

*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.

$\overline{\Pi} (\text{STUDENTE} \bowtie \text{PIANI} \bowtie \text{COMPOSIZIONE} \bowtie (\text{ESAMI} \setminus (\text{ESAMI} \bowtie \text{PROPED})))$
 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 virgole (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 virgole).

```

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 || ',';

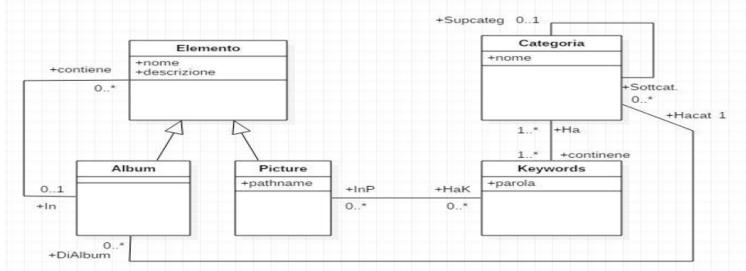
```

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)

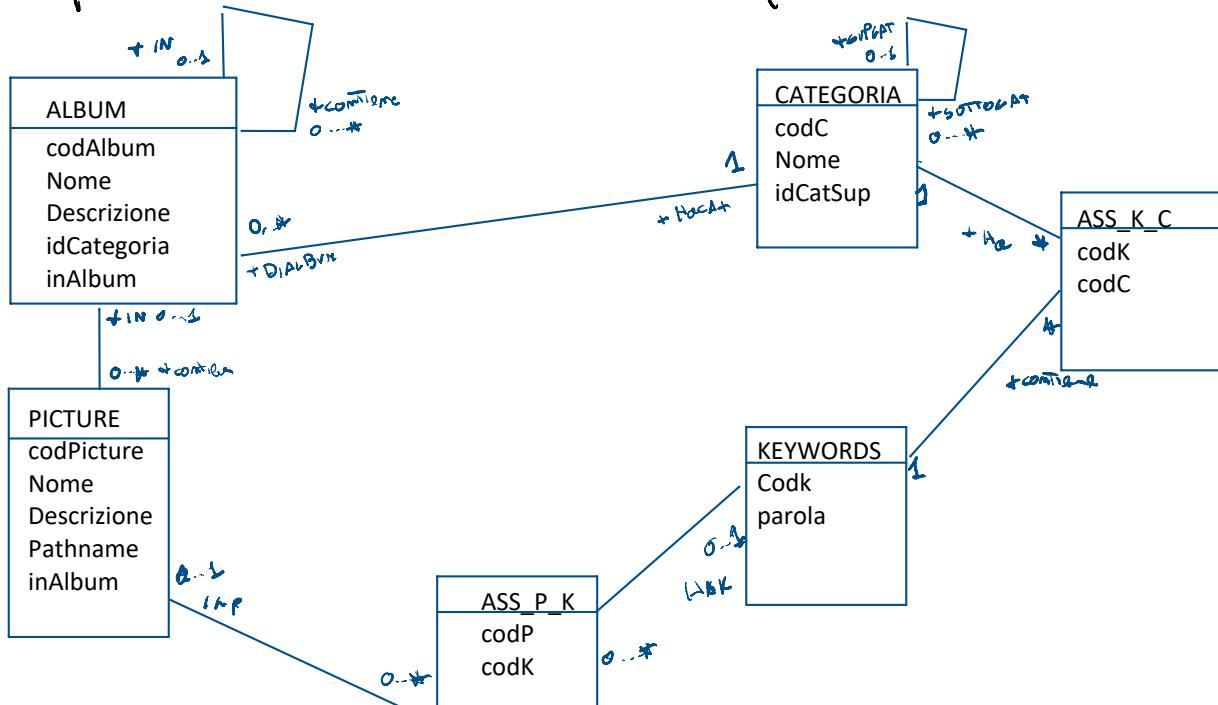
ASS_P_K (CODP, CODK)

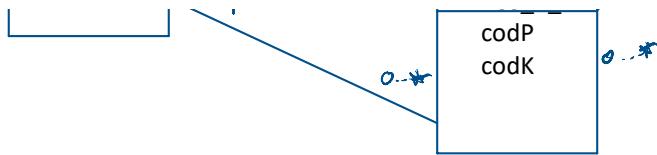
KEYWORDS (CODK, PAROLA)

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 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)
);

```

```

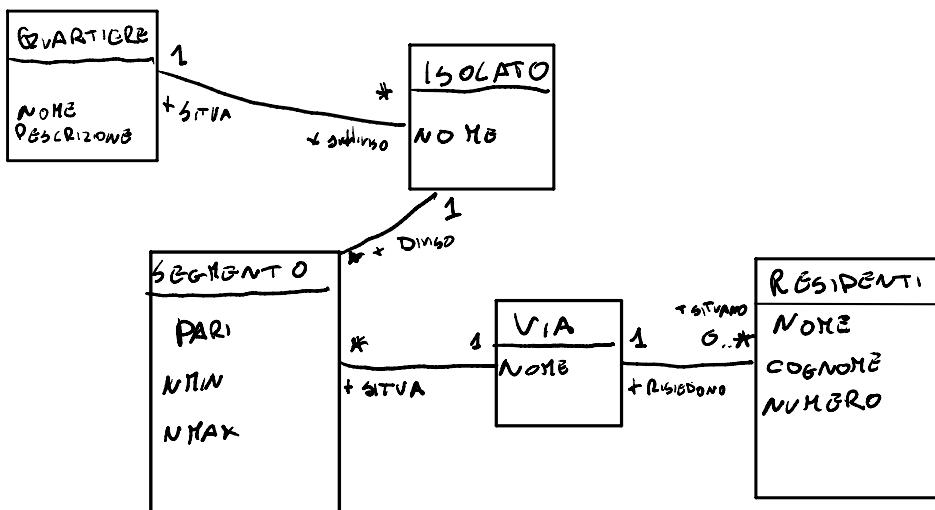
CREATE TABLE ASS_C_K (
CODC INT NOT NULL,
CODK INT NOT NULL,
CONSTRAINT FK_C FOREIGN KEY (CODC) REFERENCES CATEGORIA(CODC),
CONSTRAINT FK_K FOREIGN KEY (CODK) REFERENCES KEYWORDS(CODK)
);

```

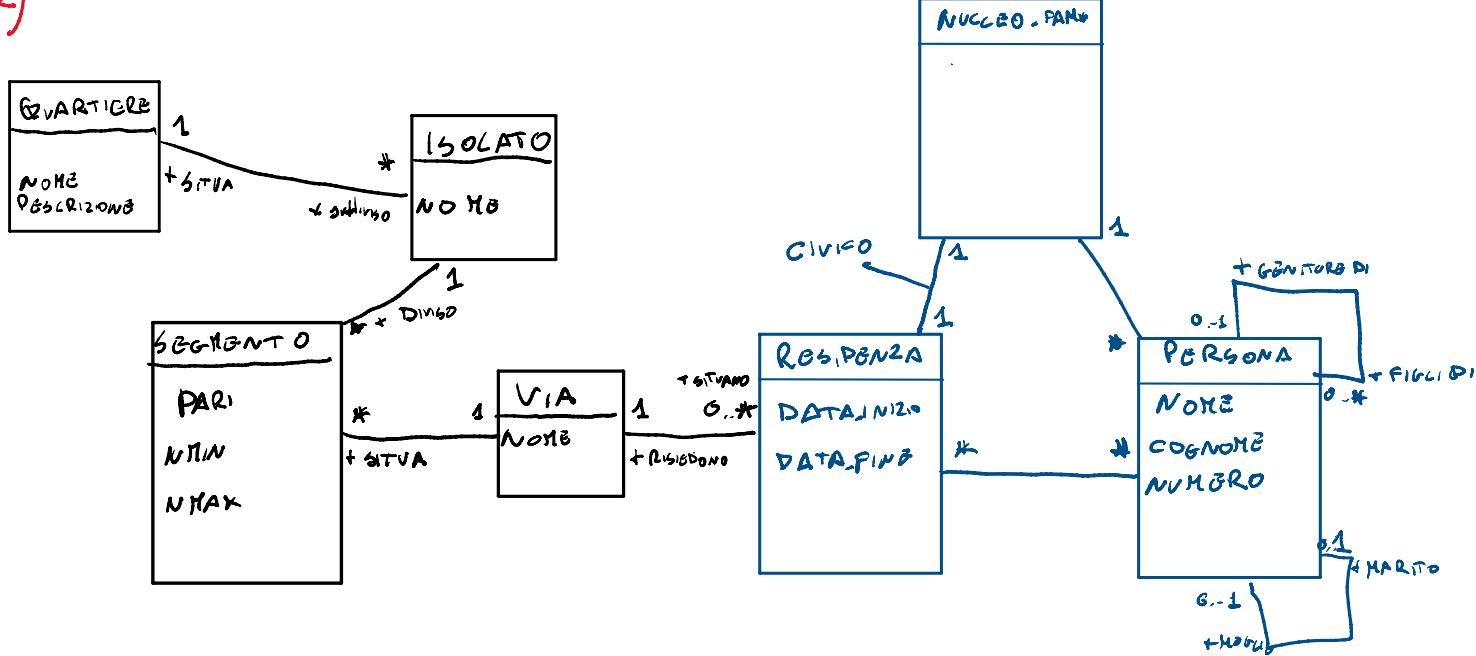
`QUARTIERE(CodQ, Nome, Descrizione), ISOLATO(CodI, CodQ, Nome)`
`VIA(CodV, Nome), SEGMENTO(CodS, CodV, CodI, Pari, NMin, NMax)`
`RESIDENTI(CF, Nome, Cognome, CodV, Numero).`

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)

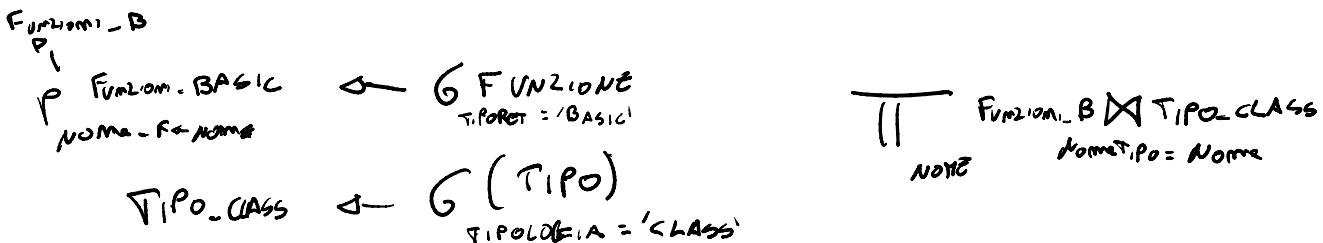
VINCOLO INTERRELAZIONALI

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



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

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

$$TP \leftarrow \emptyset \text{ (TIPO)} \\ \text{Tipologia} = 'CLASS' \\ ATT \leftarrow \emptyset \text{ (ATTRIBUTO)} \\ \text{TipoAttR} = 'BASIC'$$

$$\overline{\prod_{NOME TIPO} TP \bowtie_{NOME = NOME TIPO} ATT}$$

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
```

$\bar{Foto}(\underline{\text{CodF}}, \text{NomeFile}, \text{Proprietario}, \text{Dimensione}, \text{Descrizione}, \text{Tempo})$
 $\bar{TAG}(\text{CodF}, \text{parola})$
 $\bar{Key}(\text{parola})$
 $\bar{Classificazione}(\text{Parola1}, \text{Parola2}, \text{Relazione})$

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.

$$\overline{\Pi}_{\text{CODF}} \left(G \left(\begin{array}{l} \text{Proprietario} \leq \text{MAX}(\text{Tempo})(\text{Foto}) \\ \text{Proprietario} = \text{Proprietario} \text{ AND } \text{Tempo} = \text{MAX}(\text{Tempo}) \end{array} \right) \right)$$

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(DSCR, 1, INSTR(DSCR, 1, ' '));
DSCR := SUBSTR(DSCR, INSTR(DSCR, 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 || ',';
```

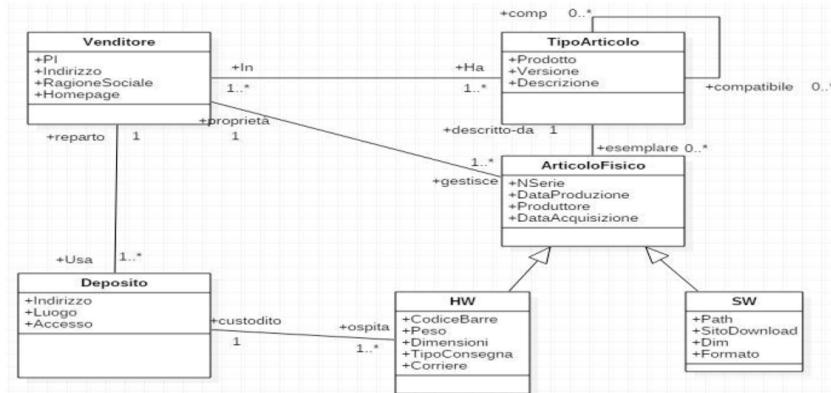
I := I + 1;

END LOOP

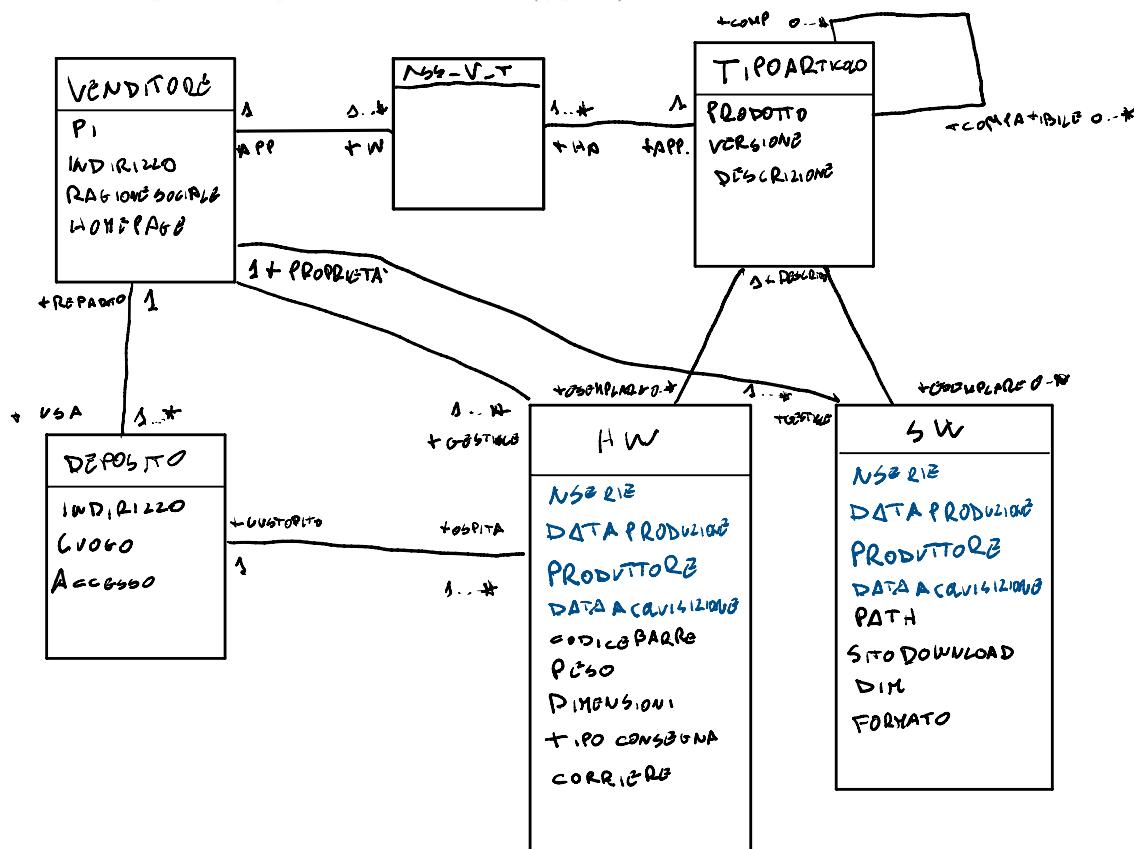
RISULTATO := SUBSTR(RISULTATO, 1, LENGTH(RISULTATO) - 2);

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) 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)
);

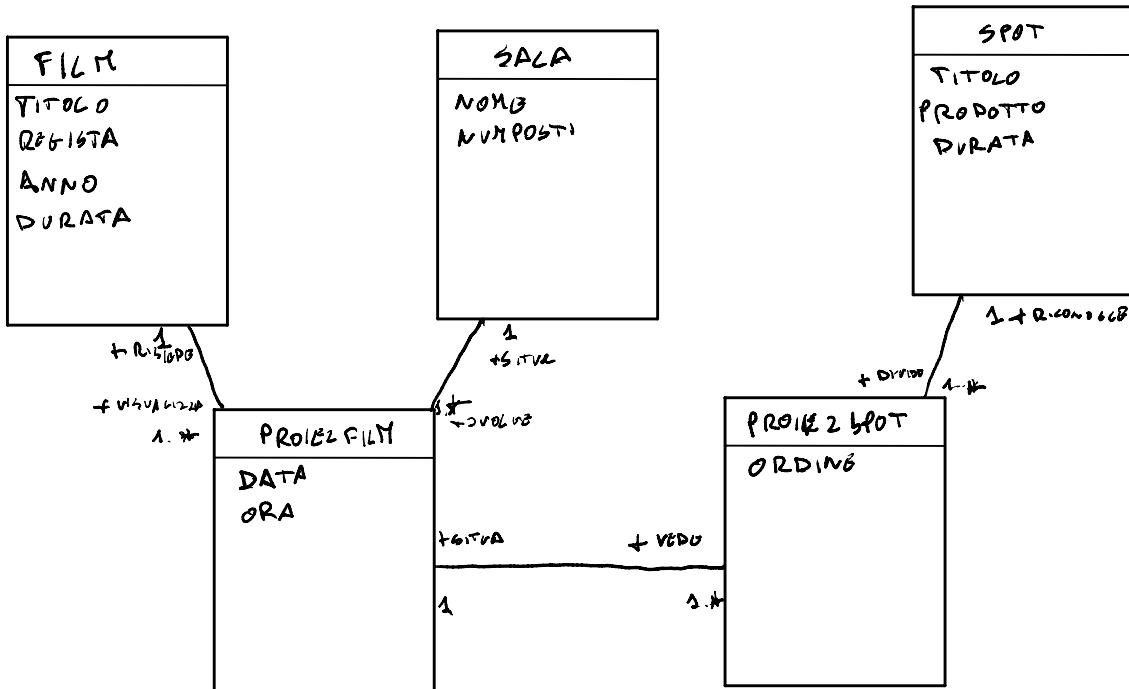
```

```

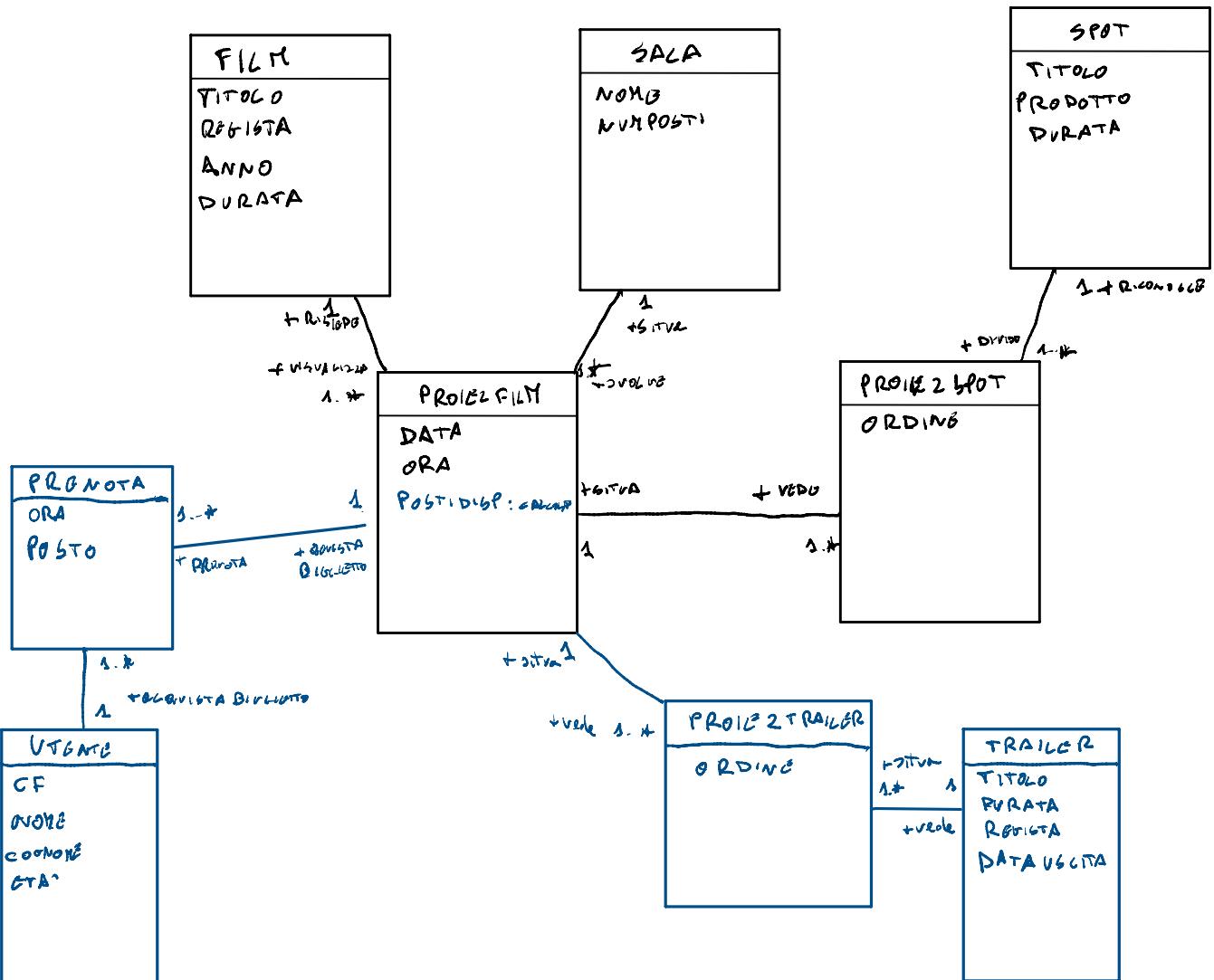
CREATE ASS_V_TA (
CODV INT NOT NULL,
PRODOTTOTA VARCHAR(300) NOT NULL,
VERSIONE INT NOT NULL,
CONSTRAINT FK_ASS_V_TA_TA FOREIGN KEY (PRODOTTOTA, VERSIONETA) REFERENCES TIPOARTICOLO (PRODOTTO, VERSIONE),
CONSTRAINT FK_ASS_V_TA_V FOREIGN KEY (CODV) REFERENCES VENDITORE (CODV)
);

```

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)
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.

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

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(Doi, 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 $\rightarrow P(Doi \text{ } \text{SCRIVE} \text{ } \text{Orcid} \leftrightarrow \text{Doi})$

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

NO $\rightarrow \left(\bigcap_{\text{Orcids} \leftrightarrow \text{Doi}} (\text{SCRIVE} \Delta \text{SCRIVE}) \Delta \text{AUTORE} \right)$

$$\text{Orcids} = \text{Orcid}$$

$\overline{\bigcap}_{\text{Orcid}, \text{Nome}, \text{Cognome}} (\text{AUTORE} \setminus \text{NO})$

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:

1. Due autori di uno stesso articolo devono avere valori di ordine diverso;
2. 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;
3. 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

LIBRO(ISBN, Titolo, Editore, Anno)
ESEMPLARE(ISBN, CodiceBarre, Collocazione, Prestito, Consultazione)
UTENTE(CF, CodProfilo, Nome, Cognome, DataN)
PROFILO(CodProfilo, MaxDurata, MaxPrestito)
PRESTITI(CodPrestito, CodiceBarre, Utente, Data, Scadenza, Restituzione, Sollecito)
PRENOTAZIONE(CodPrenotazione, ISBN, Utente, 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).

• $L \leftarrow \sigma_{\text{Anno} = 2018} G(\text{LIBRO})$ $\epsilon P \leftarrow ESEMPLARE \Delta PRESTITI$
 $\text{PRESTITO} = \text{CODPRESTITO}$

3) $\prod_{\text{ISBN}} (\text{ISBN} \sum \text{MAX}(n_PRESTITI)) (R)$ 2) $R \leftarrow \rho_{\text{ISBN} \leq \text{COUNT}(\text{CODPRESTITO})} (L \Delta \epsilon P)$
 $\text{ISBN}, n_PRESTITI$

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 contenete 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

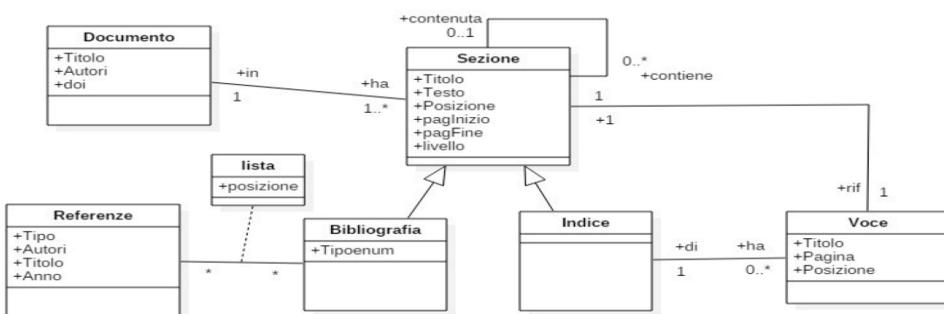
END LOOP

END LOOP

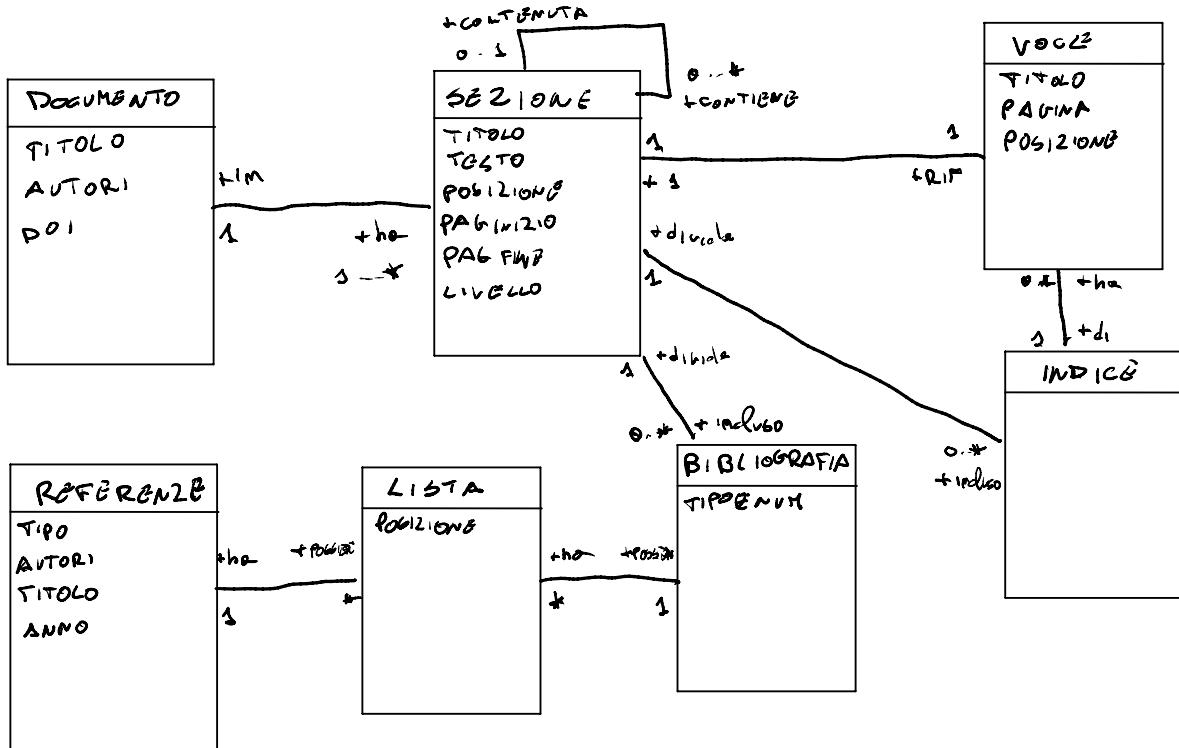
RISULTATO := SUBSTR(RISULTATO, 1, LENGTH(RISULTATO) - 1);

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, N^o Rⁱghe)

ATTRIBUTI(CodTab, Nome, Tipo, NotNull)

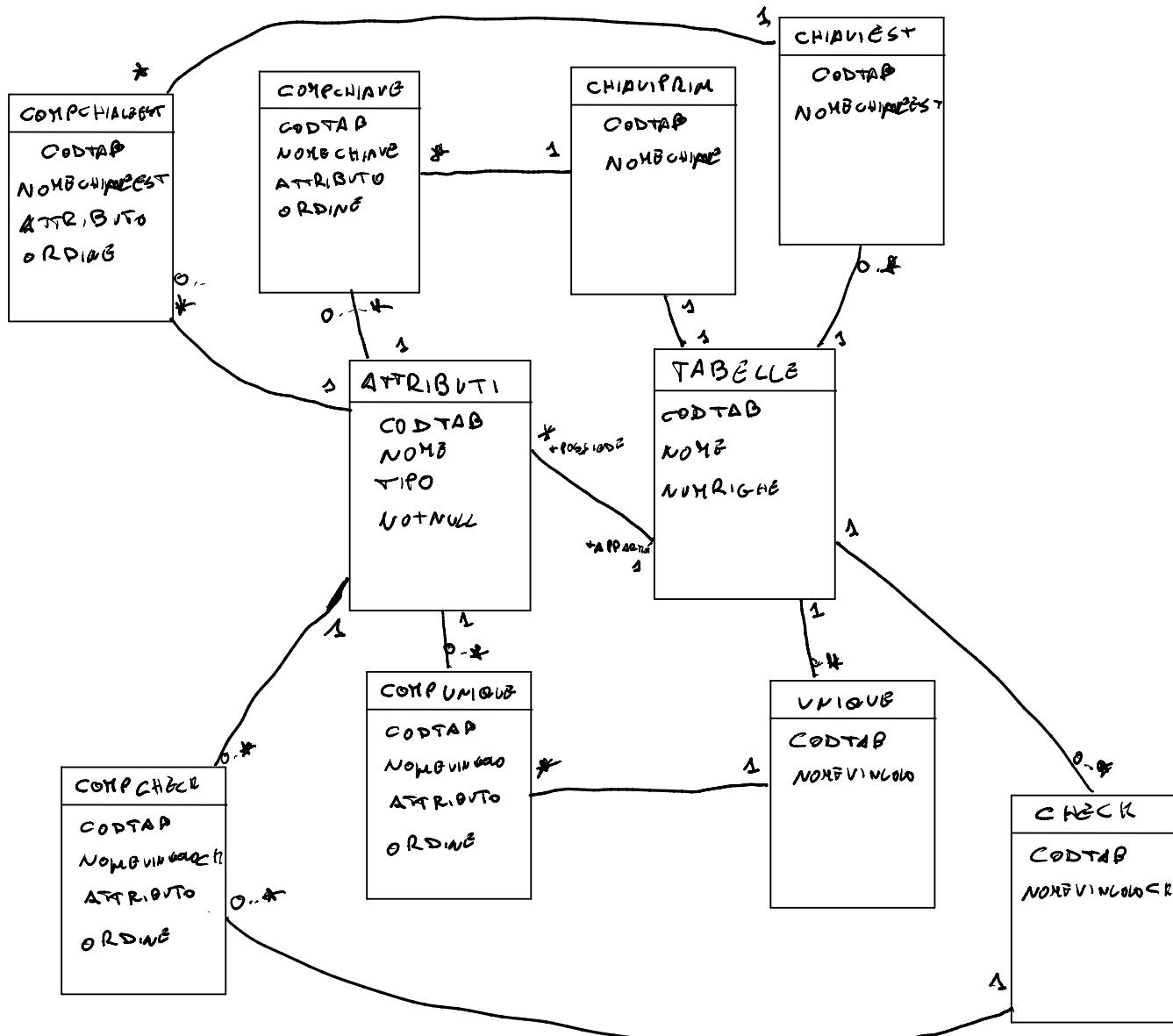
UNIQUE(CodTab, NomeVincolo)

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)