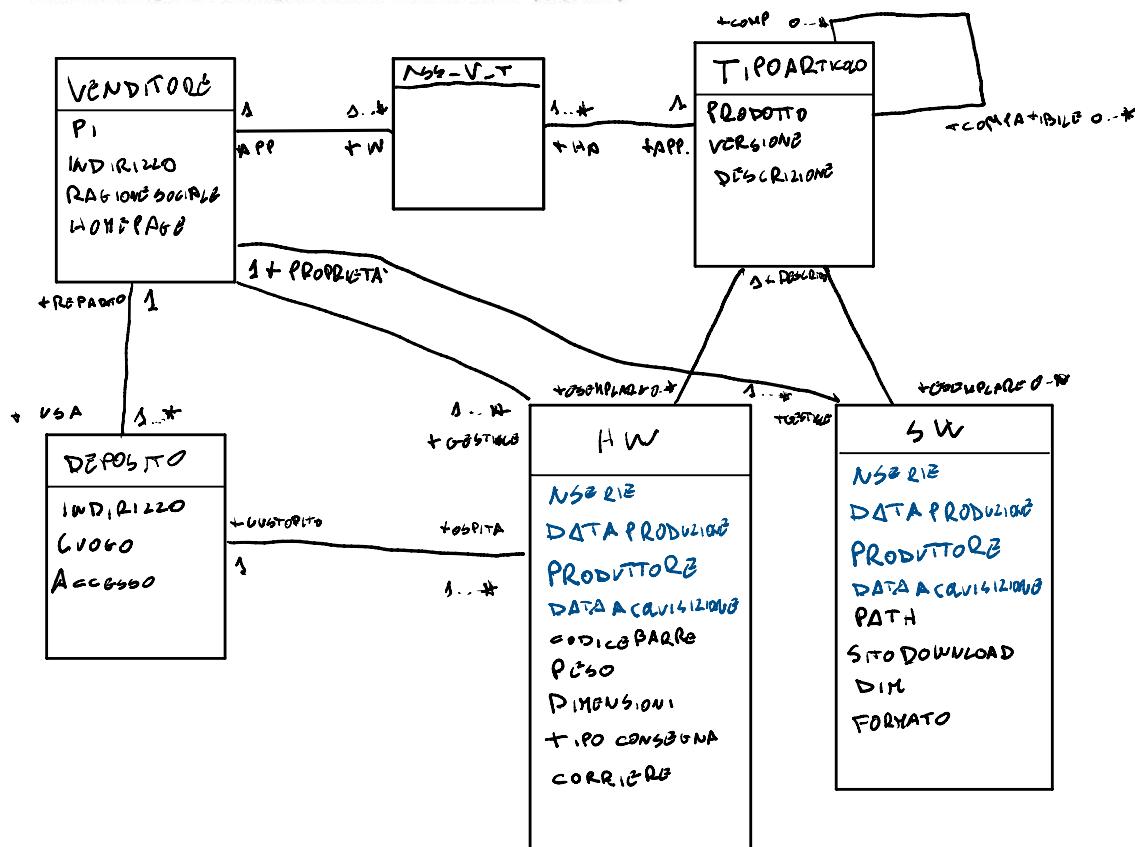
END LOOP

RISULTATO := SUBSTR(RISULTATO, 1, LENGTH(RISULTATO) - 2);

```
RETURN RISULTATO;  
END
```



1. Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (6 punti);
  2. Si forniscano gli schemi relazionali per il class diagram ristrutturato (6 punti);
  3. Si definisca usando SQL le tabelle (4 punti);
  4. Nella definizione delle tabelle si scrivano i vincoli: (a) La data di acquisizione è successiva alla data di produzione; (b) Il tipo di consegna per gli oggetti HW ha i soli valori 'solo ritiro', 'solo spedizione', 'ritiro e spedizione'. Il valore del corriere viene associato se e solo se il tipo di consegna non è 'solo ritiro'. (4 punti)



VENGONO INSERITI VINCOLI DI REFERENCE SULLE CHIAVI ESTERNE. VINCOLI DI UNICITA SU CHIAVI PRIMARIE. LE 2 CHIAVI ESTERNE IN HW E SW DI TIPO ARTICOLO SONO FACOLTATIVE.

VENDITORE(CODV, PI, INDIRIZZO, RAGIONESOCIALE, HOMEPAGE)

DEPOSITO(CODD, INDIRIZZO, LUOGO, ACCESSO, CODV)

HW(NSERIE, DATAPRODUZIONE, PRODUTTORE, DATAACQUISIZIONE, CODICEBARRE, PESO, DIMENSIONI, TIPOCONSEGNA, CORRIERE, CODD, CODV,  
PRODOTTOTA, VERSIONETA)

SW(NSERIE, DATAPRODUZIONE, PRODUTTORE, DATAACQUISIZIONE, PATH, SITODOWNLOAD, DIM, FORMATO, PRODOTTOTA, VERSIONETA)

TIPOARTICOLO(PRODOTTO, VERSIONE, DESCRIZIONE, IDTA)

ASS\_V\_T(CODV, PRODOTTOTA, VERSIONETA)

3)

```
CREATE TABLE VENDITORE (
    CODV INT NOT NULL PRIMARY KEY,
    PI VARCHAR(100) NOT NULL,
    INDIRIZZO VARCHAR(300) NOT NULL,
    RAGIONESOCIALE VARCHAR(300) NOT NULL,
    HOMEPAGE VARCHAR(200) NOT NULL,
    CONSTRAINT U1 UNIQUE (CODV)
);
```

```
CREATE TABLE DEPOSITO (
    CODD INT NOT NULL,
    INDIRIZZO VARCHAR(300) NOT NULL,
    LUOGO VARCHAR(300) NOT NULL,
    ACCESSO VARCHAR(100) NOT NULL,
    CODV INT NOT NULL,
    CONSTRAINT PK1D PRIMARY KEY (CODD),
    CONSTRAINT U1D UNIQUE (CODD),
    CONSTRAINT FK_D_V FOREIGN KEY (COV) REFERENCES VENDITORE (CODV)
);
```

```
CREATE TABLE HW (
    NSERIE INT NOT NULL,
    DATAPRODUZIONE DATE NOT NULL,
    PRODUTTORE VARCHAR(200) NOT NULL,
    DATAACQUISIZIONE DATE NOT NULL,
    CODICEBARRE INT NOT NULL,
    PESO FLOAT NOT NULL,
    DIMENSIONI INT NOT NULL,
    TIPOCONSEGNA VARCHAR(200) NOT NULL,
    CORRIERE VARCHAR(300) NOT NULL,
    CODD INT NOT NULL,
    CODV INT NOT NULL,
    PRODOTTOTA VARCHAR(300) NULL,
    VERSIONETA INT NULL,
    CONSTRAINT PK1HW PRIMARY KEY (NSERIE),
    CONSTRAINT U1HW UNIQUE (NSERIE),
    CONSTRAINT FK_HW_D FOREIGN KEY (CODD) REFERENCES DEPOSITO (CODD),
    CONSTRAINT FK_HW_V FOREIGN KEY (CODV) REFERENCES VENDITORE (CODV),
    CONSTRAINT FK_SW_TA FOREIGN KEY (PRODOTTOTA, VERSIONETA) REFERENCES TIPOARTICOLO (PRODOTTO, VERSIONE),
    CONSTRAINT CK1_HW CHECK DATAACQUISIZIONE > DATAPRODUZIONE,
    CONSTRAINT CK2_HW CHECK TIPOCONSEGNA IN ('SOLO RITIRO', 'SOLO SPEDIZIONE', 'RITIRO E SPEDIZIONE'),
    CONSTRAINT CK3_HW CHECK CORRIERE IS NULL AND TIPOCONSEGNA LIKE 'SOLO RITIRO'
);
```

```
CREATE TABLE SW (
    NSERIE INT NOT NULL,
    DATAPRODUZIONE DATE NOT NULL,
    PRODUTTORE VARCHAR(200) NOT NULL,
    DATAACQUISIZIONE DATE NOT NULL,
```

```

PATH VARCHAR(400) NOT NULL,
SITODOWNLOAD VARCHAR(400) NOT NULL,
DIM INT NOT NULL,
FORMATO VARCHAR(100) NOT NULL,
CODV INT NOT NULL,
PRODOTTOTA VARCHAR(300) NULL,
VERSIONETA INT NULL,
CONSTRAINT PK1SW PRIMARY KEY (NSERIE),
CONSTRAINT U1SW UNIQUE (NSERIE),
CONSTRAINT FK_SW_V FOREIGN KEY (CODV) REFERENCES VENDITORE (CODV),
CONSTRAINT FK_SW_TA FOREIGN KEY (PRODOTTOTA, VERSIONETA) REFERENCES TIPOARTICOLO (PRODOTTO, VERSIONE),
CONSTRAINT CK1_SW CHECK DATAACQUISIZIONE > DATAPRODUZIONE
);

```

```

CREATE TABLE TIPOARTICOLO (
PRODOTTO VARCHAR(300) NOT NULL,
VERSIONE INT NOT NULL,
DESCRIZIONE VARCHAR(1000) NOT NULL,
PRODOTTOTA VARCHAR(300) NULL,
VERSIONETA INT NULL,
CONSTRAINT PK1TA PRIMARY KEY (PRODOTTO, VERSIONE)
CONSTRAINT U1TA UNIQUE (PRODOTTO, VERSIONE)
CONSTRAINT FK_TA_TA FOREIGN KEY (PRODOTTOTA, VERSIONETA) REFERENCES TIPOARTICOLO (PRODOTTO, VERSIONE)
);

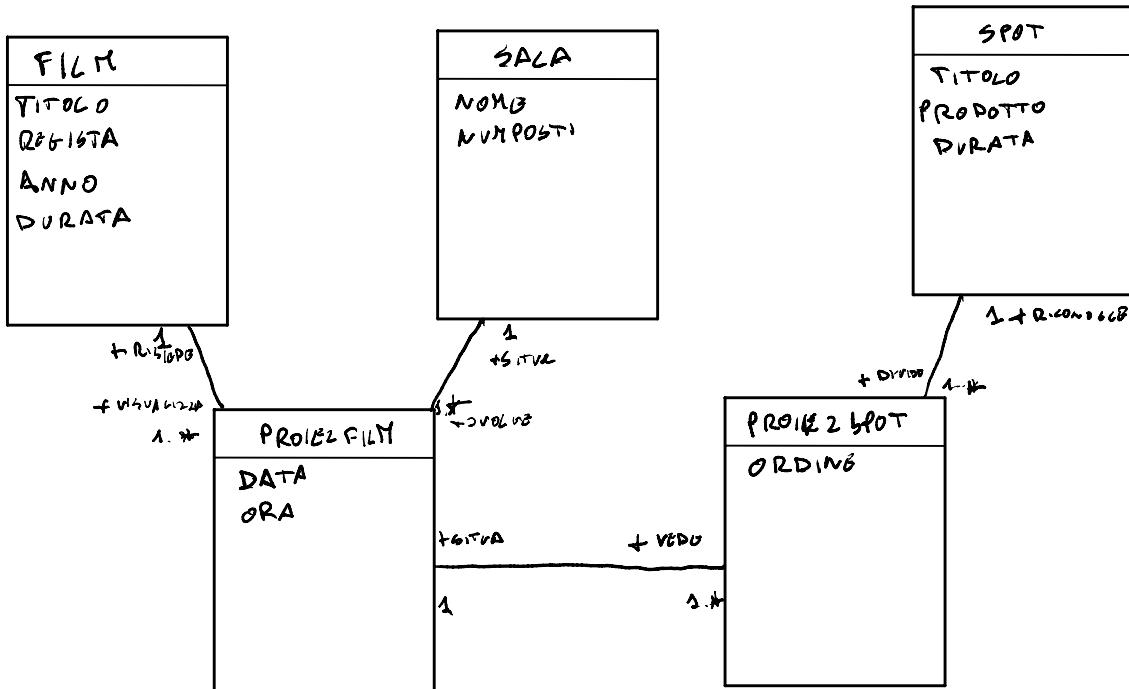
```

```

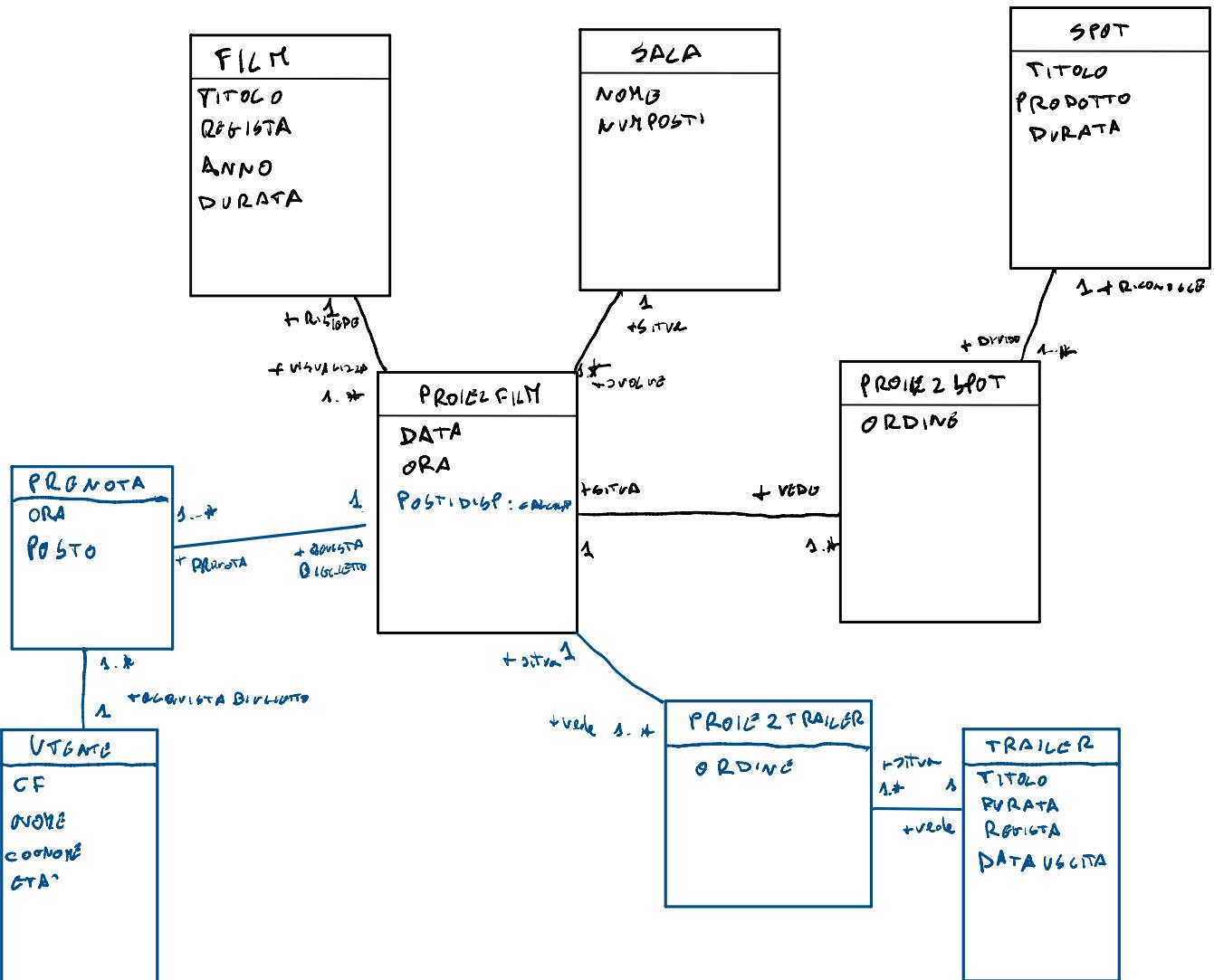
CREATE ASS_V_TA (
CODV INT NOT NULL,
PRODOTTOTA VARCHAR(300) NOT NULL,
VERSIONE INT NOT NULL,
CONSTRAINT FK_ASS_V_TA_TA FOREIGN KEY (PRODOTTOTA, VERSIONETA) REFERENCES TIPOARTICOLO (PRODOTTO, VERSIONE),
CONSTRAINT FK_ASS_V_TA_V FOREIGN KEY (CODV) REFERENCES VENDITORE (CODV)
);

```

*FILM(CodFilm, Titolo, Regista, Anno, Durata)*  
*SALA(CodSala, Nome, NumPosti)*  
*PROIEZFilm(CodProiez, CodSala, CodFilm, Data, Ora)*  
*SPOT(CodSpot, Titolo, Prodotto, Durata)*  
*PROIEZSpot(CodSpot, CodProiez, Ordine).*



2



3

### VINCOLO DI DOMINIO

```
ALTER TABLE FILM
CHECK ANNO >= YEAR(SYSDATE)
```

### VINCOLO INTRARELAZIONALE

```
ALTER TABLE FILM
CHECK TITOLO <> REGISTA
```

### VINCOLO INTERRELAZIONALE

```
ALTER TABLE PRENOTA
CHECK ORA < (SELECT P.ORA FROM PROIEZFILM P WHERE P.CODPROIEZ = CODPR)
```

MAGAZZINO(CodA, Descrizione, Collocazione, Quantita)  
 ORDINE(CodO, DataOrdine, CodFornitore, DataArrivo)  
 COMPORDINE(CodA, CodO, Quantita)  
 FORNITORE(PI, RagSociale, Via, Citta)  
 CATALOGO(CodA, PI, Prezzo)

**Esercizio 01** (Punti 8) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce il codice degli ordini e la ragione sociale dei fornitori per ordini non ancora arrivati ed urgenti (un ordine è urgente quando tutti gli articoli dell'ordine hanno scorta zero in magazzino).

$\overline{\Pi} \quad \overline{\Pi} \quad \overline{\Pi}$  6 ( ORDINE  $\bowtie$  COMPORDINE  $\bowtie$  MAGAZZINO )  
 ORDF 6 ( ORDINE  $\bowtie$  COMPORDINE  $\bowtie$  MAGAZZINO )  
 DATAARRIVO ISNULL  
 DATAARRIVO < DataOrdine  
 CODO, DATAORDINE, CODFORNITORE, DATAARRIVO  
 CODO, RAGSOCIALE  
 CODO, RAGSOCIALE

**Esercizio 02** (Punti 7) Si scriva una interrogazione in SQL che fornisca le PI dei fornitori che hanno nel catalogo tutti gli articoli che nel magazzino hanno scorta zero.

```

SELECT F.PI FROM FORNITORI
NATURAL JOIN CATALOGO C
NATURAL JOIN MAGAZZINO M
WHERE M.QUANTITA = 0;
  
```

**Esercizio 03** (Punti 8) Si esprima nel modo più adeguato il seguente insieme di vincoli:

- Un articolo non compare più di una volta nello stesso ordine;
- La data di arrivo di un ordine è successiva a quella di invio (DataOrdine);
- Un ordine può riferirsi solo ad articoli presenti nel magazzino
- Quando viene inserita la data di arrivo dell'ordine viene aggiornata automaticamente la quantità degli articoli in magazzino (alla quantità esistente si somma quella dell'ordine per ogni articolo);

1)

```

ALTER TABLE COMPORDINE
ADD CONSTRAINT U1 UNIQUE(CODA, CODO)
  
```

2)

```

ALTER TABLE ORDINE
ADD CONSTRAINT CK1 CHECK DATAARRIVO > DATAORDINE
  
```

3)

```

ALTER TABLE COMPORDINE
ADD CONSTRAINT CK1_CO CHECK EXIST (SELECT * FROM MAGAZZINO M WHERE M.CODA = CODA)
  
```

4)

```

CREATE TRIGGER T1
AFTER UPDATE OF DATAARRIVO ON ORDINE
  
```

CURSOR C1

```

IS SELECT M.CODA FROM ORDINE O
NATURAL JOIN COMPORDINE C
NATURAL JOIN MAGAZZINO M
WHERE O.CODO = OLD.CODO;

CA MAGAZZINO.CODA%TYPE;

BEGIN

OPEN C1
WHILE C1%FOUND
FETCH C1 INTO CA
LOOP

UPDATE MAGAZZINO
SET QUANTITA =
((SELECT M.QUANTITA FROM MAGAZZINO M
WHERE M.CODA = CA)
+
(SELECT C.QUANTITA
FROM COMPORDINE C
WHERE C.CODO = OLD.CODO
AND C.CODA = CA)

WHERE CODA = CA

END LOOP
CLOSE C1
END

```

**Esercizio 04 (Punti 9)** Si scriva una procedura che riceve in ingresso una stringa di codici di articolo separati dal simbolo separatore # e un intero k. La procedura controlla per ogni fornitore qual è il prezzo complessivo per un ordine di k esemplari di ogni articolo nella lista e sceglie il fornitore che ha il prezzo più vantaggioso. La procedura dunque inserisce nel database un ordine al fornitore scelto per tutti gli articoli. Supponendo che il codice dell'ordine sia un intero, il codice dell'ordine da inserire è l'intero successivo rispetto a quello già usato.

```

CREATE PROCEDURE P1
( CODICI_ARTICOLO VARCHAR(1000), INT K )

CODICI VARCHAR(1000) := CODICI_ARTICOLO;
PAROLA VARCHAR(200);
PREZZOCO CATALOGO.PREZZO%TYPE;

MAXPREZZO CATALOGO.PREZZO%TYPE := 0;
FORMAX FORNITORE.PI%TYPE;

CURSOR C1
IS
SELECT PI FROM FORNITORE;

FOR FORNITORE.PI%TYPE;

MAXORDINE.CODO%TYPE;

BEGIN

CREATE TABLE TEMP
( PAROLA MAGAZZINO.CODA%TYPE NOT NULL PRIMARY KEY);


```

```

WHILE CODICI <> ''
LOOP

PAROLA := SUBSTR(CODICI, 1, INSTR(CODICI, 1, '#'));
CODICI := SUBSTR(CODICI, INSTR(CODICI, 1, '#') + 2);

INSERT INTO TEMP VALUES(PAROLA);

END LOOP;

OPEN C1
WHILE C1%FOUND
FETCH C1 INTO FOR
LOOP

SELECT SUM(PREZZO) INTO PREZZOCO FROM FORNITORE F
JOIN ORDINE O ON O.CODFORNITORE = F.PI
JOIN COMPORDINE CO ON CO.CODO = O.CODO
JOIN CATALOGO CA ON CA.PI = F.PI AND CA.CODA = CO.CODA

WHERE CO.CODA IN (SELECT PAROLA FROM TEMP) AND F.PI = FOR
AND CO.QUANTITA = K

IF PREZZOCO > MAXPREZZO
THEN
MAXPREZZO := PREZZOCO;
FORMAX := FOR;
END IF

END LOOP
CLOSE C1

SELECT MAX(CODO) FROM ORDINE;

IF MAXPREZZO IS NOT NULL
THEN
INSERT INTO ORDINE VALUES (MAXORD + 1, SYSDATE(), FORMAX, NULL);
END IF

END

```

**Esercizio 05 (Punti 6)** Utilizzando SQL DINAMICO Si scriva una procedura che riceve in ingresso il nome di una tabella e una stringa di nomi di attributi separati dal carattere separatore #. La funzione ha l'effetto di aggiungere alla tabella passata per parametro un (unico) vincolo di unicità che coinvolge tutti gli attributi passati nella stringa.

```

CREATE PROCEDURE P2
( TABELLA UNSER_TABLE%TABLE_NAME, STRINGA VARCHAR(1000) )

COMANDO := 'ALTER TABLE ' || TABELLA || ' ADD CONSTRAINT U1 UNIQUE (' ;
S VARCHAR(1000) := STRINGA;
ATTR VARCHAR(200);

BEGIN

WHILE S <> ""
LOOP
ATTR = SUBSTR(S, 1, INSTR(S, 1, '#'));


```

```

S := SUBSTR(S, INSTR(S, '#') + 2);

COMANDO := COMANDO || ATTR || ';';

END LOOP

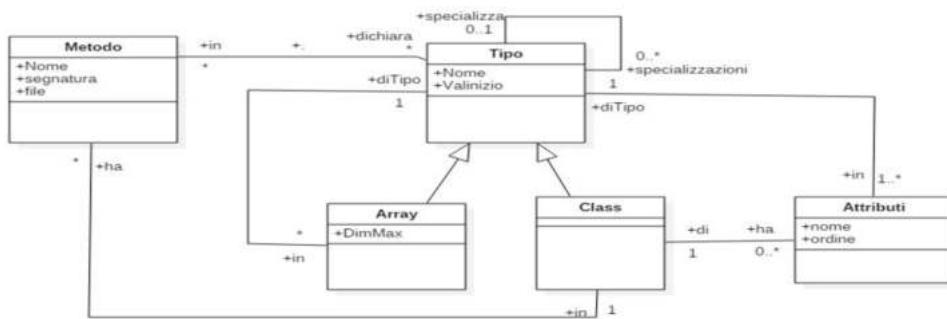
COMANDO := SUBSTR(COMANDO, 1, LENGTH(COMANDO) - 1);

COMANDO := COMANDO || ')';

EXECUTE IMMEDIATE COMANDO

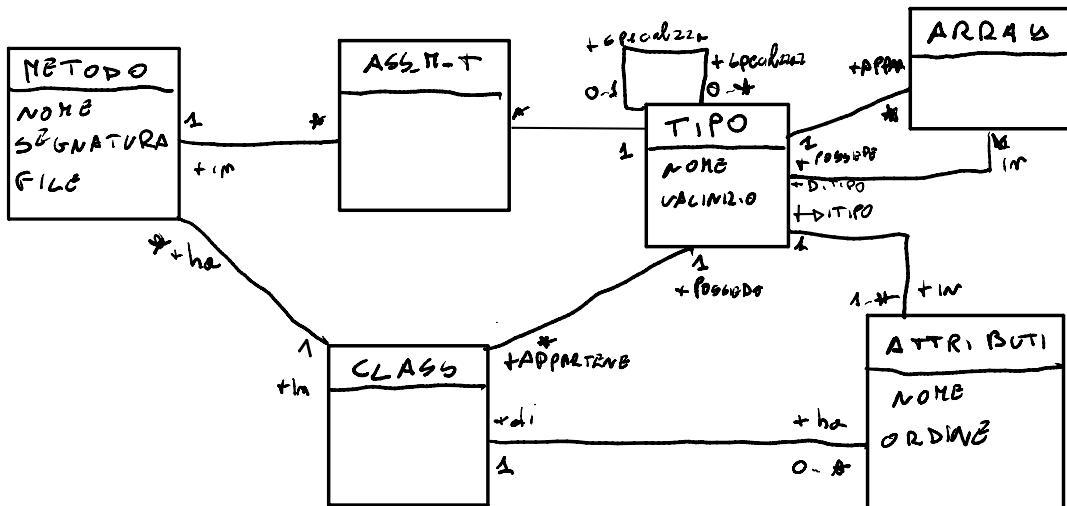
```

END



**Esercizio 11** Si consideri la bozza di Class Diagram in figura che descrive una base di dati per la memorizzazione di tipi di dato (oggetto). I tipi di dato sono strutturati tradizionalmente secondo lo schema. La specializzazione è di tipo parziale disgiunta.

1. Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (6 punti);
2. Si forniscano gli schemi relazionali per il class diagram ristrutturato (6 punti);
3. Si definisca usando SQL le tabelle (4 punti);
4. Nella definizione delle tabelle si scrivano i vincoli: (a) due attributi della stessa classe non possono avere lo stesso nome; (b) solo un tipo class può essere la specializzazione di un altro tipo nella relazione ricorsiva (4 punti)



METODO(CODM, NOME, SEGNATURA, FILE, CODC)

ASS\_M\_T(CODT, CODM)

```
TIPO(CODT, NOME, VALINIZIO, IDSPEC)
```

```
CLASS(CODC, CODT)
```

```
ARRAY(CODAR, CODT, IDT)
```

```
ATTRIBUTI(CODAT, NOME, ORDINE, CODC, CODT)
```

```
CREATE TABLE METODO (
    CODM INT NOT NULL,
    NOME VARCHAR(100) NOT NULL,
    SEGNATURA VARCHAR(200) NOT NULL,
    FILE VARCHAR(300) NOT NULL,
    CODC INT NOT NULL,
    CONSTRAINT PK_M PRIMARY KEY (CODM),
    CONSTRAINT UK_M UNIQUE (CODM),
    CONSTRAINT FK_M_C FOREIGN KEY (CODC) REFERENCES CLASS (CODC)
);
```

```
CREATE TABLE ASS_M_T (
    CODM INT NOT NULL,
    CODT INT NOT NULL,
    CONSTRAINT FK_ASS_M FOREIGN KEY (CODM) REFERENCES METODO (CODM),
    CONSTRAINT FK_ASS_T FOREIGN KEY (CODT) REFERENCES TIPO (CODT)
);
```

```
CREATE TABLE TIPO (
    CODT INT NOT NULL,
    NOME VARCHAR(100) NOT NULL,
    VALINIZIO INT NOT NULL,
    IDSPEC INT NULL,
    CONSTRAINT PK_T PRIMARY KEY (CODT),
    CONSTRAINT FK_T_T FOREIGN KEY (IDSPEC) REFERENCES CLASS (CODC)
);
```

```
CREATE TABLE CLASS (
    CODC INT NOT NULL,
    CODT INT NOT NULL,
    CONSTRAINT PK_C PRIMARY KEY (CODC),
    CONSTRAINT FK_C_T FOREIGN KEY (CODT) REFERENCES TIPO (CODT)
);
```

```
CREATE TABLE ARRAY (
    CODAR INT NOT NULL,
    CODT INT NOT NULL,
    IDT INT NOT NULL,
    CONSTRAINT PK_AR PRIMARY KEY (CODAR),
    CONSTRAINT FK_AR_T FOREIGN KEY (CODT) REFERENCES TIPO (CODT),
    CONSTRAINT FK_AR_T2 FOREIGN KEY (IDT) REFERENCES TIPO (CODT)
);
```

```
CREATE TABLE ATTRIBUTI (
    CODAT INT NOT NULL,
    NOME VARCHAR(200) NOT NULL,
    ORDINE VARCHAR(100) NOT NULL,
    CODC INT NOT NULL,
    CODT INT NOT NULL,
    CONSTRAINT PK_ATT PRIMARY KEY (CODAT),
    CONSTRAINT FK_ATT_C FOREIGN KEY (CODC) REFERENCES CLASS (CODC),
    CONSTRAINT FK_ATT_T FOREIGN KEY (CODT) REFERENCES TIPO (CODT),
    CONSTRAINT U_NAME_ATT UNIQUE (NOME)
);
```

*MAGAZZINO(CodA, Descrizione, Collocazione, Quantita)*  
*LISTINO(CodA, Quantita, Prezzo)*  
*ORDINE(CodO, DataOrdine, CodCliente, DataInvio, Completo)*  
*COMPORDINE(CodA, CodO, Quantita, Prezzo)*  
*CLIENTE(CF, Nome, Cognome, Via, Citta)*

**Esercizio 01** (Punti 8) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce il codice degli ordini non completi (*Completo = 'N'*) in cui TUTTI gli articoli compresi nell'ordine hanno scorta sufficiente in magazzino per soddisfare la richiesta.

*MAGAZZINO.QANTITA' ≥ COMPORDINE.QANTITA'*

$\exists \left( \begin{array}{l} \text{ORDINE } \bowtie \text{p(COMPORDINE)} \bowtie \text{MAGAZZINO} \\ \text{COMPOQUANTITA' = QUANTITA'} \\ \text{QUANTITA'} \geq \text{COMPQUANTITA'} \text{ AND } \text{Completo} = 'N' \end{array} \right)$

CODO

**Esercizio 02** (Punti 8) Si scriva una vista in SQL che per ogni cliente e per ogni anno a partire dall'anno 2000 fornisca la seguente informazione di riepilogo: il numero di ordini fatti in quell'anno, il numero di articoli distinti acquistati in quell'anno, la quantità complessiva di articoli acquistati in quell'anno, l'ammontare complessivo (prezzo) del materiale acquistato in quell'anno.

```
CREATE VIEW V1 (CLIENTE CLIENTE.CF%TYPE, ANNO INT)
AS
SELECT * FROM
(SELECT COUNT(*) AS NUMERO_ODINI_FATTI FROM ORDINE O
WHERE O.CODCLIENTE = CLIENTE AND YEAR(DATAORDINE) = ANNO)
LEFT JOIN
(SELECT COUNT(*) AS NUMERO_ARTICOLI_DISTINTI FROM ORDINE O
NATURAL JOIN COMPORDINE C
WHERE O.CODCLIENTE = CLIENTE AND YEAR(DATAORDINE) = ANNO)
LEFT JOIN
(SELECT SUM(QUANTITA) AS QUANTITA_ARTICOLI, SUM(PREZZO) AS PREZZO_TOTALE FROM ORDINE O
NATURAL JOIN COMPORDINE C
WHERE O.CODCLIENTE = CLIENTE AND YEAR(DATAORDINE) = ANNO)
```

```
CREATE VIEW V2 ()
AS
SELECT * FROM
(SELECT O.CODCLIENTE, COUNT(*) AS NUMERO_ODINI_FATTI FROM ORDINE O
WHERE YEAR(DATAORDINE) >= 2000
GROUP BY O.CODCLIENTE)
NATURAL JOIN
(SELECT O.CODCLIENTE, COUNT(*) AS NUMERO_ARTICOLI_DISTINTI FROM ORDINE O
NATURAL JOIN COMPORDINE C
WHERE YEAR(DATAORDINE) >= 2000
GROUP BY O.CODCLIENTE)
NATURAL JOIN
(SELECT O.CODCLIENTE, SUM(QUANTITA) AS QUANTITA_ARTICOLI, SUM(PREZZO) AS PREZZO_TOTALE FROM
ORDINE O
NATURAL JOIN COMPORDINE C
WHERE YEAR(DATAORDINE) >= 2000
GROUP BY O.CODCLIENTE)
```

**Esercizio 03** (Punti 8) Si scriva un trigger che viene azionato quando il valore dell'attributo

**Esercizio 03** (Punti 8) Si scriva un trigger che viene azionato quando il valore dell'attributo *Completo* viene aggiornato passando da valore '*N*' a valore '*S*'. L'azione del trigger è la seguente: prima si verifica che il magazzino disponga delle quantità richieste per ogni articolo presente nell'ordine. Se il vincolo non viene soddisfatto il valore dell'attributo *Completo* viene riaggiornato a '*N*'. Altrimenti si procede ad aggiornare le scorte di magazzino per ogni articolo presente nell'ordine sottraendo la quantità dell'articolo indicata nell'ordine.

```

CREATE TRIGGER T1
BEFORE UPDATE OF COMPLETO ON ORDINE
FOR EACH ROW

CURSOR C1
IS
(SELECT M.CODA FROM ORDINE O
NATURAL JOIN COMPORTINE C
NATURAL JOIN MAGAZZINO M
WHERE O.CODO = OLD.CODO)

CODICEM MAGAZZINO.CODA%TYPE;

BEGIN

IF OLD.COMPLETO = 'N' AND NEW.COMPLETO = 'S'
THEN

IF NOT EXIST (SELECT * FROM ORDINE O
NATURAL JOIN COMPORTINE C
NATURAL JOIN MAGAZZINO M
WHERE O.CODO = OLD.CODO AND
M.QUANTITA < C.QUANTITA)
THEN

OPEN C1
WHILE C1%FOUND
FETCH C1 INTO CODICEM
LOOP

UPDATE MAGAZZINO
SET QUANTITA =
(SELECT QUANTITA FROM MAGAZZINO WHERE CODA = CODICEM) -
(SELECT QUANTITA FROM COMPORTINE WHERE CODA = CODICEM AND CODO = OLD.CODO)
WHERE CODA = CODICEM;

END LOOP
CLOSE C1

ELSE
:NEW.COMPLETO = 'N';

--OPPURE SI FA UN UPDATE, OPPURE SI GENERA UNA ECCEZIONE
--RAISE EXCEPTION

END IF

END

```

**Esercizio 04** (Punti 7) Si esprima nel modo più adeguato il seguente insieme di vincoli:

- Un articolo non compare più di una volta nello stesso ordine;
- Il prezzo di listino di un articolo cala per quantitativi superiori (due voci di listino dello stesso articolo hanno prezzo inferiore per quantitativi superiori);
- Il prezzo di un articolo nell'ordine è inferiore o uguale a quello di listino;
- Un ordine può riferirsi solo ad articoli elencati nella tabella magazzino.

1)

```
ALTER TABLE COMPORDINE  
ADD CONSTRAINT UK1 UNIQUE (CODA, CODO)
```

2)

```
ALTER TABLE LISTINO  
ADD CONSTRAINT CK1  
CHECK NOT EXIST (SELECT FROM LISTINO L1 JOIN LISTINO L2 ON L1.CODA = L2.CODA WHERE  
(L1.QUANTITA < L2.QUANTITA AND (L1.PREZZO / L1.QUANTITA) <= (L2.PREZZO / L2.QUANTITA))  
OR (L1.QUANTITA > L2.QUANTITA AND (L1.PREZZO / L1.QUANTITA) >= (L2.PREZZO / L2.QUANTITA))  
)
```

3)

```
ALTER TABLE COMPORDINE  
ADD CONSTRAINT CK2  
CHECK PREZZO <= (SELECT L.PREZZO FROM LISTINO L WHERE L.CODA = CODA)
```

4)

```
ALTER TABLE COMPORDINE  
ADD CONSTRAINT CK3  
CHECK EXIST (SELECT * FROM MAGAZZINO M WHERE M.CODA = CODA)
```

**Esercizio 05** (Punti 8) Utilizzando SQL DINAMICO si scriva una funzione che riceve in ingresso una stringa di parole separate dal carattere separatore #. Ciascuna parola rappresenta una parola descrittiva di un articolo. La procedura ha l'effetto di recuperare i codici di articolo che hanno almeno una delle parole della lista contenute nell'attributo Descrizione. Tali codici vanno sequenzializzati separandoli con # e restituiti come valore.

```
CREATE FUNCTION F1  
( STRINGA VARCHAR(1000) )  
RETURN VARCHAR(1000)  
  
S VARCHAR(1000) := STRINGA;  
PAROLA VARCHAR(200);
```

```
RISULTATO VARCHAR(1000) := ";
```

```
SM MAGAZZINO.DESCRIZIONE%TYPE;  
SP VARCHAR(200);
```

```
ROW MAGAZZINO%ROWTYPE;
```

```
CURSOR C1  
AS  
SELECT *  
FROM MAGAZZINO;
```

```
BEGIN
```

```
WHILE S <> ''
LOOP

PAROLA := SUBSTR(S, 1, INSTR(S, 1, '#'));
S := SUBSTR(S, INSTR(S, 1, '#') + 2);
```

```
OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO ROW;
```

```
SM = ROW.DESCRIZIONE;
```

```
WHILE SM <> ''
LOOP

SP = SUBSTR(SM, 1, INSTR(SM, 1, ' '));
SM = SUBSTR(SM, INSTR(SM, 1, ' ') + 2);
```

```
IF SP LIKE PAROLA
THEN
SM = '';
```

```
RISULTATO := RISULTATO || ROW.CODA || '#';
```

```
END IF
```

```
END LOOP
```

```
END LOOP
CLOSE C1
```

```
END LOOP
```

```
RETURN RISULTATO;
END
```

*AUTORE(Orcid, Nome, Cognome, Affiliazione)**SCRIVE(Doi, Orcid, Ordine)**ARTICOLO(Doi, Titolo, Anno, pagI, pagF, NomeRivista, ISNN, Numero, Fascicolo)**BIBLIOGRAFIA(Doi, DoiCitato)**KEYWORDS(Doi, Parola)*

**Esercizio 01** (Punti 7) Si scriva una interrogazione in algebra relazionale che, se valutata, restituisca orcid, nome e cognome degli autori che non hanno MAI scritto articoli in collaborazione (sempre autori unici dei loro articoli).

*SCRIVE ↪ P(DOIS & DOI, ORCID & ORCID)*

1	1	0
1	2	1
2	1	0
2	3	0
3	0	0

$$\text{NO} \rightarrow \left( \bigcap_{\text{DOI} \neq \text{DOI}} (\text{SCRIVE} \bowtie \text{SCRIVE}) \bowtie \text{AUTORE} \right) \quad \text{ORCID} = \text{ORCID}$$

$$\overline{\bigcap}_{\text{ORCID}, \text{NAME}, \text{COGNOME}} (\text{AUTORE} \setminus \text{NO})$$

**Esercizio 02** (Punti 7) Si scriva in SQL una interrogazione che restituisca coppie di identificativi di documenti (doi1, doi2) che rispettano il seguente criterio: le parole chiave associate a doi2 sono TUTTE associate anche a doi1.

```
SELECT A1.DOI, A2.DOI
FROM ARTICOLO A1 JOIN KEYWORDS K1 ON
A1.DOI = K1.DOI
JOIN
ARTICOLO A2 JOIN KEYWORDS K2 ON
A2.DOI = K2.DOI
WHERE
(SELECT COUNT(*) FROM KEYWORDS WHERE DOI = A2.DOI) =
(SELECT COUNT(*) FROM KEYWORDS WHERE DOI = A2.DOI AND
PAROLA IN (SELECT PAROLA FROM KEYWORDS WHERE DOI =
A1.DOI))
```

**Esercizio 03** (Punti 7) Si implementino nel modo più opportuno i seguenti vincoli:

- Due autori di uno stesso articolo devono avere valori di ordine diverso;
- Con riferimento alla tabella BIBLIOGRAFIA, l'anno di pubblicazione del lavoro identificato da doi deve essere uguale o successivo all'anno di pubblicazione del lavoro identificato da DoiCitato;
- Un articolo non deve citare più di una volta un altro articolo. *non può citare lo stesso articolo due volte*

1

```
ALTER TABLE SCRIVE
ADD CONSTRAINT U1 UNIQUE (DOI, ORDINE)
```

3

```
ALTER TABLE BIBLIOGRAFIA
ADD CONSTRAINT U2 UNIQUE (DOI, DOICITATO)
--ADD CONSTRAINT U2 UNIQUE (DOI)
```

2

```
ALTER TABLE ARTICOLO
ADD CONSTRAINT CK1 CHECK NOT EXIST
```

```
(SELECT * FROM BIBLIOGRAFIA B JOIN ARTICOLO A ON A.DOI = B.DOI
WHERE A.ANNO < (SELECT ANNO FROM ARTICOLO WHERE DOI = B.DOICITATO))
```

**Esercizio 04 (Punti 7)** Si scriva una funzione PLSQL che riceve l'Orcid di un autore e che restituisca una lista ordinata di doi di lavori dell'autore organizzata come segue. La lista deve essere presentata in ordine decrescente per numero di citazioni dei lavori. Un lavoro va incluso nella lista se e solo se il suo numero di citazioni è maggiore al numero della sua posizione nella lista (es. un lavoro al quinto posto deve avere almeno 5 citazioni).

```
CREATE FUNCTION F1
( IDAUTORE IN AUTORE.ORCID%TYPE )
RETURN VARCHAR(1000)
```

```
CURSOR C1
IS
SELECT DISTINCT DOI FROM SCRIVE S
WHERE S.ORCID = IDAUTORE

LISTA VARCHAR2(1000);
INDICE INT;

DOITEMP ARTICOLO.DOI%TYPE;
NUMEROCITAZIONI INT;

BEGIN
INDICE := 0;
```

```
OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO DOITEMP
```

```
SELECT COUNT(*) INTO NUMEROCITAZIONI FROM BIBLIOGRAFIA B
WHERE B.DOI = DOITEMP;
```

```
IF NUMEROCITAZIONI >= INDICE
THEN
```

```
LISTA := LISTA || DOITEMP || ';';
INDICE := INDICE + 1;
```

```
END IF
```

```
END LOOP
```

```
LISTA := SUBSTR(LISTA, 1, LENGTH(LISTA) - 1);
```

**Esercizio 02 (8 punti)** Si scriva una interrogazione in SQL che fornisca coppie di codici di alberi tali che il secondo elemento della coppia è un sottoalbero del primo elemento della coppia (ogni nodo e ogni arco del sottoalbero deve essere nodo e arco, rispettivamente, del primo albero).

```
SELECT A1.CODA, A2.CODA
FROM ALBERI A1
JOIN ALBERI A2
ON A1.RADICE = A2.RADICE
WHERE
(SELECT COUNT(*) FROM COMPNODI CN WHERE CN.CODA = A2.CODA) =
(SELECT COUNT(*) FROM COMPNODI CN2 WHERE CN2.CODA = A2.CODA AND CN2.CODN IN
(SELECT CODN FROM COMPNODI WHERE CODA = A1.CODA))
AND
(SELECT COUNT(*) FROM COMPARCHI CA WHERE CA.CODA = A2.CODA) =
(SELECT COUNT(*) FROM COMPARCHI CA2 WHERE CA2.CODA = A2.CODA AND CA2.FIGLIO IN
(SELECT FIGLIO FROM COMPARCHI WHERE CODA = A1.CODA))
```

**Esercizio 05 (Punti 7)** Utilizzando SQL DINAMICO Si scriva una funzione che riceve in ingresso un orcid corrispondente a un autore ed una stringa di valori per Anno separati dal carattere separatore #. La funzione restituisce in una stringa la lista dei lavori dell'autore pubblicati negli anni indicati.

```
CREATE FUNCTION F1
( CODICEAUTORE AUTORE.ORCID%TYPE, STRINGA VARCHAR2(1000) )
```

```
COMANDO := 'SELECT S.DOI FROM SCRIVE S JOIN ARTICOLO A ON A.DOI = S.DOI WHERE A.DOI = || CODICEAUTORE ||
'AND A.ANNO IN (';
```

```
CURSOR C1
IS
```

EXECUTE IMMEDIATE COMANDO

AN ARTICOLO.ANNO%TYPE;  
S VARCHAR2(1000) := STRINGA;

COD ARTICOLO.DOI%TYPE;  
LISTA VARCHAR2(1000) := ":";

BEGIN

WHILE S <> ''  
LOOP

AN = SUBSTR(S, 1, INSTR(S, 1, '#'));  
S := SUBSTR(S, INSTR(S, 1, '#') + 2);

COMANDO := COMANDO || AN || ',';

END LOOP

COMANDO := SUBSTR(COMANDO, 1, LENGTH(COMANDO) - 1);  
COMANDO := COMANDO || ':';

OPEN C1

WHILE C1%FOUND

LOOP

FETCH C1 INTO COD

LISTA := LISTA || COD || ',';

END LOOP

LISTA = SUBSTR(LISTA, 1, LENGTH(LISTA) - 1);

RETUN LISTA;

END

---

$\text{LIBRO}(\underline{\text{ISBN}}, \text{Titolo}, \text{Editore}, \text{Anno})$   
 $\text{ESEMPLARE}(\underline{\text{ISBN}}, \underline{\text{CodiceBarre}}, \text{Collocazione}, \text{Prestito}, \text{Consultazione})$   
 $\text{UTENTE}(\underline{\text{CF}}, \text{CodProfilo}, \text{Nome}, \text{Cognome}, \text{DataN})$   
 $\text{PROFILO}(\text{CodProfilo}, \text{MaxDurata}, \text{MaxPrestito})$   
 $\text{PRESTITI}(\underline{\text{CodPrestito}}, \underline{\text{CodUtente}}, \text{Data}, \text{Scadenza}, \text{Restituzione}, \text{Sollecito})$   
 $\text{PRENOTAZIONE}(\underline{\text{CodPrenotazione}}, \underline{\text{ISBN}}, \text{Utente}, \text{Data})$

**Esercizio 01** (Punti 7) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce l'ISBN del libro che ha avuto nell'anno 2018 il maggior numero di prestiti (si intende per numero di prestiti di un libro il numero di prestiti di tutti i suoi esemplari).

•  $L \leftarrow \sigma_{\text{Anno} = 2018} \pi_{\text{ISBN}} (\text{LIBRO})$        $L \in \text{LIBRO}$   
 $\text{PRSTTO} = \text{CODPrestito}$   
 3)  $\overline{\prod}_{\text{ISBN}} \left( \text{ISBN} \sum \text{MAX}(\text{n\_PRSTTO}) \right) (R)$        $R \in \text{PRESTITI}$   
 $\text{ISBN}, \text{n\_PRSTTO}$   
 $\text{LIBRO}$

**Esercizio 02** (Punti 7) Si scriva una interrogazione in SQL che restituisca coppie di utenti che nel 2018 hanno preso in prestito lo stesso numero di libri.

```

SELECT U1.CF, U2.CF
FROM UTENTI U1
JOIN PRESTITI P1 ON U1.CF = P1.UTENTE
LEFT OUTER JOIN
UTENTI U2 JOIN PRESTITI P2 ON U2.CF = P2.UTENTE

WHERE U2.CF <> U1.CF AND YEAR(P1.DATA) = 2018 AND YEAR(P2.DATA) = 2018
AND
(SELECT COUNT(*) FROM PRESTITI WHERE UTENTE = U1.CF AND YEAR(DATA) = 2018) =
(SELECT COUNT(*) FROM PRESTITI WHERE UTENTE = U2.CF AND YEAR(DATA) = 2018)
    
```

**Esercizio 03** (Punti 8) Si implementino nel modo più adeguato i seguenti vincoli:

1. Un utente non può avere più prestiti in corso (esemplari non ancora restituiti) di quanto previsto dal suo profilo.
2. Se è presente un sollecito la restituzione è stata fatta dopo la data di scadenza.
3. Quando viene aggiornato un prestito indicando una data di sollecito, tutte le prenotazioni di quell'utente vengo automaticamente cancellate.

1

```

ALTER TABLE PRESTITI
ADD CONSTRAINT CK1 CHECK
( SELECT COUNT(*) FROM PRESTITI P WHERE P.UTENTE = UTENTE AND RESTITUZIONE IS NULL ) <=
( SELECT PROF.MAXPRESTITO FROM UTENTE U INNER JOIN PROFILO PROF ON PROF.CODPROFILO =
U.CODPROFILO WHERE U.CF = UTENTE )
    
```

2

```

ALTER TABLE PRESTITI
ADD CONSTRAINT CK2 CHECK NOT EXISTS
( SELECT * FROM PRESTITI P WHERE P.SOLLECITO IS NOT NULL AND P.RESTITUZIONE <= P.SCADENZA )
    
```

3

```

CREATE TRIGGER T1
AFTER UPDATE OF SOLLECITO ON PRESTITI
FOR EACH ROW
    
```

```

BEGIN
DELETE FROM PRENOTAZIONE
WHERE UTENTE = OLD.UTENTE;
    
```

END

**Esercizio 04** (Punti 8) Si scriva una funzione che, quando viene eseguita, controlla quali sono i prestiti che hanno raggiunto la scadenza alla data dell'esecuzione ed hanno correntemente il campo Sollecito a NULL. L'effetto della funzione di inserire nel campo Sollecito la data corrente. Inoltre, la funzione restituisce alla terminazione una stringa contenente, CF, Nome, Cognome, Titolo del libro per tutti i prestiti in ritardo (separati da ;) per cui stato indicato il sollecito.

```
CREATE FUNCTION F1 ()  
RETURN VARCHAR2(1000)  
  
CURSOR C1  
IS  
SELECT P.CODPRESTITO FROM PRESTITI P  
WHERE P.SOLLECITO IS NULL AND SCADENZA <= SYSDATE();  
  
CODP PRESTITI.CODPRESTITO%TYPE;  
STRINGA VARCHAR2(1000) := ";  
  
COF UTENTE.CF%TYPE;  
NO UTENTE.NOME%TYPE;  
CO UTENTE.COGNOME%TYPE;  
TI LIBRO.TITOLO%TYPE;  
  
BEGIN  
  
OPEN C1  
WHILE C1%FOUND  
LOOP  
FETCH C1 INTO CODP  
  
UPDATE PRESTITI  
SET SOLLECITO = SYSDATE()  
WHERE CODPRESTITO = CODP;  
  
SELECT P.UTENTE INTO COF FROM PRESTITI P WHERE P.CODPRESTITO = CODP;  
SELECT U.NOME INTO NO FROM UTENTE U WHERE U.CF = COF;  
SELECT U.CONOME INTO CO FROM UTENTE U WHERE U.CF = COF;  
SELECT L.TITOLO INTO TI FROM PRESTITI P INNER JOIN ESEMPLARE E ON E.CODICEBARRE = P.CODICEBARRE INNER JOIN  
LIBERO L ON E.ISBN = L.ISBN WHERE P.CODPRESTITO = CODP;  
  
STRINGA := STRINGA || COF || NO || CO || TI || ':';  
  
END LOOP  
  
STRINGA := SUBSTR(STRINGA, 1, LENGTH(STRINGA) - 1);  
  
RETURN STRINGA  
  
END
```

**Esercizio 05** (Punti 8) Si scriva una funzione che riceve in ingresso una stringa contenente delle parole separate tra loro dal simbolo -. Si scriva una interrogazione in SQL dinamico che recupera i libri in cui almeno una delle parole della stringa compare nel titolo. La funzione restituisce una stringa contenente i titoli dei libri recuperati separati dal simbolo -.

```
CREATE FUNCTION F2  
( STRINGA VARCHAR2(1000) )  
RETURN VARCHAR2(1000)  
  
COMANDO VARCHAR2(1000) := 'SELECT L.TITOLO FROM LIBRI L';  
S VARCHAR2(1000) := STRINGA;  
P VARCHAR2(100);
```

```

T LIBRI.TITOLO%TYPE;
PT VARCHAR2(100);
TT LIBRI.TITOLO%TYPE;

CURSOR C1
IS
EXECUTE IMMEDIATE COMANDO;

RISULTATO VARCHAR2(1000) := "";

BEGIN

WHILE S <> "
LOOP
P := SUBSTR(S, 1, INSTR(S, '-', 1));
S := SUBSTR(S, INSTR(S, '-', 1) + 2);

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO T

TT := T;

WHILE TT <> "
LOOP

PT := SUBSTR(TT, 1, INSTR(TT, ' ', 1));
TT := SUBSTR(TT, INSTR(TT, ' ', 1) + 2);

IF PT = P
THEN
TT := ""
RISULTATO := RISULTATO || T || '-';
END IF

END LOOP

END LOOP

END LOOP

RISULTATO := SUBSTR(RISULTATO, 1, LENGTH(RISULTATO) - 1);

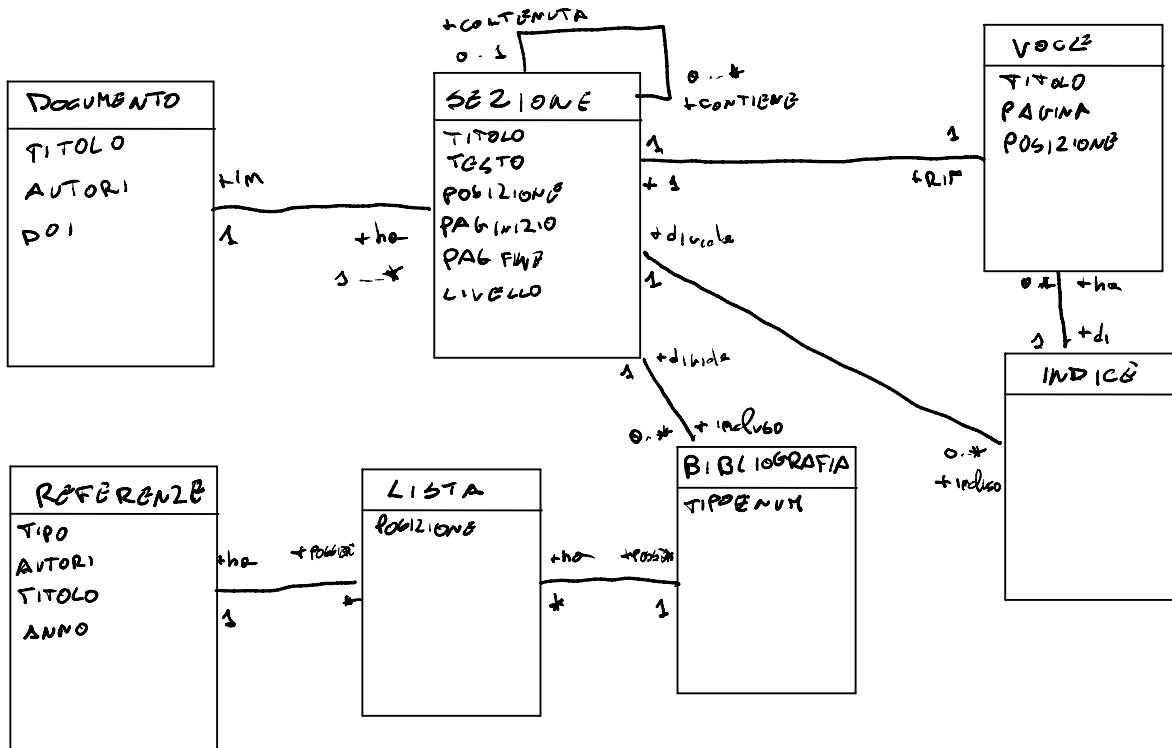
RETURN RISULTATO;
END

```

---



- Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (6 punti);
- Si forniscano gli schemi relazionali per il class diagram ristrutturato (6 punti);
- Si definisca usando SQL le tabelle (4 punti);
- Nella definizione delle tabelle si scrivano i vincoli: (a) una sezione è associata a un documento solo se ha livello 1; (b) due voci dello stesso indice hanno posizioni diverse; (4 punti)



DOCUMENTO(TITOLO, AUTORI, DOI (PK))

SEZIONE(CODS, TITOLO, TESTO, POSIZIONE, PAGINIZIO, PAGFINE, LIVELLO, IDSEZIONE, DOI)

VOCE(CODV (PK), TITOLO, PAGINA POSIZIONE, CODI (FK))

INDICE(CODI (PK), CODS (FK))

BIBLIOGRAFIA(CODB (PK), CODS (FK), TIPOENUM)

LISTA(CODR (FK), CODB (FK), POSIZIONE)

REFERENZE(CODR (PK), TIPO, AUTORI, TITOLO, ANNO)

Come vincoli, in BIBLIOGRAFIA e in INDICE il codice Sezione e' FACOLTATIVO

In Voce il codice Indice e' facoltativo

In Sezione l'IDSEZIONE e' facoltativo E IL DOI E' FACOLTATIVO

4)

ALTER TABLE SEZIONE

ADD CONSTRAINT CK1

CHECK NOT EXIST (SELECT \* FROM SEZIONE WHERE LIVELLO <> 1 AND DOI IS NOT NULL)

ALTER TABLE VOCE

ADD CONSTRAINT CK1

CHECK NOT EXIST (SELECT \* FROM VOCE V1 JOIN VOCE V2 ON V1.CODI = V2.CODI)

WHERE V1.CODV <> V2.CODV AND V1.POSIZIONE = V2.POSIZIONE)

//

TABELLE(CodTab, Nome, NumRighe)

ATTRIBUTI(CodTab, Nome, Tipo, NotNull)

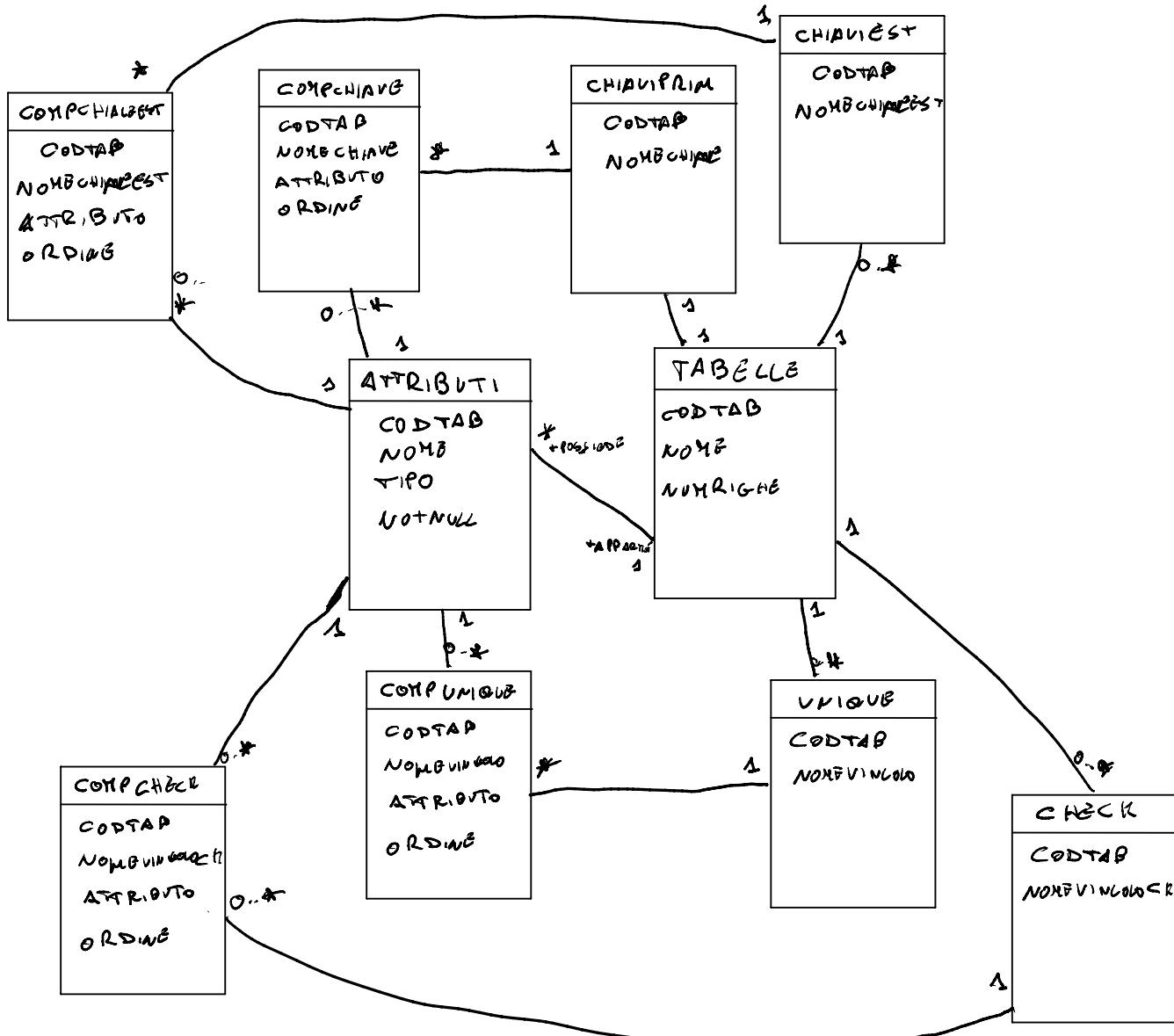
UNIQUE(CodTab, NomeVincolo)

CHIAVIPRIM(CodTab, NomeChiave)

COMPCHIAVE(CodTab, NomeChiave, Attributo, Ordine)

COMPUNIQUE(CodTab, NomeVincolo, Attributo, Ordine).

1. Si fornisca un Class Diagram di progettazione per lo schema relazionale (5 punti);
2. Si estenda il class diagram ottenuto per permettere la descrizione di vincoli di chiave esterna con tutti i loro dettagli, vincoli di check da associare agli attributi, e vincoli di check da associare alla tabella; (5 punti)
3. Per il class diagram fornito, si dia un esempio di vincolo di dominio, uno di ennupla, uno intrarelazionale ed uno interrelazionale. (3 punti).



#### VINCOLO DI DOMINIO

ALTER TABLE COMPCHECK

ADD CONSTRAINT CK1

CHECK ORDINE <> 0

#### VINCOLO DI ENNUPLA

ALTER TABLE COMPUNIQUE  
ADD CONSTRAINT CK2  
CHECK ORDINE = LENGTH(ATTRIBUTO)

**VINCOLO INTRARELAZIONALE**

ALTER TABLE TABELLE  
ADD CONSTRAINT PK1  
PRIMARY KEY (COTAB)

**VINCOLO INTERRELAZIONALE**

ALTER TABLE ATTRIBUTI  
ADD CONSTRAINT FK1  
FOREIGN KEY (COTAB) REFERENCES TABELLE (COTAB)

$\text{LOG}(\text{Cod}, \text{Operazione}, \text{CodRisorsa}, \text{ValorePrima}, \text{ValoreDopo}, \text{CodTransazione})$  $\text{RISORSA}(\text{CodRisorsa}, \text{Locazione}, \text{Valore}, \text{Stato})$  $\text{RICHIESTE}(\text{CodTransazione}, \text{Tempo}, \text{tipoAccesso}, \text{CodRisorsa})$  $\text{ASSEGNAZIONE}(\text{CodTransazione}, \text{Tempo}, \text{CodRidorsa}, \text{tipoAccesso})$ 

**Esercizio 01** (Punti 8, 20 minuti) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce le coppie di transazioni in stallo.

Una coppia di transazione ( $\text{codtransazione1}, \text{codtransazione2}$ ) è in stallo se la prima ha una richiesta per una risorsa che è correntemente assegnata alla seconda transazione e, viceversa, la seconda transazione ha una richiesta per una risorsa che è correntemente assegnata alla prima.

$$\overline{\Pi} \left( \left( \text{uno} \wedge \text{due} \right) \mid \begin{array}{l} \text{RT} \leftrightarrow \text{AT} \text{ AND } \text{RT} = \text{RT2} \text{ AND } \text{AT} = \text{AT2} \\ \text{RT} \text{ AT} \end{array} \right)$$

$$\text{uno} \leftarrow \rho \left( \text{RICHIESTE} \rightarrow \text{RISORSA} \right) \bowtie \rho \left( \text{ASSEGNAZIONE} \right) \quad \text{AT} \leftarrow \text{CODTRANSAZIONE}$$

$$\text{due} \leftarrow \rho \left( \text{RICHIESTE} \rightarrow \text{RISORSA} \right) \bowtie \rho \left( \text{ASSEGNAZIONE} \right) \quad \text{RT2} \leftarrow \text{CODTRANSAZIONE}$$

**Esercizio 02** (Punti 8, 20 minuti) Scrivere nel modo più opportuno i seguenti vincoli:

- Una risorsa assegnata in scrittura a una transazione non può essere assegnata a nessuna altra transazione.
- Una transazione può avere solo un tipo di richiesta alla volta per una risorsa (ad esempio se ha già un richiesta in lettura per una risorsa non ne può inoltrare un'altra in scrittura).
- Una transazione che ha registrato una operazione di COMMIT non può avere altre richieste registrate a tempi successivi.

1)

```
ALTER TABLE RISORSA
ADD CONSTRAINT CK1
CHECK
(SELECT COUNT(*) FROM RISORSA R NATURAL JOIN ASSEGNAZIONE A
WHERE R.STATO = 'W-LOCK') = 1
```

2)

```
ALTER TABLE RICHIESTE
ADD CONSTRAINT UK1
UNIQUE (TIPOACCESSO, CODRISORSA)
```

3)

```
ALTER TABLE LOG
ADD CONSTRAINT CK1 CHECK NOT EXIST
(SELECT * FROM LOG L NATURAL JOIN RICHIESTE R WHERE L.OPERAZIONE = 'COMMIT'
AND R.TEMPO < (SELECT TEMPO FROM RICHIESTE WHERE CODTRANSAZIONE >
R.CODTRANSAZIONE AND CODRISORSA = L.CODRISORSA))
```

**Esercizio 03** (Punti 8, 20 minuti) Si implementi il seguente trigger. Quando una transazione registra una operazione di COMMIT sul log, tutte le sue assegnazioni vengono cancellate. Le risorse a lei assegnate tornano libere. Si consulta la tabella delle richieste e se vi sono richieste per le risorse liberate si assegna le risorse alle richieste in base al tempo di regis-

**Esercizio 03** (Punti 8, 20 minuti) Si implementi il seguente trigger. Quando una transazione registra una operazione di COMMIT sul log, tutte le sue assegnazioni vengono cancellate. Le risorse a lei assegnate tornano libere. Si consulta la tabella delle richieste e se vi sono richieste per le risorse liberate si assegna le risorse alle richieste in base al tempo di registrazione (chi prima richiede prima viene servito) introducendo le assegnazioni nella tabella ASSEGNAZIONE e modificando lo stato delle risorse (se nuovamente assegnate hanno uno stato di R-LOCK o W-LOCK a seconda della richiesta e se non sono riassegnate hanno uno stato UNLOCK).

```
CREATE TRIGGER T1
AFTER INSERT ON LOG
FOR EACH ROW
```

```
CURSOR C1
IS
SELECT CODTRANSAZIONE FROM RICHIESTE
WHERE CODRISORSA = NEW.CODRISORSA
ORDER BY TEMPO ASC;

C RICHIESTE.CODTRANSAZIONE%TYPE;
T RICHIESTE.TEMPO%TYPE;
TA RICHIESTE.TIPOACCESSO%TYPE;

BEGIN

IF NEW.OPERAZIONE = 'COMMIT'
THEN

DELETE FROM ASSEGNAZIONE WHERE CODTRANSAZIONE =
NEW.CODTRANSAZIONE AND NEW.CODRISORSA = CODRISORSA;

UPDATE RISORSA
SET STATO = 'UNLOCK'
WHERE CODRISORSA = NEW.CODRISORSA;

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO C

INSERT INTO ASSEGNAZIONE VALUES(C, T, NEW.CODRISORSA, TA);

UPDATE RISORSA SET STATO = TA
WHERE CODRISORSA = NEW.CODRISORSA;

END LOOP

END
END
```

**Esercizio 04** (Punti 8) Si scriva una funzione con parametro intero Tout che per tutte le transazioni che sono in attesa per un tempo superiore a Tout (differenza tra il tempo di registrazione della richiesta e il tempo corrente) controlli se sono coinvolti in un deadlock (cioè se esiste un'altra transazione in deadlock nel senso indicato nel primo esercizio). La funzione

```
CREATE FUNCTION F1
(TOUT IN INT)

CURSOR C1
IS
SELECT CODTRANSAZIONE FROM RICHIESTE
WHERE (SYSTIME() - TEMPO) > TOUT;

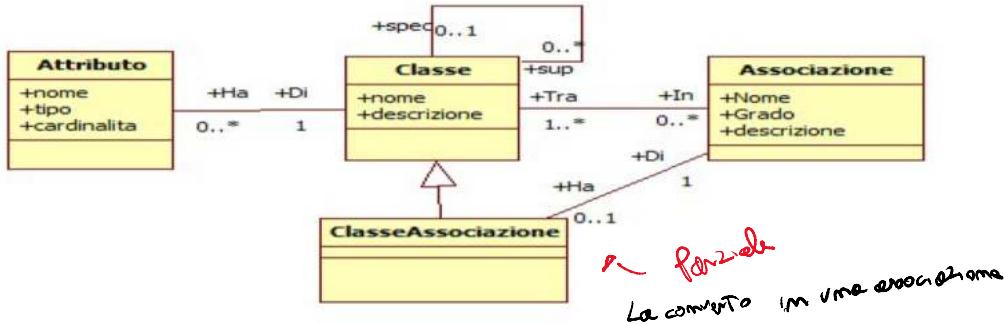
C RICHIESTE.CODTRANSAZIONE%TYPE;
```

Restituisce una stringa concatenata composta da tutti i codici di transazione in deadlock indicate.

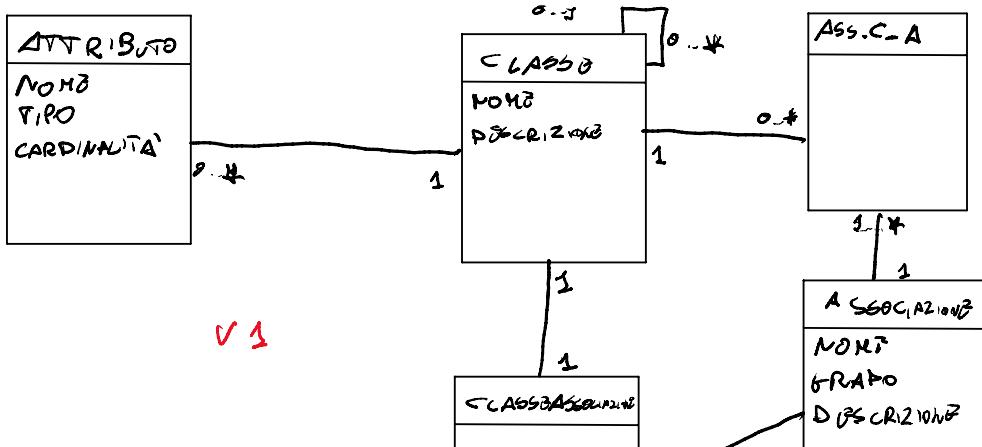
```

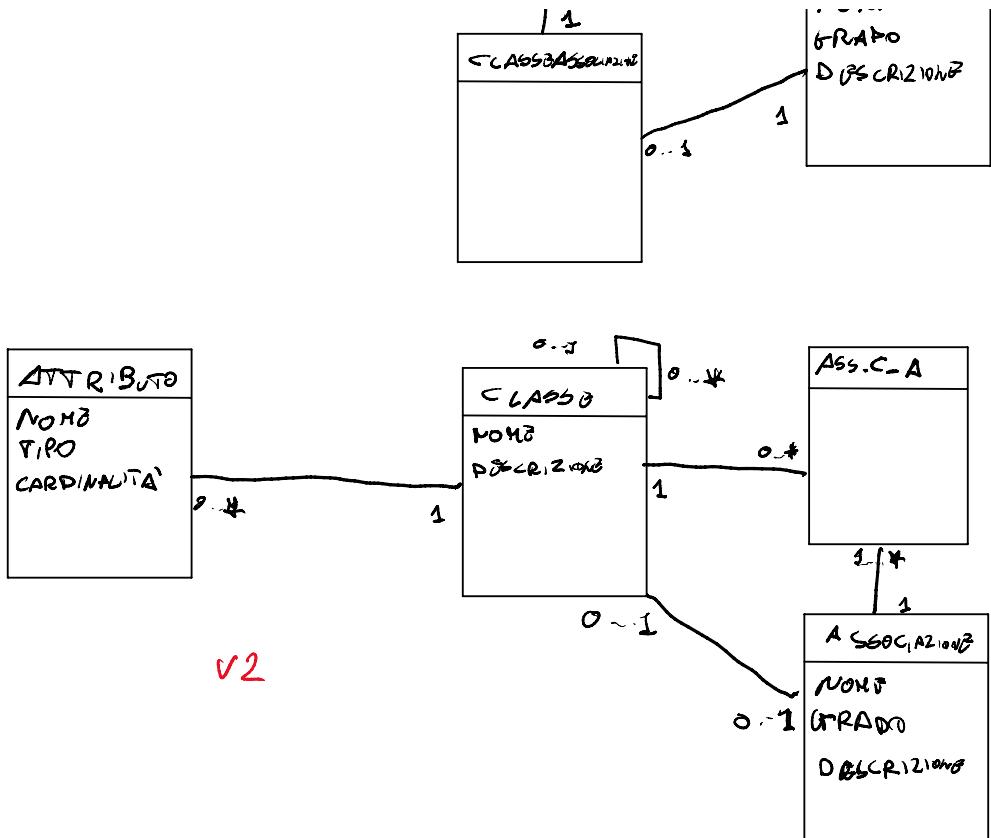
S VARCHAR2(1000) := '';
BEGIN
OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO C
IF EXIST
(SELECT R1.CODTRANSAZIONE AS RT, A1.CODTRANSAZIONE AS AT FROM RISORSA
RS1 NATURAL JOIN RICHIESTE R1
JOIN ASSEGNAZIONE A1 ON RS1.CODRISORSA = A1.CODRISORSA
WHERE AT <> RT)
INTERSECT
(SELECT R1.CODTRANSAZIONE AS AT1, A1.CODTRANSAZIONE AS RT1 FROM
RISORSA RS1 NATURAL JOIN RICHIESTE R1
JOIN ASSEGNAZIONE A1 ON RS1.CODRISORSA = A1.CODRISORSA
WHERE AT1 <> RT1)
WHERE RT = RT1 AND AT = AT1
AND ((RT = C) OR (AT = C))
THEN
STRINGA := STRINGA || C || ',';
END IF;
END LOOP
STRINGA := SUBSTR(STRINGA, 1, LENGTH(STRINGA) - 1);
RETURN STRINGA;
END

```



**Esercizio 05 (30 minuti)** Si consideri la bozza di Class Diagram in figura che descrive un frammento di un class diagram, con classi, specializzazioni di classi e associazioni tra classi. Si scriva prima lo schema logico per il class diagram e si definiscano poi le tabelle in SQL.





ATTRIBUTO (CODATT (PK), NOME, TIPO, CARDINALITA, CODC (FK))

CLASSE (CODC (PK), NOME, DESCRIZIONE, IDCLASSE (FK))

ASSOCIAZIONE (CODASS (PK), NOME, GRADO, DESCRIZIONE)

ASS\_C\_A(CODC (FK), CODASS (FK))

```

CREATE TABLE ATTRIBUTO (
    CODATT INT NOT NULL,
    NOME VARCHAR2(200) NOT NULL,
    TIPO VARCHAR2(500) NOT NULL,
    CARDINALITA VARCHAR2(100) NOT NULL,
    CODC INT NULL,
    CONSTRAINT PK1 PRIMARY KEY (CODATT),
    CONSTRAINT FK1 FOREIGN KEY (CODC) REFERENCES CLASSE (CODC );
)

```

```

CREATE TABLE CLASSE (
    CODC INT NOT NULL,
    NOME VARCHAR2(200) NOT NULL,
    DESCRIZIONE VARCHAR2(1000) NOT NULL,
    IDCLASSE INT NULL,
    CONSTRAINT PK1 PRIMARY KEY (CODC),
    CONSTRAINT FK1 FOREIGN KEY (IDCLASSE) REFERENCES CLASSE (CODC );
)

```

```

CREATE TABLE ASSOCIAZIONE (
    CODASS INT NOT NULL,
    NOME VARCHAR2(200) NOT NULL,
    GRADO INT NOT NULL,
    DESCRIZIONE VARCHAR2(1000) NOT NULL,
    CONSTRAINT PK1 PRIMARY KEY (CODASS) );

```

```

CREATE TABLE ASS_C_A (
    CODC INT NOT NULL,
    CODASS INT NOT NULL,
    CONSTRAINT FK1 FOREIGN KEY (CODC) REFERENCES CLASSE (CODC),
    CONSTRAINT FK2 FOREIGN KEY (CODASS) REFERENCES ASSOCIAZIONE (CODASS) );

```

1)  $\rho_1 = \rho(RICHIESTE)$   
 $CTS, T3, TAS, CR3$

$F = T3 = TEMPO$  AND  $CR3 = CODRISORDA$

$V3$   
 $\prod \left( G \left( R3 \bowtie ASSEGNAZIONE \right) \right)$   
 $CR3 \hookrightarrow CODTRANSAZIONE$   
 $CTS, CODTRANSAZIONE$

$CT3 = RICHIESTA$

$CODTRANSAZIONE = ASSEGNAZIONE$

$\prod \left( \begin{array}{c} \rho(V3) \\ \hline CTO, CTS \end{array} \bowtie \begin{array}{c} \rho(V3) \\ \hline CT2, CT3 \end{array} \right)$   
 $CTO = CT3$  AND  $CT3 = CT2$   
 $CTO, CT3$

2)  
CREATE ASSORTION 1  
CHECK NOTOKIST

SELECT \* FROM ASSIGNAZIONE Q1 JOIN ASSIGNAZIONE Q2  
ON Q1.CODRISORDA = Q2.CODRISORDA  
WHERE Q1.CODTRANSAZIONE < Q2.CODTRANSAZIONE AND  
Q1.TIPOACCESSO = 'W-CLOCK' AND Q2.TIPOACCESSO = 'W-CLOCK'

ALTER TABLE RICHIESTE  
ADD CONSTRAINT UNIQ\_V3 (CODTRANSAZIONE, TIPOACCESSO)

CREATE ASSORTION

CHECK NOTOKIST

SELECT CODTRANSAZIONE AS CT3, R.TEMPO AS T3

FROM LOG JOIN RICHIESTE R

WHERE L.CODTRANSAZIONE = 'COMMI'

NAME D01N

SELECT L.CODTRANSAZIONE AS CT3, R.TEMPO AS T2 FROM LOG NAT JOIN RICHIESTE R

WHERE T2 > T3

3)

*LOG(Cod, Operazione, CodRisorsa, ValorePrima, ValoreDopo, CodTransazione)  
 RISORSA(CodRisorsa, Locazione, Valore, Stato)  
 RICHIESTE(CodTransazione, Tempo, tipoAccesso, CodRisorsa)  
 ASSEGNAZIONE(CodTransazione, Tempo, CodRidorsa, tipoAccesso)*

**Esercizio 01 (Punti 8)** Si scriva una interrogazione in algebra relazionale che se valutata fornisce le transazioni che non hanno registrato la operazione di COMMIT sul log (sono ancora attive) e non hanno risorse assegnate.

$$\text{TOTB} \leftarrow \sum_{\text{COTRANSAZIONI}} \left( \begin{array}{l} \text{G(LOG)} \\ \text{CODRISORSA = NULL} \end{array} \right) \setminus \sum_{\text{COTRANSAZIONI}} \left( \begin{array}{l} \text{G(LOG)} \\ \text{OPERAZIONE = 'COMMIT'} \end{array} \right)$$

$$(\text{TOTG}) \setminus (\text{TOTB})$$

**Esercizio 02 (Punti 8)** Scrivere una interrogazione (vista) che considera le operazioni successive all'ultimo CHECK (solo operazioni successive per tempo al tempo dell'operazione di CHECK più recente nel LOG) e per quelle operazioni produce il seguente conteggio (tempo, N\_transazioni, N\_Commit, N\_Abort, N\_risorse\_riscritte) dove tempo è il valore dell'ultimo CHECK, N\_transazioni il numero di transazioni distinte dopo l'ultimo CHECK, N\_Commit e N\_Abort il numero di operazioni di COMMIT e ABORT, N\_risorse\_riscritte il numero di risorse distinte riscritte.

```
CREATE VIEW V1 ()  

AS  

(SELECT MAX(TEMPO) AS TEMPO FROM LOG NATURAL JOIN RICHIESTE WHERE OPERAZIONE = 'CHECK')  

LEFT OUTER JOIN  

()  

SELECT COUNT(L.CODTRANSAZIONE) AS N_TRANSAZIONI,  

FROM LOG L  

NATURAL JOIN RICHIESTE R  

WHERE R.TEMPO > (SELECT MAX(TEMPO) FROM LOG NATURAL JOIN RICHIESTE WHERE OPERAZIONE =  

'CHECK')  

)  

LEFT OUTER JOIN  

()  

SELECT COUNT(L.OPERAZIONI) AS N_COMMI,  

FROM LOG L  

NATURAL JOIN RICHIESTE R  

WHERE R.TEMPO > (SELECT MAX(TEMPO) FROM LOG NATURAL JOIN RICHIESTE WHERE OPERAZIONE =  

'CHECK') AND L.OPERAZIONE = 'COMMIT'  

)  

LEFT OUTER JOIN  

()  

SELECT COUNT(L.OPERAZIONI) AS N_ABORT,  

FROM LOG L  

NATURAL JOIN RICHIESTE R  

WHERE R.TEMPO > (SELECT MAX(TEMPO) FROM LOG NATURAL JOIN RICHIESTE WHERE OPERAZIONE =  

'CHECK') AND L.OPERAZIONE = 'ABORT'  

)  

LEFT OUTER JOIN  

()  

SELECT DISTINCT COUNT(L.CODRISORSA) AS N_RISORSE_RISCritte,  

FROM LOG L  

NATURAL JOIN RICHIESTE R  

WHERE R.TEMPO > (SELECT MAX(TEMPO) FROM LOG NATURAL JOIN RICHIESTE WHERE OPERAZIONE =  

'CHECK')
```

**Esercizio 03 (Punti 8)** Si implementi il seguente trigger. Quando una transazione registra una operazione di ABORT sul log, tutte le scritture fatte dalla transazione e riportate sul LOG devono essere annullate in ordine inverso a quelle in cui sono state fatte. Per annullare le scritture si deve consultare il log e si deve assegnare ad ogni risorsa scritta dalla transazione il valore ValorePrima riportato nel LOG. Inoltre, le risorse assegnate alla transazione

**Esercizio 03** (Punti 8) Si implementi il seguente trigger. Quando una transazione registra una operazione di ABORT sul log, tutte le scritture fatte dalla transazione e riportate sul LOG devono essere annullate in ordine inverso a quelle in cui sono state fatte. Per annullare le scritture si deve consultare il log e si deve assegnare ad ogni risorsa scritta dalla transazione il valore ValorePrima riportato nel LOG. Inoltre, le risorse assegnate alla transazione devono tornare libere: si rimuovono le assegnazioni alla transazione e lo stato della risorsa assume valore UNLOCK.

```
CREATE TRIGGER T1
AFTER INSERT ON LOG
FOR EACH ROW

CURSOR C1
IS
SELECT L.CODRISORSA
FROM LOG L
JOIN RICHIESTE R ON R.CODTRANSAZIONE = L.CODTRANSAZIONE
WHERE L.CODTRANSAZIONE = NEW.CODTRANSAZIONE
AND L.OPERAZIONE = 'ABORT'
ORDER BY (R.TEMPO) ASC;

CURSOR C2
IS
SELECT A.CODRISORSA
FROM ASSEGNAZIONE A WHERE
A.CODTRANSAZIONE = NEW.CODTRANSAZIONE;

CR RISORSA.CODRISORSA%TYPE;

BEGIN

IF NEW.OPERAZIONE = 'ABORT'
THEN

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO CR

UPDATE RISORSA
SET VALORE = NEW.VALOREPRIMA
WHERE CODICERISORSA = CR;

END LOOP

OPEN C2
WHILE C2%FOUND
LOOP
FETCH C2 INTO CR

UPDATE RISORSA
SET STATO = 'UNLOCK'
WHERE CODRISORSA = CR;

END LOOP

DELETE FROM ASSEGNAZIONE
WHERE CODTRANSAZIONE = NEW.CODTRANSAZIONE;

END IF

END
```

**Esercizio 04 (Punti 8)** Si scriva una funzione con parametro intero Tout che per tutte le transazioni T1 che sono in attesa per una risorsa per un tempo superiore a Tout (differenza fra il tempo di registrazione della richiesta e il tempo corrente) controlli se ci sia un deadlock (cioè se esiste un'altra transazione T2 che occupa la risorsa richiesta e la transazione T2 richiede una risorsa assegnata alla transazione T1). La funzione restituisce una stringa coi codici delle transazioni T1 in deadlock così trovate.

```

CREATE FUNCTION F1
( TOUT IN INT )

CURSOR C1
IS
SELECT L.CODTRANSAZIONE
FROM LOG L JOIN RICHIESTE R ON R.CODTRANSAZIONE = L.CODTRANSAZIONE
WHERE (SYSTIME() - R.TEMPO) > TOUT;

STRINGA VARCHAR2(1000) := "";
CT RICHIESTE.CODTRANSAZIONE%TYPE;

BEGIN

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO CT

IF EXIST
((SELECT L1.CODTRANSAZIONE AS C1, L2.CODTRANSAZIONE AS C2
FROM LOG L1 NATURAL JOIN RICHIESTE R1 NATURAL JOIN RISORSA RS1
JOIN LOG L2 ON L2.CODRISORSA = RS1.CODRISORSA JOIN RICHIESTE R2 ON
L2.CODTRANSAZIONE = R2.CODTRANSAZIONE
WHERE L1.CODTRANSAZIONE <> L2.CODTRANSAZIONE
)
INTERSECT
(
SELECT L1.CODTRANSAZIONE AS C2, L2.CODTRANSAZIONE AS C1
FROM LOG L1 NATURAL JOIN RICHIESTE R1 NATURAL JOIN RISORSA RS1
JOIN LOG L2 ON L2.CODRISORSA = RS1.CODRISORSA JOIN ASSEGNAZIONE R2 ON
L2.CODTRANSAZIONE = R2.CODTRANSAZIONE
WHERE L1.CODTRANSAZIONE <> L2.CODTRANSAZIONE
)
WHERE CT = C1 OR CT = C2;)

THEN

STRINGA := STRINGA || CT || ',';

END IF
END LOOP

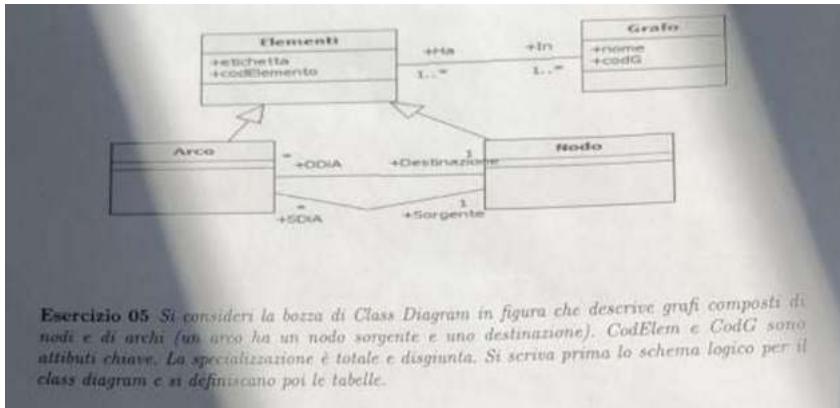
STRINGA := SUBSTR(STRINGA, 1, LENGTH(STRINGA) - 1);

RETURN STRINGA;

END

```

---



GRAFO  
 ASS\_ARCO\_GRAFO      ASS\_NODO\_GRAFO  
ARCO — NODO

**Esercizio 05** Si consideri la bozza di Class Diagram in figura che descrive grafi composti di nodi e di archi (un arco ha un nodo sorgente e uno destinazione). CodElem e CodG sono attributi chiave. La specializzazione è totale e disgiunta. Si scriva prima lo schema logico per il class diagram e si definiscano poi le tabelle.

ARCO(CODA (PK), ETICHETTA, DDIA (FK), SDIA (FK))

NODO(CODN (PK), ETICHETTA)

ASS\_ARCO\_GRAFO (CODA (FK), CODG (FK))

ASS\_NODO\_GRAFO (CODN (FK), CODG (FK))

GRAFO (CODG (PK), NOME)

```
CREATE TABLE GRAFO (
  CODG INT NOT NULL,
  NOME VARCHAR2(200) NOT NULL,
  CONSTRAINT PK1 PRIMARY KEY (CODG);
```

```
CREATE TABLE ARCO (
  CODA INT NOT NULL,
  ETICHETTA VARCHAR2(300) NOT NULL,
  DDIA INT NOT NULL,
  SDIA INT NOT NULL,
  CONSTRAINT PK1 PRIMARY KEY (CODA),
  CONSTRAINT FK1 FOREIGN KEY (DDIA) REFERENCES NODO (CODN),
  CONSTRAINT FK2 FOREIGN KEY (SDIA) REFERENCES NODO (CODN));
```

```
CREATE TABLE NODO (
  CODN INT NOT NULL,
  ETICHETTA VARCHAR2(300) NOT NULL,
  CONSTRAINT PK1 PRIMARY KEY (CODN));
```

```
CREATE TABLE ASS_ARCO_GRAFO (
  CODA INT NOT NULL,
  CODG INT NOT NULL,
  CONSTRAINT FK1 FOREIGN KEY (CODA) REFERENCES ARCO (CODA),
  CONSTRAINT FK2 FOREIGN KEY (CODG) REFERENCES GRAFO (CODG));
```

```
CREATE TABLE ASS_NODO_GRAFO (
  CODN INT NOT NULL,
  CODG INT NOT NULL,
  CONSTRAINT FK1 FOREIGN KEY (CODN) REFERENCES NODO (CODN),
  CONSTRAINT FK2 FOREIGN KEY (CODG) REFERENCES GRAFO (CODG));
```

*PERSONA(CF, Nome, Cognome, DataN, DataM, CittaN)  
RESIDENZA(CF, DataI, Via, Num, CittA, DataF)  
GENITORI(CF, CFMadre, CFPadre)*

**Esercizio 01 (8 punti, 20 minuti)** Si scriva una interrogazione in algebra relazionale che fornisca se valutata nomi e cognomi delle persone in vita che sono state residenti per tutta la vita nella città di nascita.

PVITA  $\rightsquigarrow \sigma$  (PERSONA)

R  
 $\frac{1}{1}$   
DATAI IS NULL

PERSONA  $\bowtie$   $\left( \text{cf} \sum_{\text{count(cf)}} \left( \text{PVITA} \bowtie \text{RESIDENZA} \right) \right)$

$\rho^S$   
 $\frac{1}{1} \rightsquigarrow \rho(R)$   
 $\text{m\_R} = \text{count(cf)}$

$\prod \left( \sigma(R^S) \middle| \begin{array}{l} \text{m\_R} = 1 \text{ AND } \text{CITTAN} = \text{CITTIA} \\ \text{Nome, cognome} \end{array} \right)$

V. d. - = RESIDENZA.C-

**Esercizio 02 (8 punti, 20 minuti)** Si esprima nel modo più adeguato il seguente insieme di vincoli:

- Ogni persona ha al più un padre ed una madre;
- Per ogni persona ci deve essere al più una residenza correntemente attiva;
- Quando viene fissata la data di morte viene anche chiusa in quella data la residenza.

1)

ALTER TABLE GENITORI

ADD CONSTRAINT U1 UNIQUE (CF, CFMADRE),  
ADD CONSTRAINT U2 UNIQUE (CF, CFPADRE);

2)

ALTER TABLE RESIDENZA

ADD CONSTRAINT CK1 CHECK NOT EXIST

(SELECT \* FROM

((SELECT R.CF, COUNT(\*) AS N\_RESIDENZE FROM RESIDENZA R

GROUP BY R.CF)

NATURAL JOIN

((SELECT R.CF, COUNT(\*) AS RES\_NONATTIVE FROM RESIDENZA R

WHERE R.DATAF IS NOT NULL

GROUP BY R.CF))

WHERE N\_RESIDENZE - RES\_NONATTIVE > 1)

3) -- CHECK

ALTER TABLE PERSONA

```

ADD CONSTRAINT CK2 CHECK NOT EXIST
(SELECT * FROM
(SELECT P.CF AS CF, P.DATAM AS DATAMORTE FROM PERSONA P WHERE P.DATAM IS NOT NULL)
NATURAL JOIN
(SELECT R.CF AS CF, MAX(DATAF) AS DATAFINERESIDENZA FROM RESIDENZA R GROUP BY R.CF)
WHERE DATAMORTE <> DATAFINERESIDENZA
)

```

3) -- TRIGGER

```

CREATE TRIGGER T1
AFTER UPDATE OF DATAM ON PERSONA
WHEN (OLD.DATAM IS NULL AND NEW.DATAM IS NOT NULL)
FOR EACH ROW
BEGIN

UPDATE RESIDENZA
SET DATAF = NEW.DATAM
WHERE CF = OLD.CF AND DATAI = (SELECT R.DATAI FROM RESIDENZA WHERE R.CF = OLD.CF AND DATA F IS
NULL);

END

```

**Esercizio 03 (8 punti, 20 minuti)** Si scriva una vista che per ogni città e ogni anno restituiscia il numero dei nati in città in quell'anno, il numero dei morti residenti in città in quell'anno, il numero persone che hanno preso la residenza nella città quell'anno e non erano residenti prima (Ingresso) il numero di persone che hanno lasciato la città (Uscita) avendo cambiato nell'anno città di residenza. (Schema ANAGRAFE(ANNO,CITTA,NATI,MORTI,INGRESSO,USCITA)).

```

CREATE VIEW ANAGRAFE (CITTA, ANNO)
SELECT * FROM VNATIANNO NATURAL JOIN
VNMORTIRC NATURAL JOIN VINGRESSO NATURAL JOIN VUSCITA;

```

```

CREATE VIEW VNATIANNO(CITTA, ANNO, NATI)
SELECT P.CITTAN, YEAR(P.DATAN), COUNT(P.CF) AS NATI FROM PERSONA P
GROUP BY P.CITTAN, YEAR(P.DATAN);

```

```

CREATE VIEW VNMORTIRC(CITTA, ANNO, MORTI)
SELECT R.CITTA, YEAR(P.DATAM), COUNT(P.CF) AS MORTI FROM PERSONA P NATURAL JOIN RESIDENZA R
WHERE P.DATAM IS NOT NULL
GROUP BY R.CITTA, YEAR(P.DATAM);

```

```

CREATE VIEW VINGRESSO(CITTA, ANNO, INGRESSO)
SELECT R.CITTA, YEAR(R.DATAI), COUNT(P.CF) AS INGRESSO FROM PERSONA P NATURAL JOIN RESIDENZA R
WHERE (SELECT COUNT(*) FROM RESIDENZA WHERE CF = P.CF) = 1
GROUP BY R.CITTA, YEAR(R.DATAI);

```

```

CREATE VIEW VUSCITA(CITTA, ANNO, USCITA)
SELECT E.CITTA, YEAR(E.DATAF), COUNT(E.CF) AS USCITA FROM
(SELECT CF AS FF, MAX(DATAF) AS DF FROM RESIDENZA R
GROUP BY CF)
JOIN RESIDENZA E ON E.CF = FF AND E.DATAF = DF
GROUP BY E.CITTA, YEAR(E.DATAF);

```

```
GROUP BY E.CITTA, YEAR(E.DATAF);
```

**Esercizio 04 (8 punti, 20 minuti)** Si scriva una procedura PLSQL che riceve in ingresso il CF di una persona e restituisce una stringa di caratteri contenente il nome e cognome di tutti gli antenati in linea femminile (natre, nonna materna, madre della nonna materna etc.).

```
CREATE PROCEDURE P1 (F IN PERSONA.CF%TYPE, S OUT VARCHAR2(1000))
```

```
CURSOR C1
```

```
IS
```

```
SELECT CFMADRE FROM GENITORI
```

```
WHERE CF = CODF;
```

```
CODF PERSONA.CF%TYPE := F;
```

```
N PERSONA.NOME%TYPE;
```

```
C PERSONA.COGNOME%TYPE;
```

```
BEGIN
```

```
--CODF := (SELECT CFMADRE FROM PERSONA WHERE CF = F);
```

```
OPEN C1
```

```
WHILE C1%FOUND
```

```
LOOP
```

```
FETCH C1 INTO CODF
```

```
SELECT NOME INTO N FROM PERSONA WHERE CF = CODF;
```

```
SELECT COGNOME INTO C FROM PERSONA WHERE CF = CODF;
```

```
S := S || N || ' ' || C || ';' ;
```

```
END LOOP
```

```
S := SUBSTR(S, 1, LENGTH(S) - 1);
```

```
END
```

$\text{FILE}(\text{File\_ID}, \text{Path}, \text{Dimensione}, \text{Nome})$   
 $\text{BLOCCHI}(\text{Cod\_B}, \text{File\_ID}, \text{Ordine}, \text{Dimensione})$   
 $\text{UTENTE}(\text{Usr\_ID}, \text{email}, \text{NickName}, \text{IP})$   
 $\text{POSSIEDE\_BLOCCHI}(\text{Usr\_ID}, \text{Cod\_B}, \text{File\_ID}, \text{data})$   
 $\text{POSSIEDE\_FILE}(\text{Usr\_ID}, \text{File\_ID}, \text{data})$   
 $\text{SCAMBIO}(\text{User\_UP}, \text{User\_Down}, \text{File\_ID}, \text{Cod\_B}, \text{Data}, \text{Ora}, \text{Completo})$

**Esercizio 01** (Punti 8, 20 minuti) Si scriva una interrogazione in algebra relazionale che per ogni file e per ogni anno fornisca il numero di utenti distinti che hanno fatto operazioni di (DOWNLOAD) sui blocchi del file e il numero di utenti distinti che hanno fatto operazioni di (UPLOAD) in quell'anno sui blocchi del file (schema richiesto SUMMARY(File\_ID, Anno, Num\_UP, Num\_DOWN)).

$$\begin{array}{l}
 F1 \quad \Delta_{\text{Down}} \leftarrow \rho \left( \text{FILE\_ID}, \text{YEAR}(\text{DATA}) \geq \text{COUNT}(\text{USR\_UP}) \mid (\text{F1}) \right) \\
 \qquad \qquad \qquad \text{FILE\_ID, ANNO, NUM\_UP} = \text{COUNT}(\text{USR\_UP}) \\
 \overline{\Pi} \left( \rho \left( \text{SCAMBIO} \right) \right) \\
 \qquad \qquad \qquad \text{COMPLETO} = \text{TRUE} \\
 \qquad \qquad \qquad \text{DATA, USR\_UP, FILE\_ID} \\
 \qquad \qquad \qquad \text{SUMMARY} \leftarrow (\text{Down} \bowtie \text{Up})
 \end{array}$$
  

$$\begin{array}{l}
 F2 \quad \Delta_{\text{Up}} \leftarrow \rho \left( \text{FILE\_ID}, \text{YEAR}(\text{DATA}) \geq \text{COUNT}(\text{USR\_DOWN}) \mid (\text{F2}) \right) \\
 \qquad \qquad \qquad \text{FILE\_ID, ANNO, NUM\_DOWN} = \text{COUNT}(\text{USR\_DOWN}) \\
 \overline{\Pi} \left( \rho \left( \text{SCAMBIO} \right) \right) \\
 \qquad \qquad \qquad \text{COMPLETO} = \text{TRUE} \\
 \qquad \qquad \qquad \text{DATA, USR\_DOWN, FILE\_ID}
 \end{array}$$

**Esercizio 02** (Punti 8, 20 minuti) Si scriva una interrogazione in SQL che fornisca coppie (Usr\_ID, File\_Id) dove Usr\_ID ha scaricato nel 2019 TUTTI i blocchi del file File\_Id.

```

CREATE VIEW V1(USR_ID, FILE_ID)
SELECT USR_ID, FILE_ID
FROM GETN_BLOCCHI_SCARICATI BS JOIN GETN_BLOCCHI_FILE B ON B.FILE_ID = BS.FILE_ID
WHERE N_BLOCCHI_DOWNLOAD = NBLOCCHI;

CREATE VIEW GETN_BLOCCHI_SCARICATI(USR_ID, FILE_ID, NBLOCCHI_DOWNLOAD)
SELECT U.USR_ID, PB.FILE_ID, COUNT(*) AS N_BLOCCHI
FROM UTENTE U NATURAL JOIN POSSIEDE_BLOCCHI PB
JOIN SCAMBIO S ON S.USER_DOWN = U.USR_ID AND S.FILE_ID = PB.FILE_ID AND PB.COD_B = S.COD_B;
GROUP BY U.USR_ID, PB.FILE_ID;

CREATE VIEW GETN_BLOCCHI_FILE(FILE_ID, NBLOCCHI)
SELECT FILE_ID, COUNT(*) FROM BLOCCHI
GROUP BY FILE_ID;

```

**Esercizio 03** (Punti 8, 20 minuti) Si implementino nel modo più adeguato i seguenti vincoli:

1. La data di possesso del file coincide con la data maggiore del possesso dei suoi blocchi;
2. Quando viene completato lo scambio di un blocco (attributo completo diviene TRUE), nella tabella POSSIEDE\_BLOCCHI vengono fatte le seguenti operazioni: 1) viene aggiunto il possesso del blocco all'utente che ha fatto il download con la data di scaricamento 2) se il possesso del file fosse già presente si aggiorna la data del possesso del file con la data dello scaricamento del blocco.
3. Due blocchi dello stesso file non possono avere lo stesso numero d'ordine.

1)

```
CREATE ASSERTION A1 CHECK NOT EXIST
SELECT * FROM POSSIEDE_FILE PF
WHERE PF.DATA < (SELECT MAX(DATA) FROM POSSIEDE_BLOCCHI
                  WHERE USR_ID = PF.USR_ID AND FILE_ID = PF.FILE_ID)
```

3)

```
ALTER TABLE BLOCCHI
ADD CONSTRAINT U1 UNIQUE (FILE_ID, ORDINE)
```

2)

```
CREATE TRIGGER T1
AFTER UPDATE OF COMPLETO ON SCAMBIO
WHEN (OLD.COMPLETO = FALSE AND NEW.COMPLETO = TRUE)
FOR EACH ROW
BEGIN
```

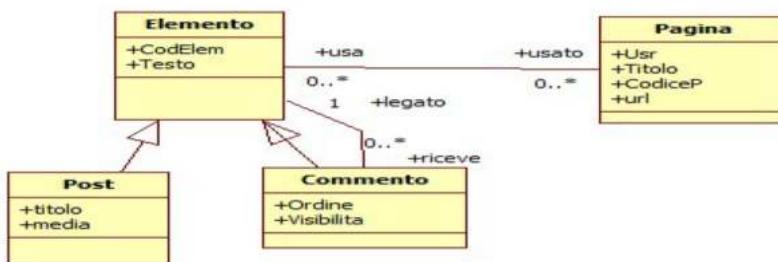
```
INSERT INTO POSSIEDE_BLOCCHI VALUES
(OLD.USER_DOWN, OLD.COD_B, OLD.FILE_ID, OLD.DATA);
```

```
IF EXIST (SELECT * FROM POSSIEDE_FILE PF WHERE PF.USR_ID = OLD.USER_DOWN AND
          PF.FILE_ID = OLD.FILE_ID)
THEN
```

```
UPDATE POSSIEDE_FILE
SET DATA = OLD.DATA
WHERE USR_ID = OLD.USER_DOWN AND FILE_ID = OLD.FILE_ID;
```

```
END IF
```

```
END
```



**Esercizio 01** (Punti 9, 30 minuti) Si consideri la bozza di Class Diagram in figura che descrive pagine di post e di commenti a post in un sistema di comunicazione social. Si scriva prima lo schema logico per il class diagram e si definiscano poi le tabelle.

POST(CODPO, TESTO, TITOLO, MEDIA)

COMMENTO(CODC, TESTO, ORDINE, VISIBILITA, IDPO, IDC)

PAGINA(CODICEP, USR, TITOLO, URL)

ASS\_PO\_PA(CODPO, CODICEP)

ASS\_C\_PA(CODC, CODICEP)

```
CREATE TABLE POST(
CODPO INT NOT NULL,
TESTO VARCHAR2(1000) NOT NULL,
TITOLO VARCHAR2(200) NOT NULL,
MEDIA VARCHAR2(300) NOT NULL,
CONSTRAINT PK_POST PRIMARY KEY (CODPO );
```

```
CREATE TABLE COMMENTO(
CODC INT NOT NULL,
TESTO VARCHAR2(1000) NOT NULL,
ORDINE INT NOT NULL,
VISIBILITA VARCHAR2(100) NOT NULL,
IDPO INT NULL,
IDC INT NULL,
CONSTRAINT PK_COMMENTO PRIMARY KEY (CODC),
CONSTRAINT FK_C_PO (IDPO) REFERENCES POST (CODPO),
CONSTRAINT FK_C_C (IDC) REFERENCES COMMENTO (CODC );
```

```
CREATE TABLE PAGINA (
CODICEP INT NOT NULL,
USR VARCHAR2(300) NOT NULL,
TITOLO VARCHAR2(300) NOT NULL,
URL VARCHAR2(1000) NOT NULL,
CONSTRAINT PK_PAGINA PRIMARY KEY (CODICEP );
```

```
CREATE TABLE ASS_PO_PA (
CODPO INT NOT NULL,
CODICEP INT NOT NULL,
CONSTRAINT FK_ASS_PO (CODPO) REFERENCES POST (CODPO),
CONSTRAINT FK_ASS_PA (CODICEP) REFERENCES PAGINA (CODICEP );
```

```
CREATE TABLE ASS_C_PA (
CODC INT NOT NULL,
CODICEP INT NOT NULL,
CONSTRAINT FK_ASS_C (CODC) REFERENCES COMMENTO (CODC),
CONSTRAINT FK_ASS_PA (CODICEP) REFERENCES PAGINA (CODICEP );
```

$\text{CRUCIVERBA}(\text{CodC}, \text{MaxX}, \text{MaxY}, \text{Autore}, \text{Data}, \text{Tipo})$   
 $\text{CASELLA}(\text{CodC}, \text{X}, \text{Y}, \text{Tipo}, \text{Lettera})$   
 $\text{DEFINIZIONE}(\text{CodC}, \text{X}, \text{Y}, \text{Tipo}, \text{Testo}, \text{Soluzione})$   
 $\text{UTENTE}(\text{CodU}, \text{Nome}, \text{Cognome})$   
 $\text{SOLUZIONE}(\text{CodC}, \text{CodU}, \text{X}, \text{Y}, \text{Lettera}, \text{Ordine})$

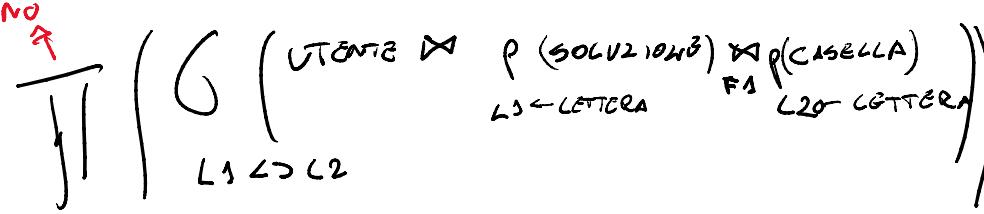
**Esercizio 02** (Punti 7) Si scriva una interrogazione SQL che restituisca per ogni utente il numero di cruciverba senza errori ma incompleti (per alcune caselle manca la soluzione).

```

CREATE VIEW V1 (UTENT, N_CRUCIVERBA)
SELECT UTENT, COUNT(S.CODC) AS N_CRUCIVERBA
FROM
(SELECT U.CODU AS UTENT, S.CODC FROM UTENTE U JOIN SOLUZIONE S ON S.CODU = U.CODU
WHERE
(SELECT COUNT(*) FROM SOLUZIONE S1 JOIN CASELLA C2 ON S1.CODC = C2.CODC AND C2.X = S1.X AND S1.Y = C2.Y
WHERE S1.CODC = S.CODC AND S1.CODU = U.CODU
AND C2.TIPO = 'BIANCA'
AND S1.LETTERA <> C2.LETTERA) = 0
AND
(SELECT COUNT(*) FROM CASELLA C1 WHERE
C1.CODC = S.CODC) -
(SELECT COUNT(*) FROM SOLUZIONE S1 JOIN CASELLA C2 ON S1.CODC = C2.CODC AND C2.X = S1.X AND S1.Y = C2.Y
AND S1.LETTERA = C2.LETTERA
WHERE
S1.CODC = S.CODC AND S1.CODU = U.CODU) > 0
GROUP BY U.CODU, S.CODC; )
GROUP BY UTENT;

```

**Esercizio 01** (Punti 8) Si scriva una interrogazione in algebra relazionale che, se valutata, restituisca per ogni utente il numero di cruciverba risolti esattamente (una soluzione è sbagliata quando una lettera associata a una cassella non corrisponde a quella prevista).



$$\prod_{\text{codu}, \text{codc}} \left( \text{UTENTE} \bowtie \rho(\text{SOLUZIONE}) \bowtie \rho(\text{CASELLA}) \right)$$

$$\text{L1} \leftrightarrow \text{LETTERA} \quad F_1 \quad \text{L2} \leftrightarrow \text{LETTERA}$$

$$F_1 = \text{SOLUZIONE}. \text{CODC} = \text{CASELLA}. \text{CODC}$$

$$\text{AND } \text{SOLUZIONE}. \text{X} = \text{CASELLA}. \text{X} \text{ AND }$$

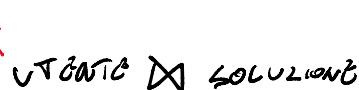
$$\text{SOLUZIONE}. \text{Y} = \text{CASELLA}. \text{Y}$$



$$\text{NO-SOL} \triangleleft \text{UTENTE} \bowtie \text{NO} \bowtie \text{SOLUZIONE}$$

$$\text{UTENTE}. \text{CODU} = \text{SOLUZIONE}. \text{CODU} \quad \text{NO}. \text{codc} = \text{SOLUZIONE}. \text{codc}$$

TUTTI



$\text{COUNT}(\text{codc})(\text{TUTTI} \setminus \text{NO-SOL})$

**Esercizio 03** (Punti 7) Si implementino nel modo più adeguato i seguenti vincoli:

1. Non si può associare più di una definizione ad una casella.
2. Se una definizione verticale comincia in una casella nera o la casella verticale è la prima o è preceduta da una casella nera (nella stessa colonna).
3. Se una definizione orizzontale è associata ad una casella non ci possono essere caselle nere nella stessa riga a partire da quella casella per tutta la lunghezza della parola soluzione.

1)  
ALTER TABLE DEFINIZIONE  
ADD CONSTRAINT U\_FK\_CASELLA UNIQUE (CODC, X, Y)

2)  
CREATE ASSERTION A1  
CHECK NOT EXIST  
SELECT \* FROM DEFINIZIONE D JOIN CASELLA C ON C.CODC = D.CODC AND D.X = C.X AND D.Y = C.Y  
WHERE D.TIPO = 'VERTICALE' AND  
C.X <> 0 AND  
(SELECT COUNT(\*) FROM CASELLA C1 WHERE C1.CODC = D.CODC AND D.Y = C1.Y AND D.X - 1 = C1.X  
AND C1.TIPO = 'BIANCO') > 0

3)  
CREATE ASSERTION A1  
CHECK NOT EXIST  
SELECT \* FROM DEFINIZIONE D JOIN CASELLA C ON C.CODC = D.CODC AND D.X = C.X AND D.Y = C.Y  
WHERE D.TIPO = 'ORIZZONTALE' AND  
(SELECT COUNT(\*) FROM CASELLA C1 WHERE C1.CODC = D.CODC AND D.X = C1.X AND  
C1.Y > C.Y AND C1.Y <= LENGTH(D.SOLUZIONE))  
O si fa così--(SELECT MAX(Y) FROM SOLUZIONE WHERE CODC = C1.CODC AND X = C1.X)  
AND C1.TIPO = 'NERO') > 0

**Esercizio 04** (Punti 7) Si scriva un trigger che viene eseguito quando viene inserita una nuova definizione per un cruciverba. Il trigger inserisce tutte le caselle del cruciverba collegate alla definizione nel seguente modo. a) usando la soluzione della definizione associa una casella bianca con la lettera per la casella fornita dalla soluzione. Esempio, la soluzione 'CASA' per la definizione orizzontale per la cassella (X=1, Y=4) porta ad associare alla 'C' alla casella (1,4), 'A' alla casella (2,4), 'S' alla casella (3,4) e 'A' alla casella (4,4). Nell'inserire le caselle e i loro valori si presti attenzione a non duplicare le caselle (si inserisce solo se non già presente). Se la casella è già inserita e il valore è diverso da quello da inserire si segnala un errore, b) dopo aver inserito le caselle associate alla definizione, si inserisce una casella nera nella casella precedente e successiva alla definizione (se non esiste già una cassella nera e se la posizione non è iniziale o terminale).

```

CREATE TRIGGER T1
AFTER INSERT ON DEFINIZIONE
FOR EACH ROW

C CHAR(1) := "";
S DEFINIZIONE.SOLUZIONE%TYPE := NEW.SOLUZIONE;

VARX INT := NEW.X;
VARY INT := NEW.Y;

BEGIN

IF NEW.TIPO LIKE 'ORIZZONTALE'
THEN

WHILE VARX - NEW.X <= LENGTH(NEW.SOLUZIONE)
LOOP

C := SUBSTR(S, 1, 2);
S := SUBSTR(S, 2);

IF EXIST (SELECT * FROM CASELLA WHERE CODC = NEW.CODC AND
X = VARX AND Y = VARY AND LETTERA <> C)
THEN
RAISE EXCEPTION

```

```

ELSE

IF NOT EXIST (SELECT * FROM CASELLA WHERE CODC = NEW.CODC AND
X = VARX AND Y = VARY AND LETTERA = C)
THEN

INSERT INTO CASELLA VALUES(NEW.CODC, VARX, VARY, 'BIANCA', C);

END IF
END IF

VARX := VARX + 1;
END LOOP

IF NEW.X - 1 >= 0 AND NOT EXIST (SELECT * FROM CASELLA WHERE CODC = NEW.CODC AND
X = NEW.X - 1 AND Y = NEW.Y AND TIPO = 'NERA')
THEN
INSERT INTO CASELLA VALUES(NEW.CODC, NEW.X - 1, NEW.Y, 'NERA', NULL);

END IF

IF VARX <= (SELECT MAXX FROM CRUCIVERBA WHERE CODC = NEW.CODC) AND NOT EXIST (SELECT * FROM CASELLA WHERE
CODC = NEW.CODC AND
X = VARX AND Y = NEW.Y AND TIPO = 'NERA')
THEN
INSERT INTO CASELLA VALUES(NEW.CODC, VARX, NEW.Y, 'NERA', NULL);

END IF

ELSE

WHILE VARY - NEW.Y <= LENGTH(NEW.SOLUZIONE)
LOOP

C := SUBSTR(S, 1, 2);
S := SUBSTR(S, 2);

IF EXIST (SELECT * FROM CASELLA WHERE CODC = NEW.CODC AND
X = VARX AND Y = VARY AND LETTERA <> C)
THEN
RAISE EXCEPTION
ELSE

IF NOT EXIST (SELECT * FROM CASELLA WHERE CODC = NEW.CODC AND
X = VARX AND Y = VARY AND LETTERA = C)
THEN

INSERT INTO CASELLA VALUES(NEW.CODC, VARX, VARY, 'BIANCA', C);

END IF
END IF

VARY := VARY + 1;
END LOOP

IF NEW.Y - 1 >= 0 AND NOT EXIST (SELECT * FROM CASELLA WHERE CODC = NEW.CODC AND
X = NEW.X AND Y = NEW.Y - 1 AND TIPO = 'NERA')
THEN
INSERT INTO CASELLA VALUES(NEW.CODC, NEW.X, NEW.Y - 1, 'NERA', NULL);

END IF

IF VARY <= (SELECT MAXY FROM CRUCIVERBA WHERE CODC = NEW.CODC) AND NOT EXIST (SELECT * FROM CASELLA WHERE

```

```

CODC = NEW.CODC AND
X = NEW.X AND Y = VARY AND TIPO = 'NERA')
THEN
INSERT INTO CASELLA VALUES(NEW.CODC, NEW.X, VARY, 'NERA', NULL);

END IF

```

```
END
```

**Esercizio 05** (*Punti 7*) Si scriva una funzione che riceve in ingresso una stringa contenente nomi di autore di cruciverba separati tra loro dal simbolo -. Si scriva una interrogazione in SQL dinamico che recupera i cruciverba creati almeno da uno degli autori. La funzione restituisce una stringa contenente i codici di cruciverba separati dal simbolo -.

```

CREATE FUNCTION F1 ( STRINGA IN VARCHAR2(1000) )
RETURN VARCHAR2(1000)

COMANDO := 'SELECT CODC FROM CRUCIVERBA WHERE AUTORE IN (';

S VARCHAR2(1000) := STRINGA;
P VARCHAR(200) := ":";

FLAG INT := 0;
C := CRUCIVERBA.CODC%TYPE;
RISULTATO VARCHAR2(1000) := "";

CURSOR C1
IS
EXECUTE IMMEDIATE COMANDO;

BEGIN

WHILE S <> ''
LOOP
FLAG := 1;
P := SUBSTR(S, 1, INSTR(S, ',', 1));
S := SUBSTR(S, INSTR(S, ',', 1));

COMANDO := COMANDO || P || ',';

END LOOP

IF FLAG <> 0
THEN
COMANDO := SUBSTR(COMANDO, 1, LENGTH(COMANDO) - 1);
COMANDO := COMANDO || ');';

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO C

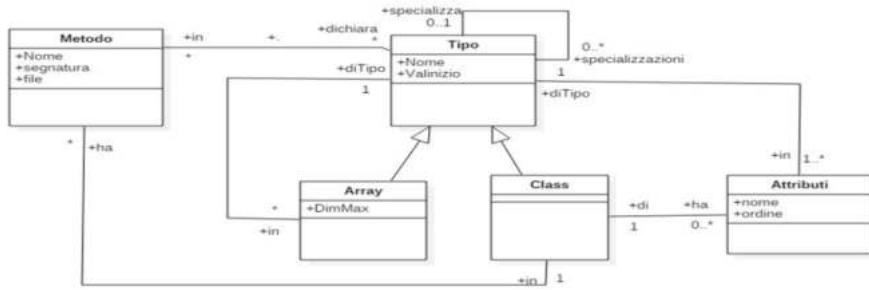
RISULTATO := RISULTATO || C || '-';

END LOOP
END IF

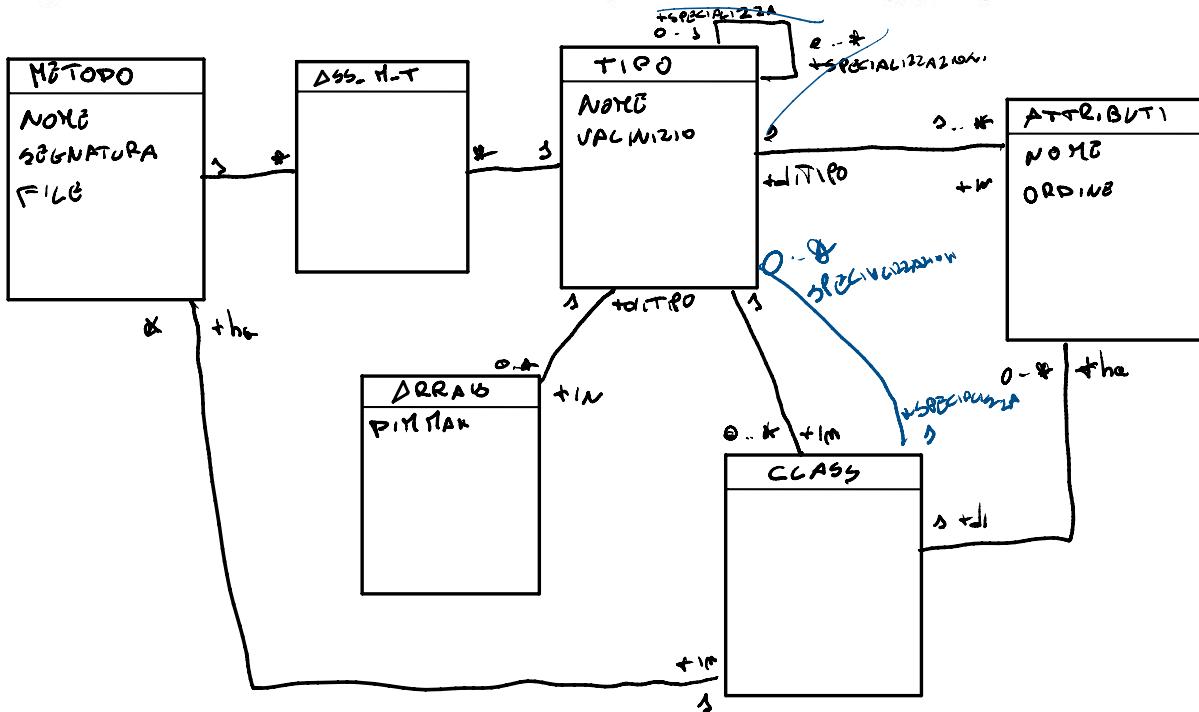
RISULTATO := SUBSTR(RISULTATO, 1, LENGTH(RISULTATO) - 1);

```

RETURN RISULTATO;  
END



1. Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (7 punti);
  2. Si forniscano gli schemi relazionali per il class diagram ristrutturato (7 punti);
  3. Si definisca usando SQL le tabelle (5 punti);
  4. Nella definizione delle tabelle si scrivano i vincoli: (a) due attributi della stessa classe non possono avere lo stesso nome; (b) solo un tipo class può essere la specializzazione di un altro tipo nella relazione ricorsiva. Si dica di che tipo di vincoli si tratta (4 punti)



METODO(CODM (PK), NOME, SEGNATURA, FILE, CODC (FK))

## ASS\_M\_T(CODM (FK), CODT (FK))

TIPO (CODT (PK), NOME, VAL INIZIO, IDTIPOS (FK))

ATTRIBUTI(CODAT (PK), NOME, ORDINE, CODT (FK), CODC (FK))

CLASS(CODC (PK), CODT (FK))

ARRAY(CODAR (PK), DIMMAX, CODT (FK))

```
CREATE TABLE METODO (
    CODM INT NOT NULL,
    NOME VARCHAR2(200) NOT NULL,
    SEGNATURA VARCHAR2(200) NOT NULL,
    FILE VARCHAR2(300) NOT NULL,
```

```
CODC INT NOT NULL,  
CONSTRAINT PK_M PRIMARY KEY (CODM),  
CONSTRAINT FK_M_C FOREIGN KEY (CODC) REFERENCES CLASS (CODC );
```

```
CREATE TABLE ASS_M_T (  
CODM INT NOT NULL,  
COTD INT NOT NULL,  
CONSTRAINT FK_ASS_M FOREIGN KEY (CODM) REFERENCES METODO (CODM),  
CONSTRAINT FK_ASS_T FOREIGN KEY (COTD) REFERENCES TIPO (COTD );
```

```
CREATE TABLE TIPO (  
COTD INT NOT NULL,  
NOME VARCHAR2(200),  
VALINIZIO INT NOT NULL,  
IDTIPOS INT NULL,  
CONSTRAINT PK_T PRIMARY KEY (COTD),  
CONSTRAINT FK_T_T FOREIGN KEY (IDTIPOS) REFERENCES TIPO (COTD);  
VINCOLO INTRARELAZIONALE
```

```
CREATE TABLE ATTRIBUTI (  
CODAT INT NOT NULL,  
NOME VARCHAR2(200) NOT NULL,  
ORDINE INT NOT NULL,  
COTD INT NOT NULL,  
CODC INT NULL,  
CONSTRAINT PK_AT PRIMARY KEY (CODAT),  
CONSTRAINT FK_AT_T FOREIGN KEY (COTD) REFERENCES TIPO (COTD),  
CONSTRAINT FK_AT_C FOREIGN KEY (CODC) REFERENCES CLASS (CODC),  
CONSTRAINT U_NOME UNIQUE (NOME));  
VINCOLO INTERRELAZIONALE
```

```
CREATE TABLE CLASS (  
CODC INT NOT NULL,  
COTD INT NOT NULL,  
CONSTRAINT PK_C PRIMARY KEY (CODC),  
CONSTRAINT FK_C_T FOREIGN KEY (COTD) REFERENCES TIPO (CODC );
```

```
CREATE TABLE ARRAY (  
CODAR INT NOT NULL,  
DIMMAX INT NOT NULL,  
COTD INT NOT NULL,  
CONSTRAINT PK_AR PRIMARY KEY (CODAR),  
CONSTRAINT FK_AR_T FOREIGN KEY (COTD) REFERENCES TIPO (CODC );
```

**Esercizio 01 (8 punti)** Si scriva una interrogazione in algebra relazionale che fornisca se valutata il codice degli alberi in cui ogni nodo (tranne le foglie) ha esattamente due discendenti.

$$\begin{aligned}
 & \text{cont} \\
 & \left( \text{CODA, CODN} \in \text{COUNT(*)} \left( \text{AC} \setminus \text{COMPARCHI} \right) \right) \Bigg) \\
 & \quad \text{N-FIGLIO} \quad \text{COPN=PADRE} \\
 & \quad \text{CODA=CODA} \\
 & \text{AC} \leftarrow (\text{ALBERI} \setminus \text{COMPNODI}) \\
 & \text{S1} \leftarrow (\text{ALBERI}, \text{NO}) \\
 & \prod_{\text{CODA}} \left( \text{CONT} \left( \text{NFIGLIO} \text{ AND } \text{N-FIGLIO} \right) \right) \\
 & \quad \text{COPN} \\
 & \prod_{\text{CODA}} (\text{S1})
 \end{aligned}$$

**Esercizio 02 (8 punti)** Si scriva una interrogazione in SQL che fornisca coppie di codici di alberi tali che il secondo elemento della coppia è un sottoalbero del primo elemento della coppia (ogni nodo e ogni arco del sottoalbero deve essere nodo e arco, rispettivamente, del primo albero).

```

CREATE VIEW (A1, A2)
SELECT A1.CODA, A2.CODA FROM ALBERO A1, ALBERO A2
WHERE
(SELECT COUNT(*) FROM ALBERI NATURAL JOIN COMPNODI WHERE CODA = A2.CODA) -
(SELECT COUNT(*) FROM ALBERI NATURAL JOIN COMPNODI WHERE CODA = A2.CODA AND
CODN IN (SELECT CODN FROM COMPNODI WHERE CODA = A1.CODA)) = 0
AND
(SELECT COUNT(*) FROM ALBERI NATURAL JOIN COMPARCHI WHERE CODA = A2.CODA) -
(SELECT COUNT(*) FROM ALBERI NATURAL JOIN COMPARCHI WHERE CODA = A2.CODA AND
FIGLIO IN (SELECT FIGLIO FROM COMPARCHI WHERE CODA = A1.CODA) AND PADRE IN (SELECT
PADRE FROM COMPARCHI WHERE CODA = A1.CODA)) = 0
AND A1.CODA <> A2.CODA

```



**Esercizio 03 (9 punti)** Si esprima nel modo più adeguato il seguente insieme di vincoli:

- Ogni nodo di un albero ha al più un padre;
- La radice di un albero non ha padre;
- I nodi padre e figlio di un arco associato ad una albero devono essere nodi dell'albero;
- Quando viene rimosso un nodo da un grafo (cancellazione dalla tabella COMPNODI) devono essere rimossi tutti gli archi e tutti i nodi del sottoalbero radicato nel nodo (discendenti);

1)

```

ALTER TABLE COMPARCHI
ADD CONSTRAINT UK_PADRE UNIQUE (PADRE, FIGLIO)

```

2)

```

CREATE ASSERTION A1 CHECK NOT EXIST
(SELECT * FROM ALBERI A JOIN COMPARCHI CA ON CA.CODA = A.CODA AND CA.FIGLIO = A.RADICE

```

```
WHERE CA.PADRE IS NOT NULL)
```

3)

```
CREATE ASSERTION A2
CHECK NOT EXIST
SELECT * FROM COMPARCHI CA WHERE
CA.PADRE IS NOT NULL AND
(CA.PADRE NOT IN (SELECT CODN FROM COMPNODI WHERE CODA = CA.CODA)
OR CA.FIGLIO NOT IN (SELECT CODN FROM COMPNODI WHERE CODA = CA.CODA))
```

4)

```
CREATE TRIGGER T1
BEFORE DELETE ON COMPNODI
FOR EACH ROW
```

```
NODO NODI.CODN%TYPE := OLD.CODN;
```

```
CURSOR C1
IS
SELECT FIGLIO
FROM COMPARCHI WHERE PADRE = OLD.CODN
AND CODA = OLD.CODA;
```

```
BEGIN
```

```
OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO NODO
```

```
DELETE FROM COMPNODI
WHERE CODA = OLD.CODA AND CODN = NODO;
```

```
END LOOP
```

```
DELETE FROM COMPARCHI
WHERE CODA = OLD.CODA AND PADRE = OLD.CODN;
```

```
END
```

**Esercizio 04 (9 punti)** Si scriva una funzione PLSQL che riceve in ingresso il codice di un albero ed il codice di un nodo. La funzione restituisce la lunghezza del cammino massimo nel sottoalbero radicato nel nodo dato e il numero di foglie del sottoalbero radicato nel nodo dato. Si svolga l'esercizio con programmazione iterativa (non ricorsiva). Ci si avvalga di una struttura temporanea TMP(codA, CodN, altezza) che si suppone sia già definita che deve essere istanziata inserendo iterativamente ogni nodo del sottoalbero e l'altezza del nodo dal nodo radice del sottoalbero.

```
CREATE FUNCTION F1 ( CA ALBERI.CODA%TYPE, CN NODI.CODN%TYPE )
RETURN VARCHAR2(1000)
--TMP(CODA, CODN, ALTEZZA)
```

```
DELETE FROM TMP;
INSERT INTO TMP (CA, CN, 0);
```

```

FLAG INT := 0;
H INT := 1;
NODOATTUALE NODI.CODN%TYPE := CN;
N NODI.CODN%TYPE;

CURSOR C1
IS
SELECT FIGLIO FROM COMPARCHI
WHERE CODA = CA AND PADRE = NODOATTUALE;

R INT := 0;
R1 INT := 0;
RISULTATO VARCHAR2(1000) := "";

CURSOR C2
IS
SELECT CODN FROM TMP
WHERE ALTEZZA = H - 1;

BEGIN

WHILE H = (SELECT MAX(ALTEZZA) FROM TMP) - 1
LOOP

OPEN C2
WHILE C2%FOUND
LOOP
FETCH C2 INTO NODOATTUALE

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO N

INSERT INTO TMP(CA, N, H);

END LOOP

END LOOP

H := H + 1;
END LOOP

SELECT MAX(ALTEZZA) INTO R FROM TMP WHERE CODA = CA;

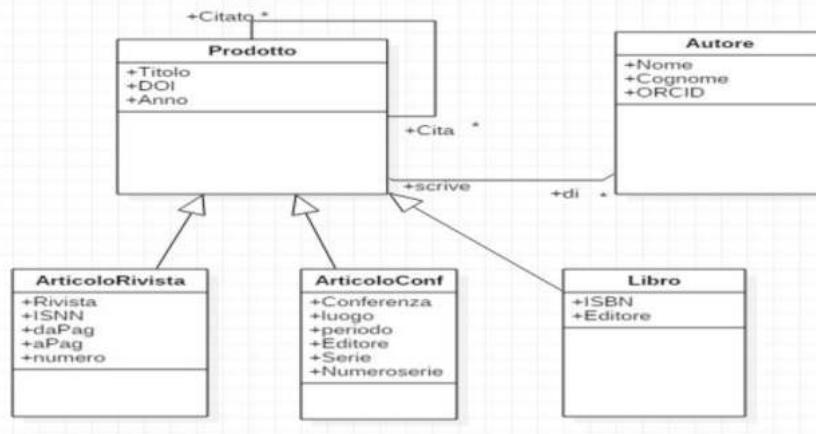
SELECT COUNT(*) INTO R1 FROM TMP WHERE NOT EXISTS
(SELECT * FROM COMPARCHI C WHERE C.CODA = CA AND C.PADRE = CODN);

RISULTATO := RISULTATO || R || ',' || R1;
RETURN RISULTATO;

END

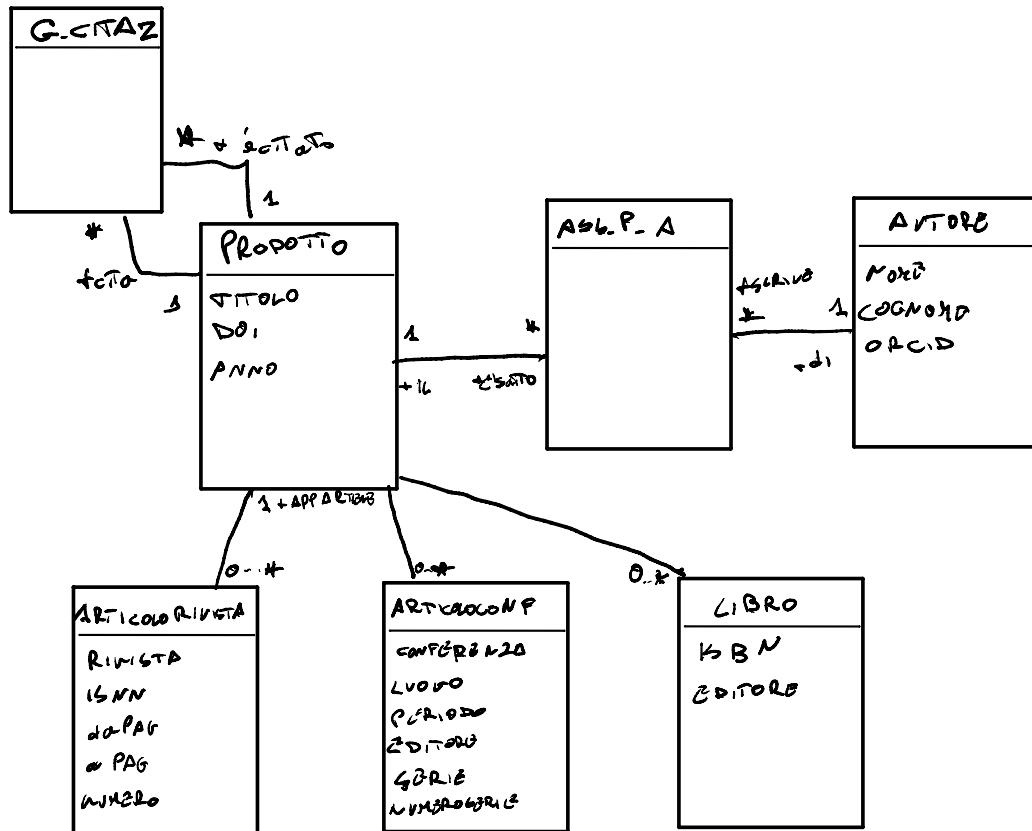
```

---



**Esercizio 11** Si consideri la bozza di Class Diagram in figura che descrive una base di dati per la memorizzazione di pubblicazioni. La specializzazione è di tipo parziale disgiunta (vi sono altre pubblicazioni oltre a quelle oggetto di specializzazione).

1. Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (7 punti);
2. Si forniscano gli schemi relazionali per il class diagram ristrutturato (7 punti);
3. Si definisca usando SQL le tabelle (5 punti);



*STUDENTE(Matricola, Nome, Cognome)*

*PIANI(CodPiano, Matricola, Data)*

*COMPOSIZIONE(CodPiano, CodI, Anno, Voto, Lode)*

*INSEGNAM(CodI, Titolo, CFU, Tipo)*

*ANNI(CodEsame, Anno)*

*PROPED(CodEsame, CodEsameProp)*

*VERBALEESAME(CodV, Data, CodI, CodDocente)*

*ESAMIVERBALE(CodV, Matricola, Voto, Lode)*

**Esercizio 01** (punti 8) Si scriva una espressione in algebra relazionale che, se valutata, fornisca il nome e cognome e la matricola degli studenti che hanno un piano di studi dove TUTTI gli insegnamenti facoltativi presenti NON hanno richiesta di propedeuticità.

$$\text{INSEGNAM-FAC} \quad \bigcap \left( \begin{array}{l} \text{P, ANI} \rightarrow \text{COMPOSIZIONE} \rightarrow \text{INSEGNAM} \\ \text{I.TIPO = 'FACOLTATIVO'} \end{array} \right)$$

$$\text{NO} \leftarrow \overline{\prod}_{\text{COD PIANO, MATRICOLA}} \left( \begin{array}{l} \text{INSEGNAM-FAC} \rightarrow \text{PROPED} \\ \text{I.CODI} = \text{P. CODESAME} \end{array} \right) \quad \text{TUTTI} \leftarrow \overline{\prod}_{\text{COD PIANO, MATRICOLA}} \left( \begin{array}{l} \text{INSEGNAM-FAC} \\ \text{I.TIPO = 'FACOLTATIVO'} \end{array} \right)$$

$$\text{SI} \leftarrow (\text{TUTTI} \setminus \text{NO}) \quad \overline{\prod}_{\text{ANNO, CODICE, MATRICOLA}} (\text{SI} \rightarrow \text{STUPONTO})$$

**Esercizio 02** (punti 8) Definire nel modo più semplice il seguente insieme di vincoli:

- lo stesso insegnamento non deve comparire più di una volta in un piano di studi;
- nella tabella PROPED ad un esame possono essere associati solo insegnamenti effettivamente presenti nella tabella INSEGNAM;
- la somma dei CFU degli insegnamenti presenti in ogni anno in un piano di studi deve essere esattamente 60 CFU.
- ogni piano di studi deve contenere tutti gli esami obbligatori.

1)

```
ALTER TABLE COMPOSIZIONE
ADD CONSTRAINT U_INSEGNAMENTO UNIQUE (CODPIANO, CODI)
```

2)

```
CREATE ASSERTION CK_ESAME
CHECK NOT EXIST
SELECT * FROM PROPED P WHERE
```

-- VERSIONE 1

```
P.CODESAME NOT IN (SELECT CODI FROM INSEGNAM) OR
P.CODESAMERPROP NOT IN (SELECT CODI FROM INSEGNAM)
```

-- VERSIONE 2

```
NOT EXIST (SELECT CODI FROM INSEGNAM WHERE CODI = P.CODESAME) OR
```

```
NOT EXIST (SELECT CODI FROM INSEGNAM WHERE CODI = P.CODESAMEPROP)
```

3)

```
CREATE ASSERTION CK_CFU
CHECK NOT EXIST
(SELECT C.CODPIANO, C.ANNO, SUM(I.CFU) FROM INSEGNAM I NATURAL JOIN COMPOSIZIONE C
GROUP BY C.CODPIANO, C.ANNO
HAVING SUM(I.CFU) <> 60)
```

-- OPPURE

```
CREATE ASSERTION CK_CFU
CHECK
(SELECT SUM(I.CFU) FROM INSEGNAM I NATURAL JOIN COMPOSIZIONE C
GROUP BY C.CODPIANO, C.ANNO) = 60
```

4)

```
CREATE ASSERTION CK_PIANO_PROP
CHECK NOT EXIST
```

```
SELECT C.CODPIANO, COUNT(*)
FROM COMPOSIZIONE C NATURAL JOIN INSEGNAM I
WHERE I.TIPO = 'OBBLIGATORIO'
HAVING COUNT(*) <> (SELECT COUNT(*) FROM INSEGNAM WHERE TIPO = 'OBBLIGATORIO')
```

**Esercizio 03** (punti 8) Si scriva un trigger che viene attivato quando viene inserito un nuovo verbale di esame. L'effetto del trigger è il seguente. Per ogni esame presente nel verbale

- si controlla che l'insegnamento sia presente nel piano di studi dello studente;
- si controlla che tutti gli insegnamenti propedeutici siano stati sostenuti (presenti con voto nel piano di studi) dallo studente;
- si registra il voto e la data nel piano di studi (se i controlli hanno effetto positivo)

```
CREATE TRIGGER T1
```

```
AFTER INSERT ON VERBALESAME
FOR EACH ROW
```

```
CURSOR C1
```

```
IS
```

```
SELECT MATRICOLA FROM ESAMIVERBALE
WHERE CODV = OLD.CODV
```

```
M STUDENTE.MATRICOLA%TYPE;
```

```
BEGIN
```

```
IF NOT EXIST (SELECT * FROM INSEGNAM WHERE CODI = OLD.CODI)
```

```
THEN
```

```
RAISE EXCEPTION
```

```
ELSE
```

```
OPEN C1
```

```
WHILE C1%FOUND
```

```
LOOP
```

```
FETCH C1 INTO M
```

```
IF NOT EXIST (SELECT * FROM PIANI P NATURAL JOIN COMPOSIZIONE C
WHERE C.CODI = OLD.CODI AND P.MATRICOLA = M)
```

```

THEN
RAISE EXCEPTION
ELSE

IF NOT EXIST
(SELECT * FROM PIANI P NATURAL JOIN COMPOSIZIONE C
 WHERE P.MATRICOLA = M AND
 C.CODI IN
 (SELECT CODESAMEPROP FROM PROPED WHERE CODESAME IN
 (SELECT C2.CODI FROM PIANI P2 NATURAL JOIN COMPOSIZIONE C2 WHERE P2.MATRICOLA = M)
 AND C.VOTO IS NULL)
RAISE EXCEPTION
ELSE

UPDATE COMPOSIZIONE
SET VOTO = OLD.VOTO, LODE = OLD.LODE
WHERE CODPIANO = (SELECT CODPIANO FROM PIANI WHERE MATRICOLA = M) AND CODI = OLD.CODI;

END IF
END LOOP

END IF
END

```

**Esercizio 04 (Punti 9)** Si scriva una procedura in PL\SQL che riceve in ingresso la matricola di uno studente e un nuovo codice di piano di studi (non ancora presente nella base di dati). La procedura crea un piano di studi avente come codice quello passato per parametro. Il piano di studi include tutti gli insegnamenti obbligatori e tutti gli insegnamenti di default. Ciascun esame è collocato nel primo anno possibile.

```

CREATE PROCEDURE P1
( MATR STUDENTE.MATRICOLA%TYPE, CP PIANI.CODPIANO%TYPE )

CURSOR C1
IS
SELECT CODI FROM INSEGNAM
WHERE TIPO IN ('OBBLIGATORIO', 'DEFAULT');

MINAN ANNI.ANNO%TYPE;
I INSEGNAM.CODI%TYPE;

BEGIN

INSERT INTO PIANI VALUES (CP, MATR, SYSDATE());

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO I

SELECT MIN(ANNO) INTO MINAN FROM ANNI WHERE CODESAME = I;

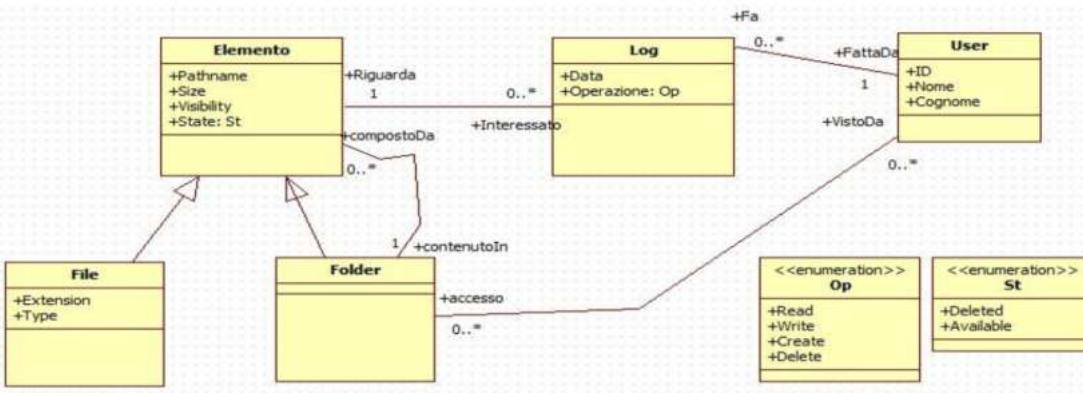
INSERT INTO COMPOSIZIONE VALUES (CP, I, MINAN, NULL, NULL);

END LOOP

END

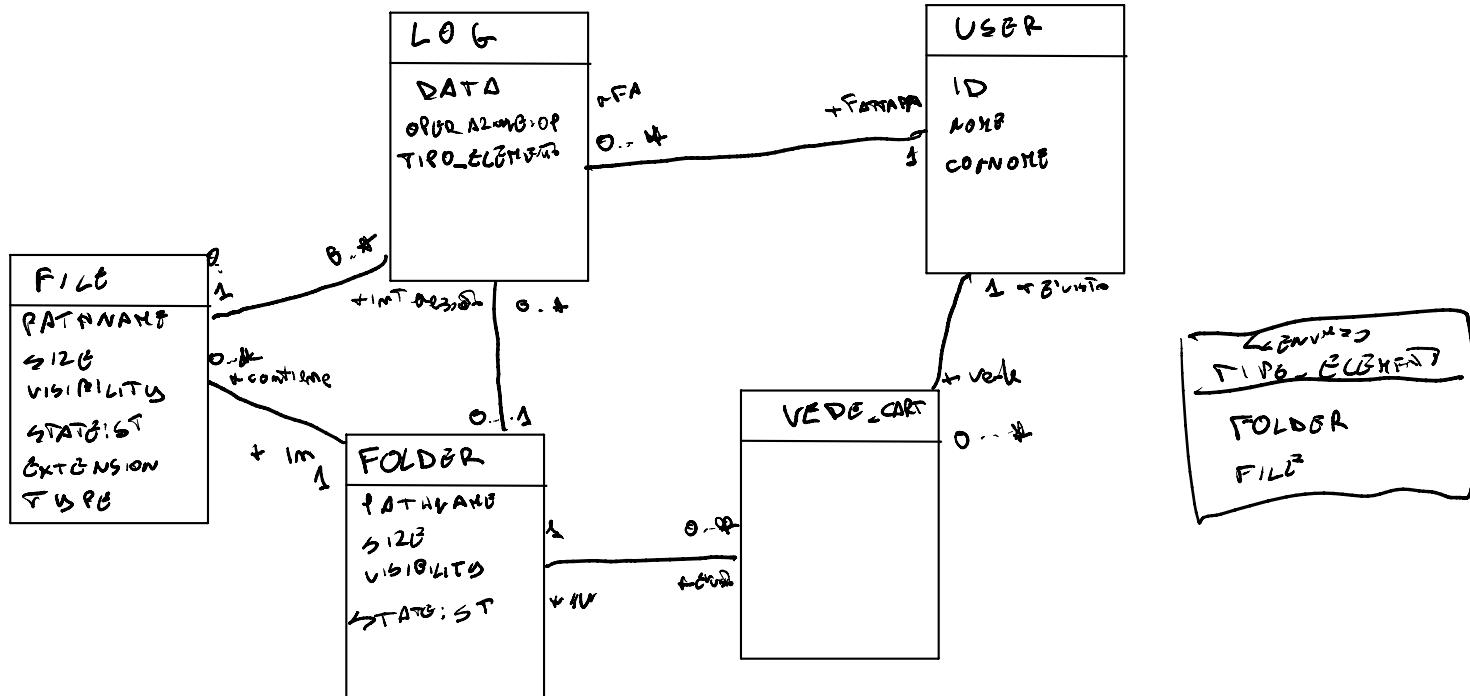
```

---



**Esercizio 11** Si consideri il class diagram riportato in figura che descrive un file system tradizionale con elementi del tipo File o Cartella con la usuale strutturazione ad albero delle cartelle. Un utente ha una relazione di visibilità sulle cartelle (le cartelle a cui può accedere). In una struttura di log sono memorizzate le operazioni fatte dagli utenti sugli elementi.

- Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (6 punti);
- Si forniscano gli schemi relazionali per il class diagram ristrutturato (7 punti);
- Si definisca usando SQL le tabelle (5 punti);
- Nella definizione delle tabelle si scrivano i vincoli: (a) Visibility FALSE quando State DELETED; (b) due elementi contenuti nello stesso folder hanno pathname diversi; (c) si dica il tipo dei due vincoli. (4 punti)



LA TABELLA VEDE\_CART PUÒ NON ESISTERE PER ALCUNE CARTELLE ED ALCUNI UTENTI

UN FILE NON È OBBLIGATO A SITUARE IN UNA CARTELLA

I LOG POSSONO OPERARE SU FILE O CARTELLE, MA UN SINGOLO LOG PUÒ OPERARE O SU UN SINGOLO FILE O SU UNA SINGOLA CARTELLA E NON SU ENTRAMBI CONTEMPORANEAMENTE.

FILE (CODOFILE (PK), PATHNAME, SIZE, VISIBILITY, ST, EXTENSION, TYPE, CODFOLDER (FK))

FOLDER (CODOFILE (PK), PATHNAME, SIZE, VISIBILITY, ST)

LOG (DATA, OP, TIPO\_ELEMENTO, IDUTENTE (FK), CODFILE (FK), CODFOLDER (FK))

USER (ID (PK), NOME, COGNOME)

VEDE\_CART (CODFOLDER (FK), IDUTENTE (FK))

```
CREATE TABLE FILE (
    CODFILE INT NOT NULL,
    PATHNAME VARCHAR2(1000) NOT NULL,
    SIZE INT NOT NULL,
    VISIBILITY BOOLEAN NOT NULL,
    ST VARCHAR2(300) NOT NULL,
    EXTENSION VARCHAR2(400) NOT NULL,
    TYPE VARCHAR2(1000) NOT NULL,
    CODFOLDER INT NULL,
    CONSTRAINT PK_FILE PRIMARY KEY (CODFILE),
    CONSTRAINT CK_ST CHECK ST IN ('DELETED', 'AVAILABLE'),
    CONSTRAINT FK_FOLDER FOREIGN KEY (CODFOLDER) REFERENCES FOLDER (CODFOLDER),
    CONSTRAINT UK_PATHNAME UNIQUE (CODFOLDER, PATHNAME));
```

```
CREATE TABLE FOLDER (
    CODFOLDER INT NOT NULL,
    PATHNAME VARCHAR2(1000) NOT NULL,
    SIZE INT NOT NULL,
    VISIBILITY BOOLEAN NOT NULL,
    ST VARCHAR2(300) NOT NULL,
    CONSTRAINT PK_FOLDER PRIMARY KEY (CODFOLDER),
    CONSTRAINT CK_ST CHECK ST IN ('DELETED', 'AVAILABLE'));
```

```
CREATE TABLE LOG (
    DATA DATE NOT NULL,
    OP VARCHAR2(600) NOT NULL,
    TIPO_ELEMENTO VARCHAR2(500) NOT NULL,
    IDUTENTE INT NOT NULL,
    CODFILE INT NULL,
    CODFOLDER INT NULL,
    CONSTRAINT FK_L_UTENTE FOREIGN KEY IDUTENTE REFERENCES USER (ID),
    CONSTRAINT FK_L_FILE FOREIGN KEY CODFILE REFERENCES FILE (CODFILE),
    CONSTRAINT FK_L_FOLDER FOREIGN KEY CODFOLDER REFERENCES FOLDER (CODFOLDER),
    CONSTRAINT CK_OP CHECK OP IN ('READ', 'WRITE', 'CREATE', 'DELETE'),
    CONSTRAINT CK_TIPO_ELEMENTO CHECK TIPO_ELEMENTO IN ('FOLDER', 'FILE'));
```

```
CREATE ASSERTION CK_ELEMENTI
CHECK NOT EXIST
SELECT * FROM LOG WHERE CODFILE IS NOT NULL AND CODFOLDER IS NOT NULL;
```

```
CREATE ASSERTION CK_ELEMENTI_FOLDER
CHECK NOT EXIST
SELECT * FROM LOG WHERE CODFILE IS NOT NULL AND OP = 'FOLDER';
```

```
CREATE ASSERTION CK_ELEMENTI_FILE
CHECK NOT EXIST
SELECT * FROM LOG WHERE CODFOLDER IS NOT NULL AND OP = 'FILE';
```

```
CREATE TABLE USER (
    ID INT NOT NULL,
    NOME VARCHAR2(500) NOT NULL,
    COGNOME VARCHAR2(700) NOT NULL,
    CONSTRAINT PK_USER PRIMARY KEY ID );
```

```
CREATE TABLE VEDE_CART (
    CODFOLDER INT NOT NULL,
    IDUTENTE INT NOT NULL,
    CONSTRAINT FK_VC_FOLDER FOREIGN KEY CODFOLDER REFERENCES FOLDER (CODFOLDER),
    CONSTRAINT FK_VC_USER FOREIGN KEY IDUTENTE REFERENCES USER (IDUTENTE));
```

```

CREATE ASSERTION VINCOLO1_FILE
CHECK NOT EXIST
SELECT * FROM FILE WHERE VISIBILITY = TRUE AND ST = 'DELETED';

```

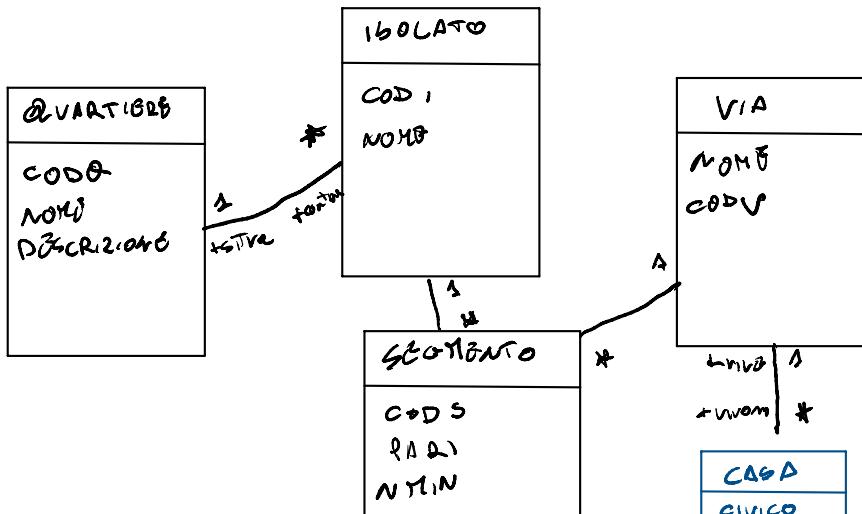
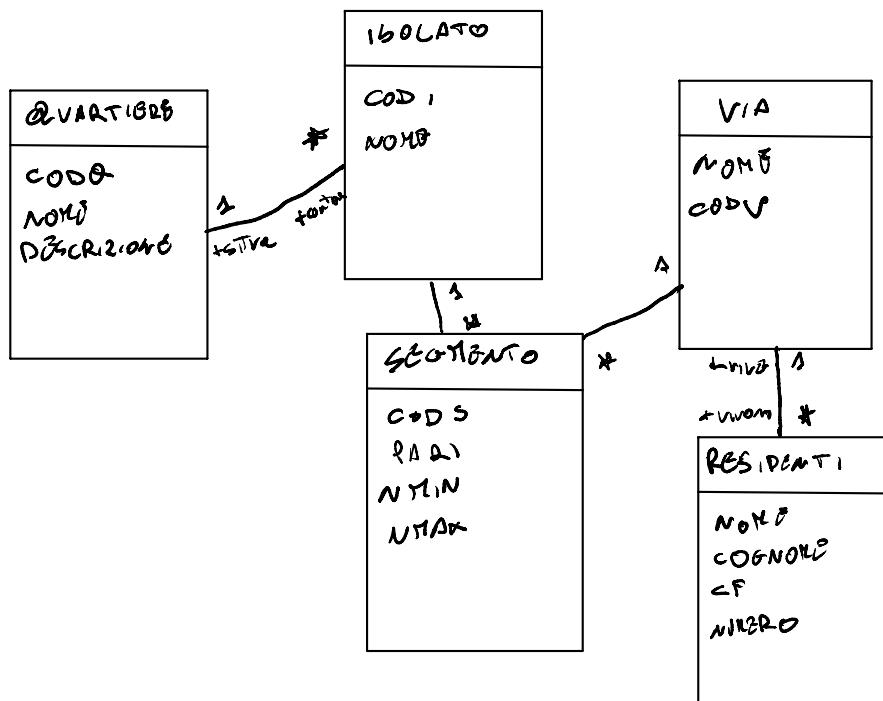
```

CREATE ASSERTION VINCOLO1_FOLDER
CHECK NOT EXIST
SELECT * FROM FOLDER WHERE VISIBILITY = TRUE AND ST = 'DELETED';

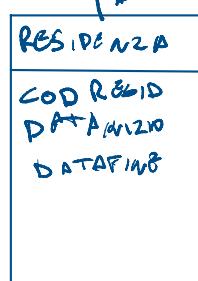
```

IL SECONDO E' UN VINCOLO INTRARELAZIONALE MENTRE IL PRIMO E' DI ENNUPLA

$\text{QUARTIERE}(\underline{\text{CodQ}}, \text{Nome}, \text{Descrizione})$   
 $\text{ISOLATO}(\underline{\text{CodI}}, \underline{\text{CodQ}}, \text{Nome})$   
 $\text{VIA}(\underline{\text{CodV}}, \text{Nome})$   
 $\text{SEGMENTO}(\underline{\text{CodS}}, \underline{\text{CodV}}, \underline{\text{CodI}}, \text{Pari}, \text{NMin}, \text{NMax})$   
 $\text{RESIDENTI}(\text{CF}, \text{Nome}, \text{Cognome}, \underline{\text{CodV}}, \text{Numero})$

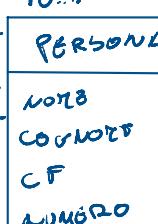


PAZI  
NTEIN  
NTIAK



Figlio Di  
0..1

Figlio Di



Figlio Di  
0..1

Figlio Di  
0..1

Figlio Di  
0..1

Figlio Di  
0..1

*SEZIONE(Documento, CodSezione, Titolo, Testo)*  
*STRUTTURA(Documento, SezContenente, SezContenuta, Posizione)*  
*PAROLE(Parola)*  
*PRESENZE(Parola, Documento, Sezione, PosParola)*

**Esercizio 01** (Punti 8) Si scriva una espressione dell'algebra relazionale che se valutata fornisca tutti i documenti in cui sono presenti tutte le parole chiave.

$$\begin{aligned}
 \text{CONTAPAROLE\_DOC} &= \exists_{\text{DOCUMENTO}} \sum_{n \in \text{PAROLE}} \text{COUNT}(x) \left( \prod_{\text{PAROLA}, \text{DOCUMENTO}} (\text{PAROLE} \in \text{DOCUMENTO}) \right) \\
 \text{CONTAPAROLE} &= \exists \left( \sum_{n \in \text{PAROLE}} \text{COUNT}(x) (\text{PAROLE}) \right) \\
 &\quad \prod_{\text{DOCUMENTO}} (\text{CONTAPAROLE\_DOC} = \text{CONTAPAROLE})
 \end{aligned}$$

**Esercizio 02** (Punti 7) Si scrivano nel modo più opportuno i seguenti vincoli:

- se per una sezione è presente una sottosezione in posizione  $i > 1$ , allora deve essere presente per la stessa sezione anche una sottosezione in posizione  $i - 1$ .
- Due sezioni dello stesso documento devono avere titolo diverso.
- La cancellazione di una parola chiave da PAROLE deve avere come conseguenza la cancellazione di tutte le indicazioni di presenza di quella parola nelle sezioni.

1)

CREATE ASSERTION

CHECK NOT EXIST

SELECT \* FROM SEZIONE S JOIN STRUTTURA ST ON ST.DOCUMENTO = S.DOCUMENTO AND

ST.SEZCONTENENTE = S.CODSEZIONE

WHERE ST.POSIZIONE &gt; 1

AND NOT EXIST

(SELECT \* FROM STRUTTURA WHERE DOCUMENTO = S.DOCUMENTO AND SEZCONTENENTE = S.CODSEZIONE AND SEZCONTENUTA &lt;&gt; ST.SEZCONTENUTA AND POSIZIONE = ST.POSIZIONE - 1)

2)

ALTER TABLE SEZIONE

ADD CONSTRAINT U\_TITOLO UNIQUE (DOCUMENTO, TITOLO);

3)

CREATE TRIGGER T1

BEFORE DELETE ON PAROLE

FOR EACH ROW

BEGIN

DELETE FROM PRESENZE WHERE PAROLA = OLD.PAROLA;

END

**Esercizio 03** (Punti 9) Si scriva un trigger che viene azionato quando viene inserita una nuova parola chiave nella tabella PAROLA. Il trigger provvede ad aggiornare la tabella PRESENZE aggiungendo le occorrenze della nuova parola nelle sezioni. Si usi la funzione INSTR per la ricerca della parola nella sezione ricordando che una parola può occorrere più volte nel testo di una sezione.

```
CREATE TRIGGER
AFTER INSERT ON PAROLE
FOR EACH ROW

CURSOR C1
IS
SELECT DOCUMENTO, CODSEZIONE, TESTO
FROM SEZIONE
GROUP BY DOCUMENTO, CODSEZIONE;

D SEZIONE.DOCUMENTO%TYPE;
CS SEZIONE.CODSEZIONE%TYPE;
T SEZIONE.TESTO%TYPE;
OFFSET INT;

BEGIN

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO D, CS, T

OFFSET := 1;

WHILE INSTR(T, OFFSET, NEW.PAROLA) <> 0
LOOP

OFFSET := INSTR(T, OFFSET, NEW.PAROLA);
INSERT INTO PRESENZE (NEW.PAROLA, D, CS, OFFSET);
OFFSET := OFFSET + LENGTH(PAROLA);

END LOOP

END LOOP

END
```

**Esercizio 04** (Punti 8) Si scriva una vista che per ogni documento ed ogni parola chiave riporti le seguenti informazioni: il numero di sezioni del documento in cui occorre la parola, il numero di occorrenze della parola in tutto il documento, la sezione del documento in cui la parola ha più occorrenze.

```
CREATE VIEW (DOCUMENTO, P_CHIAVE, N_SEZIONI, N_OCCORR, SEZ)
SELECT DOCUMENTO, PCHIAVE, NSEZIONI, NOCCORR, MAXSEZ FROM
```

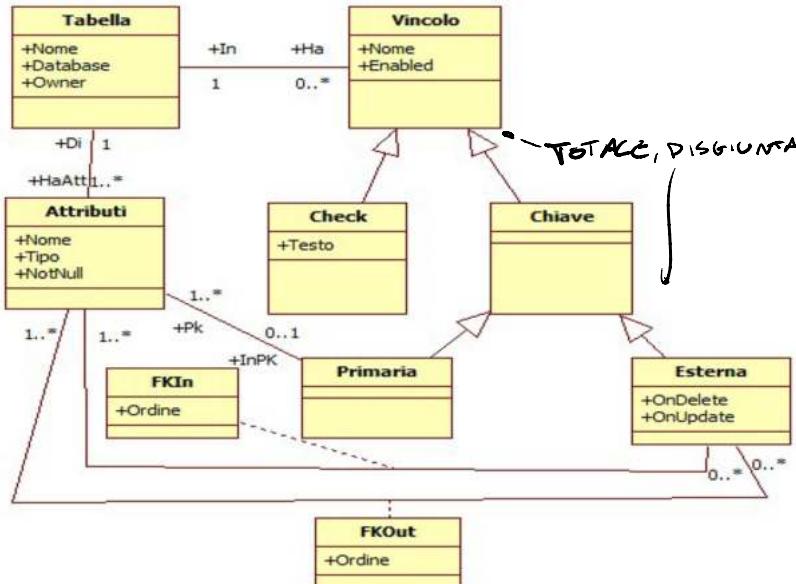
GETNSEZ NJ GETNOCCORR NJ GETSEZ  
GROUP BY DOCUMENTO, PCHIAVE;

```
CREATE VIEW GETNSEZ (DOCUMENTO, PCHIAVE, NSEZIONI)
SELECT PR.DOCUMENTO, PR.PAROLA, COUNT(DISTINCT PR.SEZIONE) FROM PRESENZE PR
GROUP BY PR.DOCUMENTO, PR.PAROLA
```

```
CREATE VIEW GETNOCCORR (DOCUMENTO, PCHIAVE, NOCCORR)
SELECT PR.DOCUMENTO, PR.PAROLA, COUNT(*) FROM PRESENZE PR
GROUP BY PR.DOCUMENTO, PR.PAROLA
```

```
CREATE VIEW GETNOCORRSEZ (DOCUMENTO, PCHIAVE, SEZ, NOS)
SELECT PR.DOCUMENTO, PR.PAROLA, PR.SEZIONE, COUNT(*) FROM PRESENZE PR
GROUP BY PR.DOCUMENTO, PR.PAROLA, PR.SEZIONE
```

```
CREATE VIEW GETSEZ (DOCUMENTO, PCHIAVE, MAXSEZ)
SELECT DOCUMENTO, PAROLA, SEZ FROM
(SELECT DOCUMENTO, PAROLA, SEZ, MAX(NOS) FROM
GETNOCCORRSEZ
GROUP BY DOCUMENTO, PAROLA, SEZ)
```

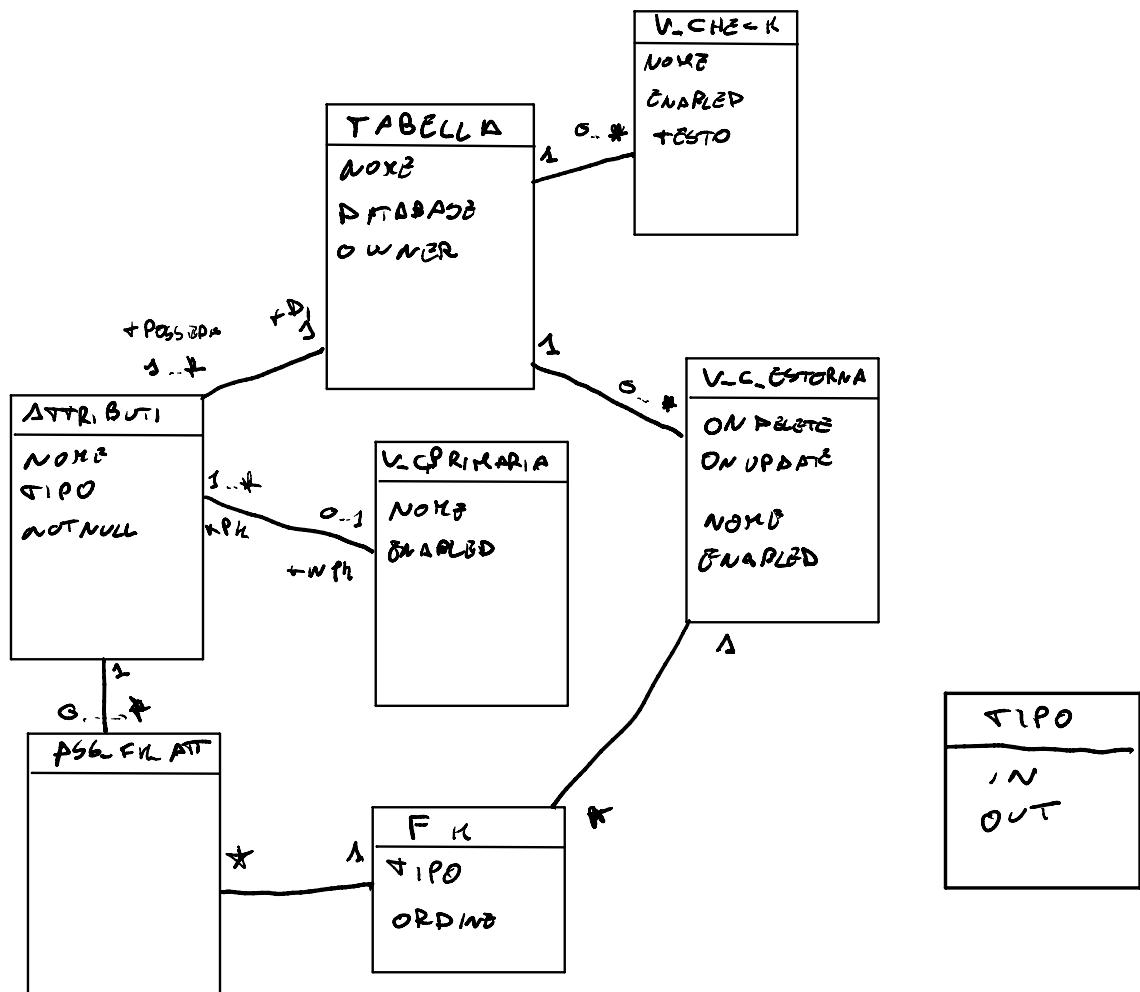


**Esercizio 11** Il class diagram riportato in figura descrive un sistema informativo per la descrizione di tabelle. Una tabella è identificata dal nome e dal database, un attributo dalla tabella e dal nome, un vincolo dal nome e dalla tabella. Per una chiave esterna l'associazione *FKin* fornisce la lista degli attributi della chiave esterna e *FKout* la lista degli attributi referenziati.

1. Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (8 punti);
  2. Si forniscano gli schemi relazionali per il class diagram ristrutturato (6 punti);
  3. Si definisca usando SQL le tabelle (5 punti);
  4. Nella definizione delle tabelle si scrivano i vincoli: (a) Un attributo associato a una chiave primaria non è nullo; (b) I valori associabili agli attributi *Ondelete* e *OnUpdate* sono *NoAction*, *Cascade* e *SetDefault*. (c) Si dica di che tipo sono i due vincoli precedenti. (4 punti)

**Esercizio 11** Il class diagram riportato in figura descrive un sistema informativo per la descrizione di tabelle. Una tabella è identificata dal nome e dal database, un attributo dalla tabella e dal nome, un vincolo dal nome e dalla tabella. Per una chiave esterna l'associazione *FKin* fornisce la lista degli attributi della chiave esterna e *FKout* la lista degli attributi referenziati.

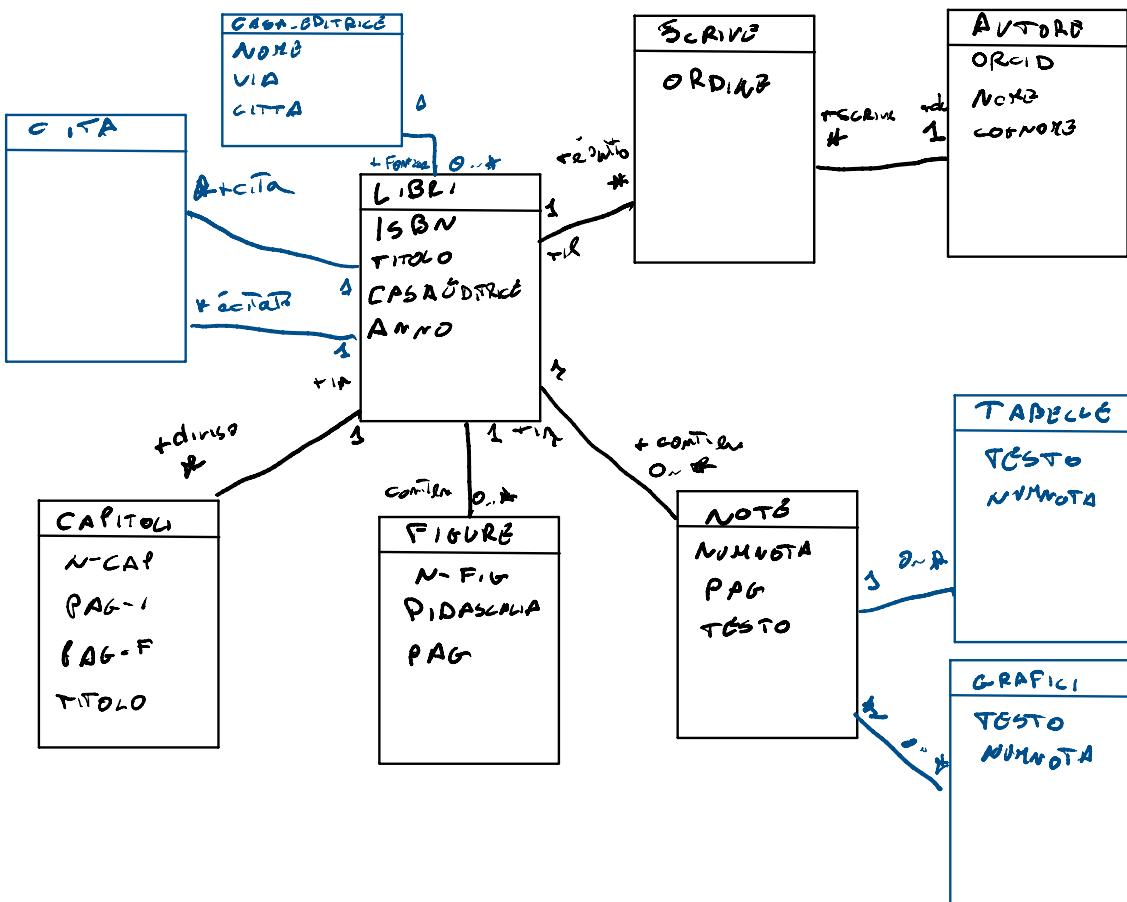
- Si ristrutturi il class diagram per renderlo adeguato ad una traduzione nel modello dei dati relazionale indicando testualmente gli eventuali vincoli di ristrutturazione (8 punti);
- Si forniscano gli schemi relazionali per il class diagram ristrutturato (6 punti);
- Si definisca usando SQL le tabelle (5 punti);
- Nella definizione delle tabelle si scrivano i vincoli: (a) Un attributo associato a una chiave primaria non è nullo; (b) I valori associabili agli attributi *OnDelete* e *OnUpdate* sono *NoAction*, *Cascade* e *SetDefault*. (c) Si dica di che tipo sono i due vincoli precedenti. (4 punti)



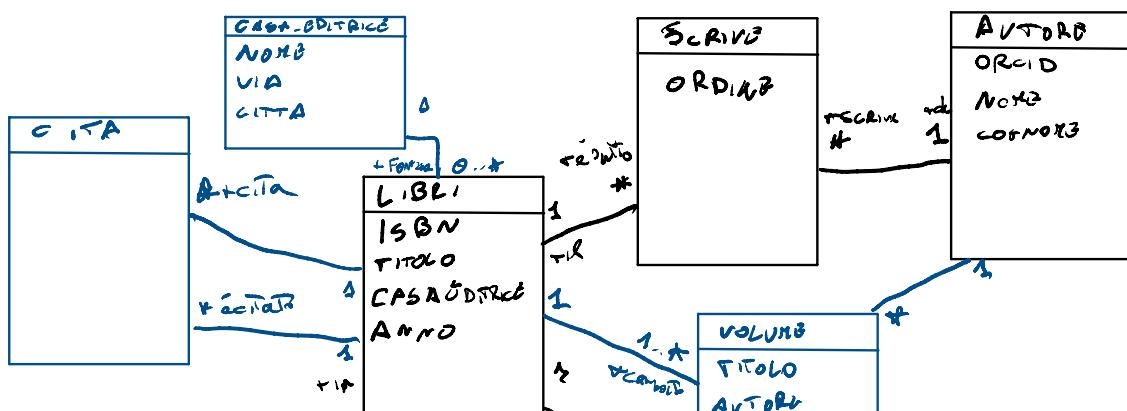
**Esercizio 12** Gli schemi relazionali di seguito riportati descrivono una base di dati per la descrizioni di libri. Dei libri viene riportata informazione sui capitoli, le figure presenti nei capitoli e le note associate alle pagine.

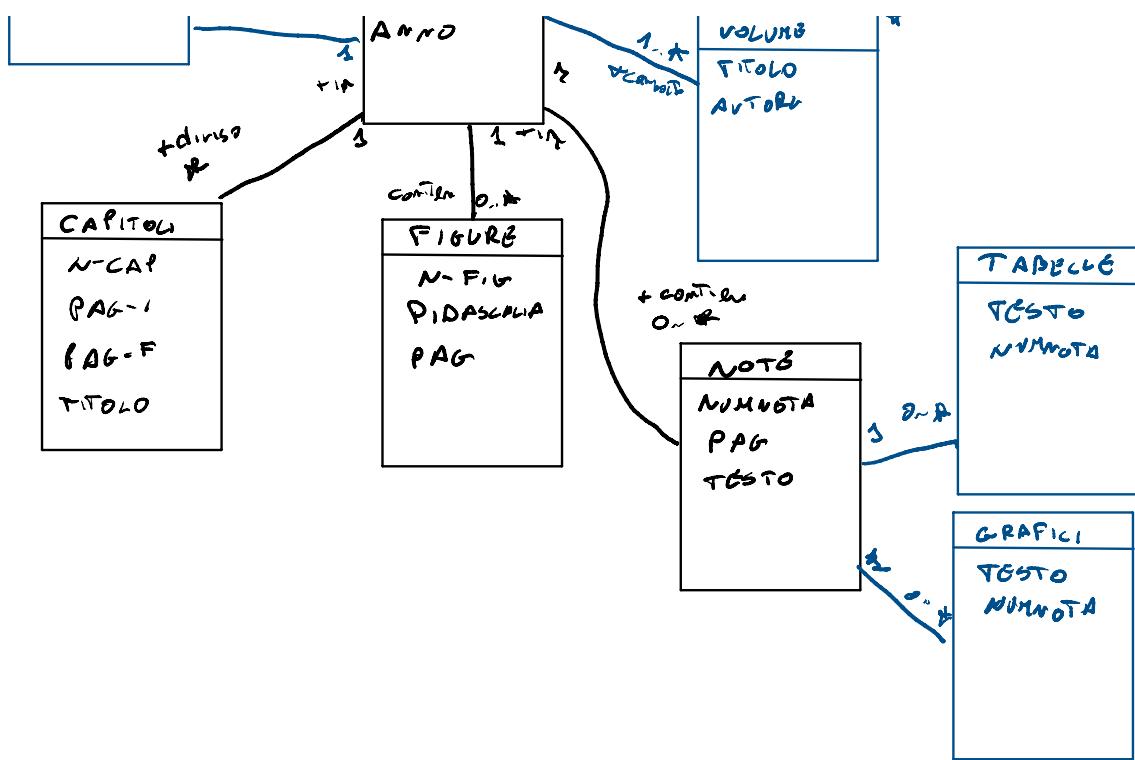
**LIBRI**(ISBN, TITOLO, CASAEDITRICE, ANNO)  
**AUTORE**(ORCID, NOME, COGNOME)  
**SCRIVE**(ORCID, ISBN, ORDINE)  
**CAPITOLI**(ISBN, N - CAP, PAG - I, PAG - F, TITOLO)  
**FIGURE**(ISBN, N - FIG, DIDASCALIA, PAG)  
**NOTE**(ISBN, NUMNOTA, PAG, TESTO)

- Si fornisca un Class Diagram di progettazione per lo schema relazionale (5 punti);
- Si estenda il class diagram ottenuto per permettere di: tenere conto delle citazioni (libri che nella bibliografia citano altri libri); informazioni sulle tabelle o grafici presenti nel testo; informazioni aggiuntive sulla casa editrice; avere libri che sono costituiti da più volumi ciascuno con titolo e autori. (5 punti).



v2





*MAGAZZINO(CodA, Descrizione, Collocazione, Quantita, SogliaMin)*  
*ORDINE(CodO, DataOrdine, CodFornitore, DataArrivo)*  
*COMPORDINE(CodA, CodO, Quantita)*  
*FORNITORE(PI, RagSociale, Via, Citta)*  
*CATALOGO(CodA, PI, Prezzo)*

**Esercizio 01** (Punti 8) Si scriva una interrogazione in algebra relazionale che, se valutata, fornisce il codice degli ordini e la ragione sociale dei fornitori per ordini non ancora arrivati e che sono urgenti

$$\begin{array}{l}
 \text{ART-OLI-LIMITATI} \\
 \rightarrow \overline{\Pi}_{\text{CODO}} \left( \sigma_{\text{QUANTITA' < SOLLISTIN}} (\text{MAGAZZINO}) \right) \\
 \\ 
 \text{ART-NON-LIMITATI} = \overline{\Pi}_{\text{CODO}} \left( (\text{MAGAZZINO}) \setminus \text{ARTCOL-LIMITATI} \right) \\
 \\ 
 \text{ORD-POR} \rightarrow \overline{\Pi}_{\text{CODO}, \text{CODFORNITORE}} \left( \sigma_{\text{DATAARRIVO IS NULL}} (\sigma_{\text{ORDGIUSTI}} (\text{ORDINE})) \right) \\
 \\ 
 \overline{\Pi}_{\text{CODO}, \text{PI}} \left( \sigma_{\text{CODFORNITORE} = \text{PI}} (\text{COMPORDINE} \bowtie \text{ORD-NON-LIMITATI}) \right)
 \end{array}$$

**Esercizio 02** (Punti 7) Si scriva una vista Riepilogo(Anno, PI, N-Ordini, Tot-Articoli, Saldo) che per ogni anno e ogni partita iva indica il numero di ordini indirizzati a quel fornitore in quell'anno, il numero di articoli complessivi ordinati al fornitore in quell'anno, il costo complessivo degli ordini in quell'anno al fornitore.

```
CREATE VIEW RIEPILOGO (ANNO, PI, N_ORDINI, TOT_ARTICOLI, SALDO)
SELECT ANNO, PI, N_ORDINI, TOT_ARTICOLI, COSTO FROM
GETNORDINI NJ GETTOTARTICOLI NJ GETSALDO
```

```
CREATE VIEW GETNORDINI (ANNO, PI, N_ORDINI)
SELECT YEAR(O.DATAORDINE), F.PI, COUNT(*) FROM ORDINE O JOIN FORNITORE F ON O.CODFORNITORE = F.PI
GROUP BY YEAR(O.DATAORDINE), F.PI
```

```
CREATE VIEW GETTOTARTICOLI (ANNO, PI, TOT_ARTICOLI)
SELECT YEAR(O.DATAORDINE), F.PI, COUNT(*) FROM ORDINE O JOIN FORNITORE F ON O.CODFORNITORE = F.PI
JOIN COMPORDINE CO ON CO.CODO = O.CODO
GROUP BY YEAR(O.DATAORDINE), F.PI
```

```
CREATE VIEW GETSALDO (ANNO, PI, COSTO)
SELECT YEAR(O.DATAORDINE), CA.PI, SUM(CA.PREZZO) FROM ORDINE O
JOIN COMPORDINE CO ON CO.CODO = O.CODO
JOIN CATALOGO CA ON CA.CODA = CO.CODO AND CA.PI = O.CODFORNITORE
GROUP BY YEAR(O.DATAORDINE), CA.PI
```

**Esercizio 03** (Punti 5) Si esprima nel modo più adeguato il seguente vincolo: Quando viene inserita la data di arrivo dell'ordine viene aggiornata automaticamente la quantità degli articoli in magazzino (alla quantità esistente si somma quella dell'ordine per ogni articolo);

```
CREATE TRIGGER T1
AFTER UPDATE OF DATAARRIVO ON ORDINE
WHEN (OLD.DATAARRIVO IS NULL AND NEW.DATAARRIVO IS NOT NULL)
FOR EACH ROW

CURSOR C1
IS
SELECT DISTINCT CODA, QUANTITA FROM COMPORDINE
WHERE CODO = OLD.CODO

IDARTICOLO MAGAZZINO.CODA%TYPE;
Q COMPORDINE.QUANTITA%TYPE;
OLDQ COMPORDINE.QUANTITA%TYPE;

BEGIN

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO IDARTICOLO, Q

SELECT QUANTITA INTO OLDQ FROM MAGAZZINO WHERE CODA = IDARTICOLO;

UPDATE MAGAZZINO
SET QUANTITA := OLDQ + Q
WHERE CODA = IDARTICOLO

END LOOP

END
```

**Esercizio 04** (Punti 8) Si scriva una procedura che riceve in ingresso una stringa di codici di articolo separati dal simbolo separatore # e un intero k. La procedura controlla per ogni fornitore qual è il prezzo complessivo per un ordine di k esemplari di ogni articolo nella lista e sceglie il fornitore che ha il prezzo più vantaggioso. La procedura dunque inserisce nel database un ordine al fornitore scelto per tutti gli articoli. Supponendo che il codice dell'ordine sia un intero, il codice dell'ordine da inserire è l'intero successivo rispetto a quello già usato.

```
CREATE PROCEDURE
( STRINGA VARCHAR2(1000), K INT )

F FORNITORE.PI%TYPE;
S VARCHAR2(1000) := STRINGA;
P VARCHAR2(200) := "";

PR INT := 0;
PREZZO INT := 0;
```

N ARTICOLO  
DIVERSI

```

PMIN INT := 99999999999;
FMIN FORNITORE.PI%TYPE;
OMAX ORDINE.CODO%TYPE;

CURSOR C1
IS
SELECT PI FROM FORNITORE

BEGIN

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO F
PREZZO := 0;
S := STRINGA;
P := "";
WHILE S <> ""
LOOP
IF INSTR(S, '#', 1) = 0 THEN P := S; S := "";
ELSE
    P := SUBSTR (S, 1, INSTR(S, '#', 1) - 1);
    S := SUBSTR (S, INSTR(S, '#', 1) + 1);
END IF

SELECT PREZZO * K INTO PR FROM CATALOGO WHERE PI = F AND CODA = P;
PREZZO := PREZZO + PR;
END LOOP

IF PREZZO < PMIN
THEN
    PMIN := PREZZO;
    FMIN := F;
END IF
END LOOP

SELECT MAX(CODO) INTO OMAX FROM ORDINE;

INSERT INTO ORDINE VALUES (OMAX, SYSDATE(), FMIN, NULL);

S := STRINGA;
P := "";
WHILE S <> ""
LOOP
IF INSTR(S, '#', 1) = 0 THEN P := S; S := "";
ELSE
    P := SUBSTR (S, 1, INSTR(S, '#', 1) - 1);
    S := SUBSTR (S, INSTR(S, '#', 1) + 1);
END IF

INSERT INTO COMPORDINE(P, OMAX, K);
END LOOP

END

```

**Esercizio 05** (Punti 6) Utilizzando SQL DINAMICO Si scriva una procedura che riceve in ingresso il nome di una tabella e una stringa di nomi di attributi separati dal carattere separatore #. La funzione ha l'effetto di aggiungere alla tabella passata per parametro un (unico) vincolo di unicità che coinvolge tutti gli attributi passati nella stringa.

```
CREATE PROCEDURE P1
(TABELLA USER_TABLE.TABLENAMES%TYPE, STRINGA VARCHAR2(1000))
```

```
C VARCHAR2(1000) := 'ALTER TABLE ' || TABELLA || ' ADD CONSTRAINT UQ UNIQUE (';  
S VARCHAR2(1000) := STRINGA;  
P VARCHAR2(300) := '';
```

```
BEGIN
```

```
WHILE S <> ""
LOOP
IF INSTR(S, '#', 1) = 0 THEN P := S; S := "";
ELSE
```

```
P := SUBSTR (S, 1, INSTR(S, '#', 1) - 1);
S := SUBSTR (S, INSTR(S, '#', 1) + 1);
```

```
END IF
```

```
C := C || P || ',';
```

```
END LOOP
```

```
C := SUBSTR(C, 1, LENGTH(C) - 1);
C := C || ');';
```

```
EXECUTE IMMEDIATE C;
END
```

*FILE(File\_ID, Path, Dimensione, Nome)*  
*BLOCCHI(Cod\_B, File\_ID, Ordine, Dimensione)*  
*UTENTE(Usr\_ID, email, NickName, IP, MaxTras)*  
*POSSIEDE\_BLOCCHI(Usr\_ID, COD\_B, File\_ID, data)*  
*POSSIEDE\_FILE(Usr\_ID, File\_ID, data)*  
*SCAMBIO(User\_UP, User\_Down, File\_ID, Cod\_B, Data, Ora, Completo)*  
*CODA(User\_ID, File\_ID, Cod\_B, Data, Ora)*

**Esercizio 03 (Punti 8, 20 minuti) Si implementino nel modo più adeguato i seguenti vincoli:**

1. *Un utente non deve avere un numero di trasferimenti in corso in upload superiore al suo valore MaxTras;*
2. *Quando viene completato lo scambio di un blocco (attributo completo diviene TRUE), nella tabella POSSIEDE\_BLOCCHI vengono fatte le seguenti operazioni: 1) viene aggiunto il possesso del blocco all'utente che ha fatto il download con la data di scaricamento 2) si inserisce il possesso del file in POSSIEDE\_BLOCCHI se non ancora presente. 3) se il possesso del file fosse già presente si aggiorna la data del possesso del file con la data dello scaricamento del blocco.*
3. *Un utente non può essere in coda di attesa per più di un blocco del medesimo file.*

1)

CREATE ASSERTION A1

CHECK NOT EXIST

(SELECT \* FROM

(SELECT USER\_UP, COUNT(\*) AS N\_TRASF FROM SCAMBIO WHERE COMPLETO = 'FALSE'

GROUP BY USER\_UP)

JOIN UTENTE U ON USER\_UP = U.USR\_ID

WHERE N\_TRASF > U.MAXTRASF)

3)

ALTER TABLE CODA

ADD CONSTRAINT U\_CODA\_ATTESA UNIQUE (USER\_ID, FILE\_ID);

2)

CREATE TRIGGER T1

AFTER UPDATE OF COMPLETO ON SCAMBIO

FOR EACH ROW

WHEN (OLD.COMPLETO = FALSE AND NEW.COMPLETO = TRUE)

BEGIN

INSERT INTO POSSIEDE\_BLOCCHI VALUES (OLD.USER\_DOWN, OLD.COD\_B, OLD.FILE\_ID, OLD.DATA);

IF NOT EXIST (SELECT \* FROM POSSIEDE\_FILE WHERE USR\_ID = OLD.USER\_DOWN AND FILE\_ID = OLD.FILE\_ID)  
THEN

INSERT INTO POSSIEDE\_FILE VALUES (OLD.USER\_DOWN, OLD.FILE\_ID, OLD.DATA);

ELSE

UPDATE POSSIEDE\_FILE SET DATA = OLD.DATA WHERE USR\_ID = OLD.USER\_DOWN AND FILE\_ID = OLD.FILE\_ID;

END

Esercizio 03 (Punti 8, 20 minuti) Si implementino nel modo più adeguato i seguenti vincoli:

1. La dimensione di un file deve essere uguale alla somma delle dimensioni dei suoi blocchi.
2. Quando viene completato lo scambio di un blocco (attributo completo diviene TRUE), nella tabella POSSIEDE\_BLOCCHI vengono fatte le seguenti operazioni: 1) viene aggiunto il possesso del blocco all'utente che ha fatto il download con la data di scaricamento 2) si inserisce il possesso del file in POSSIEDE\_BLOCCHI se non ancora presente. 3) se il possesso del file fosse già presente si aggiorna la data del possesso del file con la data dello scaricamento del blocco.
3. Due blocchi dello stesso file non possono avere lo stesso numero d'ordine.

1)

```
CREATE ASSERTION A1
CHECK NOT EXIST
SELECT * FROM
(SELECT FILE_ID AS FI, SUM(DIMENSIONE) AS DIM FROM BLOCCHI GROUP BY FILE_ID)
JOIN FILE F ON F.FILE_ID = FI
WHERE F.DIMENSIONE <> DIM;
```

3)

```
ALTER TABLE BLOCCHI
ADD CONSTRAINT U_ORDINE UNIQUE (FILE_ID, ORDINE);
```

Esercizio 02 (Punti 8, 20 minuti ) Si scriva una interrogazione in SQL che per ogni utente fornisca il numero di file completi posseduti (possiede tutti i blocchi).

```
CREATE VIEW V1 (USR_ID, N_FILE_COMPLETI)
SELECT DISTINCT USR_ID, COUNT(FILE_ID)
FROM
(SELECT USR_ID, FILE_ID FROM POSSIEDE BLOCCHI PB
WHERE
(SELECT COUNT(*) FROM BLOCCHI B WHERE B.FILE_ID = PB.FILE_ID) -
(SELECT COUNT(*) FROM POSSIEDE_BLOCCHI PB1 ON PB1.USR_ID = PB.USR_ID AND PB1.FILE_ID = PB.FILE_ID) = 0
GROUP BY USR_ID, FILE_ID)
GROUP BY USR_ID
```

→ CREATE VIEW CONFILE (FILE\_ID, N\_BLOCCHI)  
AS  
SELECT FILE\_ID, COUNT(\*)  
FROM BLOCCI  
GROUP BY FILE\_ID.

↓  
SELECT FILE\_ID, USER\_ID  
FROM POSSIEDE\_BLOCCI P  
GROUP BY FILE\_ID, USER\_ID  
HAVING COUNT(BLOCK\_ID) =  
(SELECT N\_BLOCCHI -  
FROM CONFILE H  
WHERE H.FILE\_ID = FILE\_ID) AND  
H.USER\_ID = USER\_ID

Esercizio 02 (Punti 8, 20 minuti ) Si scriva una interrogazione in SQL che fornisca coppie (Usr\_ID, File\_Id) dove Usr\_ID ha scaricato nel 2019 TUTTI i blocchi del file File\_Id.

```
CREATE VIEW SCARICA2019 (USR_ID, FILE_ID, NB)
SELECT USR_DOWN, FILE_ID, COUNT(*) FROM
SCARICA
WHERE YEAR(DATA) = 2019
GROUP BY USR_DOWN, FILE_ID;
```

```
CREATE VIEW GETNBLOCCHIFILE (FILE_ID, NBLOCCHI)
SELECT FILE_ID, COUNT(*) FROM BLOCCI
GROUP BY FILE_ID;
```

```
CREATE VIEW V1 (USR_ID, FILE_ID)
SELECT USR_ID, FILE_ID FROM
SCARICA2019 NATURAL JOIN GETNBLOCCHIFILE
WHERE NB = NBLOCCHI;
```

Esercizio 02 (Punti 8, 20 minuti ) Si scriva una interrogazione in SQL che restituisce gli identificativi dei file che hanno TUTTI i loro blocchi in coda di attesa.

```
CREATE VIEW GETNBLOCCHIFILE (FILE_ID, NBLOCCHI)
SELECT FILE_ID, COUNT(*) FROM BLOCCI
GROUP BY FILE_ID;
```

```
CREATE VIEW GETBLOCCHIATTESA (FILE_ID, NBLOCCHIATT)
SELECT FILE_ID, COUNT(*) FROM CODA
GROUP BY FILE_ID;
```

```
CREATE VIEW V2 (FILE_ID)
SELECT FILE_ID FROM
GETNBLOCCHIFILE NATURAL JOIN GETBLOCCHIATTESA
WHERE NBLOCCHIATT = NBLOCCHI;
```

---

La data di possesso del file coincide con la data maggiore del possesso dei suoi blocchi

```
CREATE VIEW GETMAXDATAB( USR_ID, FILE_ID, DATAMAX )
SELECT USR_ID, FILE_ID, MAX(DATA) FROM POSSIEDE_BLOCCHI
GROUP BY USR_ID, FILE_ID
```

```
CREATE ASSERTION A1
```

```
CHECK NOT EXIST
```

```
SELECT * FROM
```

```
GETMAXDATAB NATURAL JOIN POSSIEDE_FILE PF
WHERE PF <> GETMAXDATAB
```

*FILM(CodFilm, Titolo, Regista, Anno, Durata)  
 SALA(CodSala, Nome, NumPosti)  
 PROIEZF(CodProiez, CodSala, CodFilm, Data, Ora, MaxSpot, Maxtrailer)  
 SPOT(CodS, Titolo, Prodotto, Durata)  
 PROIEZST(CodST, CodProiez, Ordine, Durata)  
 TRAILER(CodT, CodFilm, Durata)*

**Esercizio 04 (Punti 10)** Scrivere una procedura che prende in ingresso il codice di uno spot ed un numero intero k. La procedura deve cercare k spazi di proiezione per lo spot tra tutte le proiezioni ancora da effettuare (la loro data è successiva alla data corrente SYSDATE). Una proiezione può ospitare uno spot solo se il tempo complessivo già occupato da spot già associati alla proiezione sommato alla durata dello spot da associare non eccede l'attributo MaxSpot. Nell'associazione degli spot alle proiezione vanno privilegiate le proiezioni che si effettuerano prima rispetto alla data corrente. L'associazione degli spot va inserita nella tabella PROIEZST.

```
CREATE PROCEDURE P1 ( CS IN SPOT.CODS%TYPE, K IN INT )
```

```
CURSOR C1
```

```
IS
```

```
SELECT PF.CODPROIEZ FROM PROIEZF PF
WHERE PF.DATA > SYSDATE() AND
PF.MAXSPOT >= DURATASPORT + (SELECT SUM(DURATA) FROM PROIEZST WHERE CODPROIEZ =
PF.CODPROIEZ)
ORDER BY PF.DATA ASC;
```

```
DURATASPORT SPOT.DURAT%TYPE;
MAXORDINE PROIEZST.ORDINE%TYPE;
```

```
CP PROIEZF.CODPROIEZ%TYPE;
COUNT INT := 0;
```

```
BEGIN
```

```
SELECT DURATA INTO DURATASPORT FROM SPOT WHERE CODS = CS
```

```
OPEN C1
```

```
EXIT WHEN C1%FOUND OR COUNT > K
```

```
LOOP
```

```
FETCH C1 INTO CP
```

```
SELECT MAX(ORDINE) INTO MAXORDINE FROM PROIEZST
WHERE CODPROIEZ := CP;
```

```
INSERT INTO PROIEZST VALUES ( CS, CP, MAXORDINE, DURATA);
```

```
COUNT := COUNT + 1;
```

```
END LOOP
```

```
END
```

*ALBERI(CodA, Radice)*

*NODI(CodN, Peso)*

*COMPNODI(CodA, CodN)*

*COMPARCHI(CodA, Padre, Figlio, Peso)*

---

**Esercizio 04 (9 punti)** Si scriva una funzione PLSQL che riceve in ingresso il codice di un albero ed il codice di un nodo. La funzione restituisce la lunghezza del cammino massimo nel sottoalbero radicato nel nodo dato e il numero di foglie del sottoalbero radicato nel nodo dato. Si svolga l'esercizio con programmazione iterativa (non ricorsiva). Ci si avvalga di una struttura temporanea TMP(codA, CodN, altezza) che si suppone sia già definita che deve essere istanziata inserendo iterativamente ogni nodo del sottoalbero e l'altezza del nodo dal nodo radice del sottoalbero.

```
CREATE FUNCTION F1 ( CA ALBERI.CODA%TYPE, CN NODI.CODN%TYPE )
```

```
ALT INT := 1;
```

```
BEGIN
```

```
DELETE FROM TMP;
```

```
INSERT INTO TMP(CA, CN, 0)
```

```
LOOP
```

```
INSER INTO TMP
```

```
(SELECT CA, CA.FIGLIO, ALT FROM COMPARCHI CA JOIN TMP T ON T.CODN = C.PADRE
```

```
END LOOP
```

```
END
```

*STUDENTE(Matricola, Nome, Cognome)*

*PIANI(CodPiano, Matricola, Data)*

- *COMPOSIZIONE(CodPiano, CodI, Anno, Voto, Lode)*

- *INSEGNAM(CodI, Titolo, CFU, Tipo)*

- *ANNI(CodEsame, Anno)*

- *PROPED(CodEsame, CodEsameProp)*

- *VERBALEESAME(CodV, Data, CodI, CodDocente)*

*ESAMIVERBALE(CodV, Matricola, Voto, Lode)*

---

*ANNI(CodI,Anno), PROPED(CodI,CodIProped)*

**Esercizio 03 (punti 8)** Si scriva un trigger che viene attivato quando viene inserito un nuovo verbale di esame. L'effetto del trigger è il seguente. Per ogni esame presente nel verbale

- si controlla che l'insegnamento sia presente nel piano di studi dello studente;
- si controlla che tutti gli insegnamenti propedeutici siano stati sostenuti (presenti con voto nel piano di studi) dallo studente;
- si registra il voto e la data nel piano di studi (se i controlli hanno effetto positivo)

**Esercizio 04 (Punti 9)** Si scriva una procedura in PL\SQL che riceve in ingresso la matricola di uno studente e un nuovo codice di piano di studi (non ancora presente nella base di dati). La procedura crea un piano di studi avente come codice quello passato per parametro. Il piano di studi include tutti gli insegnamenti obbligatori e tutti gli insegnamenti di default. Ciascun esame è collocato nel primo anno possibile.

```

CREATE TRIGGER T1
BEFORE INSERT ON VERBALEESAME
FOR EACH ROW

CP PIANI.CODPIANO%TYPE;
V ESAMIVERBALE.VOTO%TYPE;
L ESAMIVERBALE.LODE%TYPE;

BEGIN

IF NOT EXIST (SELECT * FROM INSEGNAM WHERE CODI = NEW.CODI)
THEN
RAISE EXCEPTION
ELSE
IF
(SELECT COUNT(*) FROM PROPED WHERE CODESAME = NEW.CODI)
-
(SELECT COUNT(*) FROM ESAMIVERBALE E NATURAL JOIN PIANI P NATURAL JOIN COMPOSIZIONE C
WHERE E.CODV = NEW.CODV AND C.VOTO IS NOT NULL
AND C.CODI IN (SELECT CODESAMEPROP FROM PROPED WHERE CODESAME = NEW.CODI))
<> 0
THEN
RAISE EXCEPTION
ELSE

(SELECT P.CODPIANO INTO CP FROM PIANO P NATURAL JOIN ESAMIVERBALE E WHERE E.CODV = NEW.CODV)

(SELECT E.VOTO INTO V ESAMIVERBALE E WHERE E.CODV = NEW.CODV)

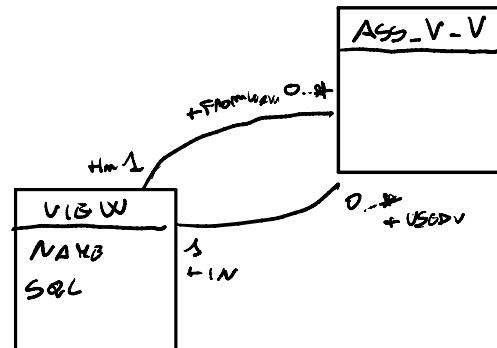
(SELECT E.LODE INTO L ESAMIVERBALE E WHERE E.CODV = NEW.CODV)

UPDATE COMPOSIZIONE
SET VOTO = V AND LODE = L
WHERE CODPIANO = CP AND CODI = NEW.CODI;

END IF

END

```



```

CREATE PROCEDURE P1
( M STUDENTE.MATRICOLA%TYPE, CP PIANI.CODPIANO%TYPE )

CURSOR C1
IS
SELECT CODI FROM INSEGNAM WHERE TIPO IN ('DEFAULT', 'OBBLIGATORIO')

MINAN ANNI.ANNO%TYPE;

CI INSEGNAM.CODI%TYPE;

BEGIN
INSERT INTO PIANI VALUES (CP, M, SYSDATE());

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO CI

```

```

SELECT MIN(ANNO) INTO MINAN FROM ANNI WHERE CODESAME = CI;
INSERT INTO COMPOSIZIONE VALUES (CP, CI, MINAN, NULL, NULL);
END LOOP

END

```

*MAGAZZINO(CodA, Descrizione, Collocazione, Quantita, SogliaMin)*  
*ORDINE(CodO, DataOrdine, CodFornitore, DataArrivo)*  
*COMPORDINE(CodA, CodO, Quantita)*  
*FORNITORE(PI, RagSociale, Via, Citta)*  
*CATALOGO(CodA, PI, Prezzo)*

---

#### EXECUTE IMMEDIATE comando;

**Esercizio 04** (Punti 8) Si scriva una procedura che riceve in ingresso una stringa di codici di articolo separati dal simbolo separatore # e un intero k. La procedura controlla per ogni fornitore qual è il prezzo complessivo per un ordine di k esemplari di ogni articolo nella lista e sceglie il fornitore che ha il prezzo più vantaggioso. La procedura dunque inserisce nel database un ordine al fornitore scelto per tutti gli articoli. Supponendo che il codice dell'ordine sia un intero, il codice dell'ordine da inserire è l'intero successivo rispetto a quello già usato.

**Esercizio 05** (Punti 6) Utilizzando SQL DINAMICO Si scriva una procedura che riceve in ingresso il nome di una tabella e una stringa di nomi di attributi separati dal carattere separatore #. La funzione ha l'effetto di aggiungere alla tabella passata per parametro un (unico) vincolo di unicità che coinvolge tutti gli attributi passati nella stringa.

```
CREATE PROCEDURE P1 ( STRINGA VARCHAR2(1000), K INT )
```

*K ARTICOLI IN ORDINE  
X OGNI ARTICOLO*

```

P VARCHAR2(100) := "";
S VARCHAR2(1000) := STRINGA;
MAXORDINE ORDINE.CODO%TYPE;
FORN FORNITORE.PI%TYPE;
PREZ INT := 0;
MAXPREZ INT := 0;
MAXFORN FORNITORE.PI%TYPE;

```

```

CURSOR C1
IS
SELECT PI FROM FORTNITORE;

```

```

CURSOR C2
IS
SELECT CODA FROM TMP;
COA MAGAZZINO.CODA%TYPE;

```

```

BEGIN

-- CARICO IN UNA TABELLA TEMPORANEA TUTTI I CODICI DI ARTICOLO tmp(codA)
WHILE LENGTH(S) >> 0
LOOP
IF INSTR(S, 1, '#') = 0 THEN P := S; S := "";
ELSE

```

```

P := SUBSTR(S, 1, INSTR(S, 1, '#') - 1);
S := SUBSTR(S, INSTR(S, 1, '#') + 1);

INSERT INTO TMP VALUES(P);

END LOOP

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO FORN

SELECT SUM(CA.PREZZO) INTO PREZ
FROM ORDINE O JOIN COMPORDINE CO ON CO.CODO = O.CODO
JOIN CATALOGO CA ON CA.CODA = CO.CODA AND CA.PI = O.CODFORNITORE
WHERE O.CODFORNITORE = FORN AND CO.QUANTITA = K
AND CO.CODA IN (SELECT CODA FROM TMP);

IF PREZ > MAXPREZ
THEN
MAXPREZ := PREZ;
MAXFORN := FORN;
END IF

END LOOP

SELECT MAX(CODO) INTO MAXORDINE FROM ORDINE;

IF MAXFORN IS NOT NULL
THEN
INSERT INTO ORDINE VALUES
(MAXORDINE, SYSDATE(), MAXFORN, NULL);

OPEN C2
WHILE C2%FOUND
LOOP
FETCH C2 INTO COA

INSERT INTO COMPORDINE VALUES
( COA, MAXORDINE, K );

END LOOP

END IF

END

```

**Esercizio 05 (Punti 6)** Utilizzando SQL DINAMICO Si scriva una procedura che riceve in ingresso il nome di una tabella e una stringa di nomi di attributi separati dal carattere separatore #. La funzione ha l'effetto di aggiungere alla tabella passata per parametro un (unico) vincolo di unicità che coinvolge tutti gli attributi passati nella stringa.

```

CREATE PROCEDURE P1
( TABELLA USER_TABLE.TABLE_NAME%TYPE, STRINGA VARCHAR2(1000) )

COMANDO VARCHAR2(1000) := 'ALTER TABLE ' || TABELLA || 'ADD CONSTRAINT U1 UNIQUE (';
S VARCHAR2(1000) := STRINGA;

```

```

P VARCHAR2(100);

BEGIN

WHILE LENGTH(S) <> 0
LOOP
IF INSTR(S, 1, '#') = 0 THEN P:= S; S := '';
ELSE
P := SUBSTR(S, 1, INSTR(S, 1, '#') + 1);
S := SUBSTR(S, INSTR(S, 1, '#') - 1);

COMANDO := COMANDO || P || ';';

END IF
END LOOP

COMANDO := SUBSTR(COMANDO, LENGTH(COMANDO) - 1);
COMANDO := COMANDO || ');';

EXECUTE IMMEDIATE COMANDO;

END

```

*PERSONA(CF, Nome, Cognome, DataN, DataM)  
RESIDENZA(CF, DataI, Via, Num, Citta, DataF)  
GENITORI(CF, CFMadre, CFPadre)  
FAMIGLIA(codF, CFCapo)  
COMPFAMIGLIA(codF, codMembro)*

**Esercizio 03 (9 punti)** Si scriva una procedura PLSQL che riceve in ingresso il CF di una persona e restituisce una stringa di caratteri contenente il nome e cognome di tutti gli antenati in linea maschile disponibili nel database (padre, nonno paterno, padre del nonno paterno etc.).

```

CREATE PROCEDURE P1
( CODF IN PERSONA.CF%TYPE, STRINGA OUT VARCHAR2(1000) )

F PERSONA.CF%TYPE := CODF;

CURSOR C1
IS
SELECT CFPADRE FROM GENITORI WHERE CF = F;

RISULTATO VARCHAR2(1000);

BEGIN

OPEN C1
WHILE C1%FOUND
LOOP
FETCH C1 INTO F

SELECT (NOME || ' ' || COGNOME) INTO RISULTATO FROM PERSONA WHERE CF = F;

STRINGA := STRINGA || RISULTATO || ';';

END LOOP

END

```

**Esercizio 04 (7 punti)** Si scriva un trigger attivato quando viene fissata la data di morte di una persona. L'effetto è il seguente:

- viene chiusa in quella data la residenza corrente della persona;
- se la persona fa parte di una famiglia viene tolta dal nucleo familiare e se in particolare la persona è il capofamiglia viene aggiornato il capofamiglia con la persona di età maggiore appartenente alla famiglia.

```
CREATE TRIGGER T1
AFTER UPDATE OF DATAMON PERSONA
WHEN (OLD.DATAM IS NULL AND NEW.DATAM IS NOT NULL)

CAPO PERSONA.CF%TYPE;
CODFAMIGLIA FAMIGLIA.CODF%TYPE;

BEGIN

UPDATE RESIDENZA SET DATAF = NEW.DATAM WHERE CF = OLD.CF AND DATAF IS NULL;

IF EXIST SELECT * FROM COMPFAMIGLIA WHERE CODMEMRBO = OLD.CF
THEN

IF EXIST SELECT * FROM FAMIGLIA WHERE CFCAPO = OLD.CF
THEN

SELECT CODF INTO CODFAMIGLIA FROM FAMIGLIA WHERE CFCAPO = OLD.CF;

SELECT P.CF INTO CAPO FROM
(SELECT P.CF, MIN(P.DATAN) FROM COMPFAMIGLIA CF NATURAL JOIN PERSONA P
WHERE CF.CODMEMBRO <> OLD.CF
GROUP BY P.CF);

UPDATE FAMIGLIA SET CFCAPO = CAPO WHERE CODF = CODFAMIGLIA;

END IF

DELETE FROM COMPFAMIGLIA WHERE CODMEMBRO = OLD.CF

END IF

END
```