Esame

Studente(matricola,nome,cognome)

Materia(id,titolo,descrizione)

Esercizi(id,testo,soluzione, materia,numerosoluzioni)

Risolto(idesercizio,idstudente,data)

-Definire le chiavi primarie ed esterne (ITINERE) **[1 punto]**

Algebra

-Trovare gli studenti che non hanno risolto esercizi per la materia "basi di dati" **[3 punti]**

$$R1 = MATERIA \bowtie_{MATERIA.id=ESERCIZI.materia} ESERCIZI \bowtie_{ESERCIZI.id=RISOLTO.idesercizio} RISOLTO$$
$$R2 = \pi_{matricola}(STUDENTE) - \delta_{idstudente \to matricola}\left(\pi_{idstudente}\left(\sigma_{titolo="BASI\ DI\ DATI"}(R1)\right)\right)$$
$$STUDENTE \bowtie R1$$

-Trovare le materie per cui sono stati risolti tutti gli esercizi (ITINERE) **[3 punti]**

$$MATERIA - \delta_{materia \to id}\left(\pi_{materia}\left(\sigma_{numerosoluzioni=0}(ESERCIZI)\right)\right)$$

SQL

-Per ogni materia contare il numero di esercizi disponibili e quelli risolti **[4 punti]**

```
SELECT t1.materia,numeroesercizi,numeroesercizirisolti
FROM (
    SELECT count(*) numeroesercizi, materia
    FROM ESERCIZI
    GROUP BY MATERIA
  ) AS t1,(
    SELECT count(*) numeroesercizirisolti, materia
    FROM ESERCIZI
    WHERE numerosoluzioni>0
    GROUP BY MATERIA
  ) AS t2
WHERE t1.materia=t2.materia
```

-Trovare gli esercizi che contengono la parola "SQL" che non sono stati risolti (ITINERE) **[4 punti]**

```
SELECT *
FROM Esercizi
WHERE testo LIKE '%SQL%' AND numerosoluzioni = 0
```

-Implementare un trigger in SQL che ogni qualvolta viene inserita una soluzione per un esercizio nella relazione Risolto aggiorna il campo numerosoluzioni della tabella esercizi (ITINERE) **[4 punti]**

```
CREATE TRIGGER T1
AFTER INSERT ON RISOLTO
FOR EACH ROW
UPDATE Esercizi
SET numerosoluzioni = numerosoluzioni + 1
WHERE id = NEW.idesercizio
```

ER **[5 punti]**

Si supponga di avere le seguenti operazioni:

q1 - Inserisci soluzione nella relazione Risolto 100 volte al giorno

q2 - Dai il numero di soluzioni proposte per un esercizio

Valutare se conviene mantenere l'attributo "numerosoluzioni"

Con ridondanza	Senza ridondanza
Q1 1 Scrittura Risolto 1 Scrittura Esercizi TOTALE = 2 S = 4 L -> 400 Q2 1 Lettura Esercizi TOTALE = 1 * numero operazioni COSTO TOTALE = 400 + 1*x	Q1 1 Scrittura Risolto TOTALE = 1 S = 2 L -> 100 Q2 1 Lettura Esercizio 10 Letture Risolto TOTALE = 11 * numero operazioni COSTO TOTALE = 200 + 11 * x

Se 200 + 11*x > 400 + x converrà mantenere la ridondanza
Ovvero 10*x -200 > 0
Quindi se x > 20 converrà mantenere la ridondanza

Normalizzazione **[6 punti]**

Dato lo schema Impiegato (Nome, Livello, Stipendio) con le seguenti dipendenze funzionali
F={Nome→Livello, Stipendio, Livello→ Stipendio}

-Dire se lo schema è in 3NF o BCNF

Chiave Nome

Non è in 3NF ne BCNF

- Se non è in BCNF decomporlo in modo tale che rispetti la BCNF

BCNF: R1(Livello,Stipendio) R2(Nome,Livello)

Itinere 1

Persona (cf, nome, cognome)

Libro (id, titolo, descrizione, autore, numerodilettori, datauscita, saga, volume)

Recensione (id, ~~libro~~, testo, data, ~~persona~~)

Letto(~~libro, persona~~, data)

-Trovare le chiavi primarie ed esterne dello schema **[1 punto]**

Algebra

-Trovare i libri che hanno almeno 2 recensioni ma che non sono stati letti **[3 punti]**

$$R1 = R2 = Recensione$$
$$\pi_{R1.libro}(R1 \bowtie_{R1.libro=R2.libro \ \wedge R1.id>R2.id} R2) - \pi_{libro}(Letto)$$

-Trovare le persone che hanno letto tutti i libri di "JK Rowling" **[3 punti]**

$$LibriJKR \coloneqq \delta_{id \to libro}\left(\pi_{id}\left(\sigma_{autore="JK\ Rowling"}(Libro)\right)\right)$$
$$\pi_{libro,persona}(Letto)/LibriJKR$$

SQL

-Implementare un vincolo che non consenta di inserire in Letto un libro di una saga se non nel corretto ordine cronologico (V1, V2,…) **[4 punti]**

```
CREATE TRIGGER T1 AFTER INSERT ON Letto
FOR EACH ROW
DECLARE X,Y,Z INT
BEGIN
  SELECT volume INTO X, saga INTO Y FROM libro WHERE id = NEW.libro
  IF (X IS NOT NULL AND X > 1) THEN
    SELECT id INTO Z FROM LIBRO WHERE saga = Y and volume = X-1;
    IF(NOT EXISTS (SELECT * FROM letto where libro = Z and persona = new.persona) THEN
      DELETE FROM letto WHERE data = new.data
    END IF
  END IF
END
```

-Per ogni autore contare il numero di libri e il numero di lettori distinti e il numero di recensioni avute da persone distinte. **[4 punti]**

```
CREATE VIEW V1 AS SELECT COUNT(DISTINCT persone) AS lettori,autore FROM Letto, Libro
WHERE libro=ID GROUP BY autore

CREATE VIEW V2 AS SELECT COUNT(DISTINCT persone) AS recesioni,autore FROM recensione,
libro WHERE libro=ID GROUP BY autore

CREATE VEW V3 AS SELECT COUNT(*) libri, autore FROM libro group by autore

SELECT libri, recesioni, lettori FROM V1 NATURAL JOIN V2 NATURAL JOIN V3
```

Itinere 2

AutoPosseduta(<u>targa</u>,idauto ,costorifornimenti,dataimmatricolazione)
Auto(<u>id</u>,marca, alimentazione,cilindrata)
Rifornimento(<u>targa,data</u>,prezzolitro,litri)
Manutanezione(<u>targa,data</u>,descrizione,costo)

-Identificare le chiavi primarie ed esterne **[1 punto]**

Algebra

-Trovare le auto, indicando marca e modello, che non sono state vendute **[2 punti]**

$$Auto \bowtie \left(\pi_{id}(Auto) - \delta_{idauto \to id}\left(\pi_{idauto}(AutoPosseduta)\right)\right)$$

-Trovare per ogni marca trovare le auto con la cilindrata maggiore **[3 punti]**

$$A1 = A2 = Auto$$
$$R1 = \pi_{A1.cilindrata,A1.marca}(A1 \bowtie_{A1.cilindrata<A2.cilindrata \wedge A1.marca=A2.marca} A2)$$
$$\pi_{cilindrata,marca}(Auto) - R1$$

SQL

-Trovare le marche di automobili che hanno venduto tutti i modelli **[3 punti]**

```
SELECT DISTINCT marca
FROM auto a1
WHERE NOT EXISTS (SELECT * FROM Auto a2 WHERE a1.marca=a2.marca AND
NOT EXISTS (SELECT * FROM AutoPosseduta WHERE a2.id=idauto))
```

-Per ogni autoposseduta mostrare quelle per il quale il numero di manutenzioni effettuate è maggiore di quello medio. Per queste visualizzare pure il costo totale della manutenzione **[4 punti]**

```
SELECT COUNT(*) AS nmat,SUM(costo), targa FROM manutenzione GROUP BY targa
HAVING nmat >= (SELECT AVG(NumManut) FROM (SELECT COUNT(*) AS NumManut FROM manutenzione
GROUP BY targa))
```

-Implementare un trigger che ogni qualvolta viene inserito un rifornimento in Rifornimento aggiorna il costo complesivo in AutoPosseduta **[2 punti]**

```
CREATE TRIGGER t1 AFTER INSERT ON Rifornimento
FOR EACH ROW
UPDATE AutoPosseduata
SET costorifornimenti = costorifornimenti + NEW.costo
WHERE targa = NEW.targa
```

Itinere 3

Libro(id,titolo,descrizione,autore ,datauscita,sequeldi ,genere)

CopiaLibro(collocazione,idlibro)

Persona(id,nome,cognome,prestiti)

Presitito(libro,persona,dataprestito,datarestituzione,restitutito)

-Identificare le chiavi primarie ed esterne dello schema **[1 punto]**

-Trovare le persone che non hanno mai chiesto libri di “Stephen King” ma hanno chiesto almeno un libro di “Joseph Conrad” **[3 punti]**

$$R1 = \delta_{collocazione \to libro}(CopiaLibro \bowtie_{idlibro=id} Libro)$$
$$R2 = \sigma_{autore="Stephen\ King"}(R1)$$
$$R3 = \sigma_{autore="Joseph\ Conrad"}(R1)$$
$$\left(\pi_{persona}(Prestito) - \pi_{persona}(Prestito \bowtie R2)\right) \bowtie R3$$

-Trovare le persone che hanno preso in prestito tutti i libri del genere “Fantasy” **[2 punti]**

$$R1 := \pi_{id}\left(\sigma_{genere="fantasy"}(Libro)\right)$$
$$R2 := \pi_{id,persona}(R1 \bowtie_{id=idlibro} CopiaLibro \bowtie_{libro=collocazione} Prestito)$$
$$R2 \div R1$$

SQL

-Trovare le coppie di libri che hanno almeno 2 sequel **[3 punti]**

```
WITH RECURSIVE sequel (
  SELECT id, sequeldi FROM libro
  UNION ALL
  SELECT libro.id,sequel.sequeldi
  FROM libro, sequel WHERE sequel.sequeldi = libro.id FROM sequel, libro
) SELECT id FROM sequel GROUP BY id HAVING count(*) >=2
```

-Trovare libri che hanno avuto piu’ prestiti di quelli medi avuti da ogni libro (!) **[4 punti]**

```
SELECT count(*) AS prestiti, libro FROM prestito,copialibro
WHERE libro = collocazione GROUP BY libro
HAVING prestiti >= (SELECT AVG(prestiti) FROM (SELECT count(*) AS prestiti, libro
FROM prestito, copialibro WHERE libro = collocazione GROUP BY libro))
```

-Implementare un vincolo che non consenta di inserire un nuovo prestito per una persona che ha ancora un libro prestato e non restituito **[2 punti]**

```
CREATE TRIGGER T1 AFTER INSERT ON Prestito
WHEN (1 < SELECT count(*) FROM prestito where persona=NEW.persona and restituito IS NULL)
DELETE FROM prestito WHERE libro = NEW.libro, persona=NEW.persona, data=NEW.data
```