GIUGNO 2025

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale, relativo alla selezione e alla gestione dei tutor in un'università:

STUDENTE(Matricola, Nome, Cognome, Genere, DataN, Nazionalità, CdS, Anno, CFU, Media)
POSIZIONE (IdP, *Tipo, Scuola, Progetto*, Descr, NumOre, NumPosti)
CORSO(IdC, Titolo, Ore, Modalità)
RICHIEDE(IdP^{POSIZIONE}, IdC^{CORSO})
DOMANDA(Matricola^{STUDENTE}, IdP^{POSIZIONE}, DataOra)
COLLOQUIO(Matricola^{STUDENTE}, Punteggio)
ASSEGNAZIONE(Matricola^{STUDENTE}, IdP^{POSIZIONE}, Ore, DataFirma$_O$)
COMPLETA(Matricola^{STUDENTE}, IdC^{CORSO}, DataOra)

Formulare le seguenti interrogazioni in **SQL**.

1. *(*)* Determinare matricola, nome e cognome di tutti gli studenti, con il punteggio del colloquio se disponibile, altrimenti NULL.

```
SELECT S.Matricola, S.Nome, S.Cognome, C.Punteggio
FROM STUDENTE S LEFT OUTER JOIN COLLOQUIO C ON S.Matricola = C.Matricola;
```

2. Determinare l'identificatore della posizione per cui sono richiesti più corsi

```
SELECT IdP
FROM RICHIEDE
GROUP BY IdP
HAVING COUNT(*) ≥ ALL (SELECT COUNT(*)
                       FROM RICHIEDE
                       GROUP BY IdP)
```

FILA B

Esercizio II.B - SQL

Si consideri il seguente schema relazionale relativo alla selezione e alla gestione dei tutor in un'università

STUDENTE(Matricola, Nome, Cognome, Genere, DataN, Nazionalità, CdS, Anno, CFU, Media)
POSIZIONE (IdP, *Tipo, Scuola, Progetto*, Descr, NumOre, NumPosti)
CORSO(IdC, Titolo, Ore, Modalità)
RICHIEDE(IdP^{POSIZIONE}, IdC^{CORSO})
DOMANDA(Matricola^{STUDENTE}, IdP^{POSIZIONE}, DataOra)
ASSEGNAZIONE(Matricola^{STUDENTE}, IdP^{POSIZIONE}, Ore, DataFirma$_O$)
COMPLETA(Matricola^{STUDENTE}, IdC^{CORSO}, DataOra)
REGISTRO(Matricola^{STUDENTE}, Data, NumOre, Descrizione, NumPartecipanti$_O$)

Formulare le seguenti interrogazioni in **SQL**.

1. *(*)* Per tutte le posizioni mostrare l'id e i corsi richiesti, anche se non richiede alcun corso (NULL se nessuno)

```
SELECT IdP, IdC
FROM POSIZIONE LEFT OUTER NATURAL JOIN RICHIEDE
```

2. Determinare la matricola dello studente che ha registrato il maggior numero totale di ore nel registro.

```
SELECT Matricola
FROM REGISTRO
GROUP BY Matricola
HAVING SUM(NumOre) ≥ ALL (SELECT SUM(NumOre)
                          FROM REGISTRO
                          GROUP BY Matricola)
```

FILA C

Esercizio II.B - SQL

Si consideri il seguente schema relazionale relativo alla selezione e alla gestione dei tutor in un'università

STUDENTE(Matricola, Nome, Cognome, Genere, DataN, Nazionalità, CdS, Anno, CFU, Media)
POSIZIONE (IdP, *Tipo, Scuola, Progetto*, Descr, NumOre, NumPosti)
CORSO(IdC, Titolo, Ore, Modalità)
RICHIEDE(IdP^{POSIZIONE}, IdC^{CORSO})
DOMANDA(Matricola^{STUDENTE}, IdP^{POSIZIONE}, DataOra)
ASSEGNAZIONE(Matricola^{STUDENTE}, IdP^{POSIZIONE}, Ore, DataFirma$_O$)
COMPLETA(Matricola^{STUDENTE}, IdC^{CORSO}, DataOra)
REGISTRO(Matricola^{STUDENTE}, Data, NumOre, Descrizione, NumPartecipanti$_O$)

Formulare le seguenti interrogazioni in **SQL**.

1.*(*)* Per tutte le posizioni, mostrare il suo identificativo e gli identificatori dei corsi richiesti (NULL se nessuno).

C. Per tutte le posizioni, mostrare il suo identificativo e le matricole degli studenti assegnati (NULL se nessuno).

```
SELECT P.IdP, A.Matricola
FROM POSIZIONE P LEFT OUTER JOIN ~~RICHIEDE~~ ASSEGNAZIONE R ON P.IdP = A.IdP
```

2. Determinare l'id della posizione per cui sono state assegnate il maggior numero di ore

```
SELECT IdP
FROM ASSEGNAZIONE
GROUP BY IdP
HAVING SUM(Ore) ≥ ALL (SELECT SUM(Ore)
                       FROM ASSEGNAZIONE
                       GROUP BY IdP)
```

FILA D

Esercizio II.B - SQL

Si consideri il seguente schema relazionale relativo alla selezione e alla gestione dei tutor in un'università

STUDENTE(Matricola, Nome, Cognome, Genere, DataN, Nazionalità, CdS, Anno, CFU, Media)
POSIZIONE (IdP, *Tipo, Scuola, Progetto*, Descr, NumOre, NumPosti)
CORSO(IdC, Titolo, Ore, Modalità)
RICHIEDE(IdP^{POSIZIONE}, IdC^{CORSO})
DOMANDA(Matricola^{STUDENTE}, IdP^{POSIZIONE}, DataOra)
ASSEGNAZIONE(Matricola^{STUDENTE}, IdP^{POSIZIONE}, Ore, DataFirma$_O$)
COMPLETA(Matricola^{STUDENTE}, IdC^{CORSO}, DataOra)
REGISTRO(Matricola^{STUDENTE}, Data, NumOre, Descrizione, NumPartecipanti$_O$)

Formulare le seguenti interrogazioni in **SQL**.

1. (*) Per ogni assegnazione, la matricola dello studente, l'id della posizione e le date in cui sono registrate voci di registro (NULL se non è presente alcuna registrazione).

```
SELECT Matricola, IdP, Data
FROM ASSEGNAZIONE LEFT OUTER NATURAL JOIN REGISTRO
```

2. Determinare l'id della posizione per cui sono state presentate più domande

```
SELECT IdP
FROM DOMANDA
GROUP BY IdP
HAVING COUNT(*) ≥ ALL (SELECT COUNT(*)
                       FROM DOMANDA
                       GROUP BY IdP)
```

LUGLIO 2025

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

PERSONA(CF, Nome, Indirizzo, Città, Provincia, CAP, Telefono, DataNascita)
AVVOCATO (AvvocatoID, $CF^{PERSONA}$)
SPECIALIZZAZIONE(CodSpec, NomeSpec)
HA($AvvocatoID^{AVVOCATO}$,$CodSpec^{SPECIALIZZAZIONE}$)
ISCRIZIONE($AvvocatoID^{AVVOCATO}$, NomeAlbo, Data)
TRIBUNALE(IdTrib, NomeTribunale, Città, Regione, CAP)
GIUDICE(GiudiceID, $CF^{PERSONA}$, AnniEsperienza, $IdTrib^{TRIBUNALE}$)
CAUSA($IdTrib^{TRIBUNALE}$,NumCausa, DescrizioneCausa, TipoCausa)
UDIENZA($IdTrib^{CAUSA}$,$NumCausa^{CAUSA}$, NumUdienza, Data, Ora, Stanza, $GiudiceID^{GIUDICE}$)
COINVOLGE($CF^{PERSONA}$, $NumCausa^{CAUSA}$, $IdTrib^{CAUSA}$, RuoloP)
RAPPRESENTA($AvvocatoID^{AVVOCATO}$,$CF^{PERSONA}$, $NumCausa^{CAUSA}$, $IdTrib^{CAUSA}$, RuoloA)

Formulare le seguenti interrogazioni in **SQL**.

1. Determinare le persone coinvolte in una causa per cui non è presente l'avvocato che le rappresenta in quella causa

```
SELECT DISTINCT CF
FROM COINVOLGE LEFT OUTER NATURAL JOIN RAPPRESENTA
WHERE AvvocatoID IS NULL
```

```
SELECT DISTINCT CF
FROM COINVOLGE
WHERE (CF, IdTrib, NumCausa) NOT IN (SELECT CF, IdTrib, NumCausa) FROM RAPPRESENTA)
```

Oppure

```
SELECT DISTINCT C.CF
FROM COINVOLGE C
WHERE NOT EXISTS (SELECT * FROM RAPPRESENTA
                  WHERE CF=C.CF AND IdTrib = C.IdTrib AND NumCausa=C.Causa)
```

2. Determinare i giudici il cui numero di anni di esperienza è superiore alla media degli anni di esperienza dei giudici appartenenti allo stesso tribunale

```
SELECT DISTINCT G.GiudiceID
FROM GIUDICE G
WHERE G.AnniEsperienza > (SELECT AVG (AnniEsperienza)
                          FROM GIUDICE
                          WHERE IdTrib=G.AnniEsperienza)
```

FILA B

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

PERSONA(CF, Nome, Indirizzo, Città, Provincia, CAP, Telefono, DataNascita)
AVVOCATO (AvvocatoID, $CF^{PERSONA}$)
SPECIALIZZAZIONE(CodSpec, NomeSpec)
HA($AvvocatoID^{AVVOCATO}$,$CodSpec^{SPECIALIZZAZIONE}$)
ISCRIZIONE($AvvocatoID^{AVVOCATO}$, NomeAlbo, Data)
TRIBUNALE(IdTrib, NomeTribunale, Città, Regione, CAP)
GIUDICE(GiudiceID, $CF^{PERSONA}$, AnniEsperienza, $IdTrib^{TRIBUNALE}$)
CAUSA($IdTrib^{TRIBUNALE}$,NumCausa, DescrizioneCausa, TipoCausa)
UDIENZA($IdTrib^{CAUSA}$,$NumCausa^{CAUSA}$, NumUdienza, Data, Ora, Stanza, $GiudiceID^{GIUDICE}$)
COINVOLGE($CF^{PERSONA}$, $NumCausa^{CAUSA}$, $IdTrib^{CAUSA}$, RuoloP)
RAPPRESENTA($AvvocatoID^{AVVOCATO}$,$CF^{PERSONA}$, $NumCausa^{CAUSA}$, $IdTrib^{CAUSA}$, RuoloA)

Formulare le seguenti interrogazioni in **SQL**.

1. Determinare i giudici assegnati a un tribunale che non hanno ancora presieduto udienze in quel tribunale

```
SELECT GiudiceID
FROM GIUDICE LEFT OUTER NATURAL JOIN UDIENZA
WHERE NumUdienza IS NULL
```

```
SELECT DISTINCT GiudiceID
FROM GIUDICE
WHERE (GiudiceID, IdTrib) NOT IN (SELECT GiudiceID, IdTrib) FROM UDIENZA)
```

Oppure

```
SELECT DISTINCT G.GiudiceID
FROM GIUDICE G
WHERE NOT EXISTS (SELECT * FROM UDIENZA
                  WHERE GiudiceID=G.GiudiceID AND IdTrib = G.IdTrib)
```

2. Determinare gli avvocati la cui anzianità di iscrizione[1] a un albo è superiore alla media di anzianità di iscrizione degli avvocati iscritti allo stesso albo.

```
SELECT I.AvvocatoID
FROM ISCRIZIONE I
WHERE CURRENT_DATE-I.Data (SELECT AVG(CURRENT_DATE -Data)
                           FROM ISCRIZIONE
                           WHERE NomeAlbo=I.NomeAlbo)
```

GIUGNO 2024

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale, relativo a tornei di tennis:

GIOCATORE(IdG, Nome, Cognome, Genere, DataN, Nazione)
TORNEO(IdT, NomeT, Luogo, DataI, DataF, NumTurni, Tipo, Terreno)
CATEGORIA(IdCat, NomeCategoria, GenereCategoria)
REGISTRAZIONE(IdT^{TORNEO}, $IdCat^{CATEGORIA}$, NumRegistrazione, DataRegistrazione, TestaDiSerie)
GIOCAIN($IdT^{REGISTRAZIONE}$, IdCat, ^{REGISTRAZIONE} $NumRegistrazione^{REGISTRAZIONE}$, $IdG^{GIOCATORE}$)
NOTA: un giocatore nel caso di singoli, due nel caso di doppi, più di due nel caso di squadre (categorie)

Formulare le seguenti interrogazioni in **SQL**.

1. Le giocatrici che hanno partecipato sia nella categoria singoli che in quella doppi nello stesso torneo

```
SELECT DISTINCT IdG
FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN CATEGORIA
WHERE Genere='F' AND NomeCategoria='Singolo'
AND (IdG,IdT) IN (SELECT DISTINCT IdG,IdT
                  FROM GIOCAIN NATURAL JOIN CATEGORIA
                  WHERE NomeCategoria='Doppio')
```

2. Il torneo a cui hanno partecipato giocatori del maggior numero di nazioni diverse

```
SELECT DISTINCT IdT
FROM GIOCAIN NATURAL JOIN GIOCATORE
GROUP BY IdT
HAVING COUNT(DISTINCT Nazione) ≥ ALL (SELECT COUNT(DISTINCT Nazione)
                                      FROM GIOCAIN NATURAL JOIN GIOCAT.
                                      GROUP BY IdT)
```

FILA B

Esercizio II.B - SQL

In riferimento al seguente schema relazionale, relativo a tornei di tennis:

GIOCATORE(IdG, Nome, Cognome, Genere, DataN, Nazione)
TORNEO(IdT, NomeT, Luogo, DataI, DataF, NumTurni, Tipo, Terreno)
CATEGORIA(IdCat, NomeCategoria, GenereCategoria)
REGISTRAZIONE(IdT^{TORNEO}, IdCat^{CATEGORIA}, NumRegistrazione, DataRegistrazione, TestaDiSerie)
GIOCAIN(IdT^{REGISTRAZIONE}, IdCat, ^{REGISTRAZIONE} NumRegistrazione^{REGISTRAZIONE}, IdG^{GIOCATORE})
NOTA: un giocatore nel caso di singoli, due nel caso di doppi, più di due nel caso di squadre (categorie)

Formulare le seguenti interrogazioni in **SQL**.

1. I/Le giocatori/rici francesi che non sono mai stati teste di serie (attributo Booleano TestaDiSerie)

```
SELECT DISTINCT IdG
FROM GIOCATORE
WHERE Nazione='Francia'
AND IdG NOT IN (SELECT DISTINCT IdG
                FROM GIOCAIN NATURAL JOIN REGISTRAZIONE
                WHERE TestaDiSerie=true)
```

2. Il giocatore che ha disputato il maggior numero di tornei diversi

```
SELECT DISTINCT IdG
FROM GIOCAIN
GROUP BY IdG
HAVING COUNT(DISTINCT IdT) ≥ALL (SELECT COUNT(DISTINCT IdT)
                                 FROM GIOCAIN
                                 GROUP BY IdG)
```

FILA C

Esercizio II.B - SQL

In riferimento al seguente schema relazionale relativo a tornei di tennis:

GIOCATORE(IdG, Nome, Cognome, Genere, DataN, Nazione)
TORNEO(IdT, NomeT, Luogo, DataI, DataF, NumTurni, Tipo, Terreno)
CATEGORIA(IdCat, NomeCategoria, GenereCategoria)
REGISTRAZIONE(IdT^{TORNEO}, IdCat^{CATEGORIA}, NumRegistrazione, DataRegistrazione, TestaDiSerie)
GIOCAIN(IdT^{REGISTRAZIONE}, IdCat, ^{REGISTRAZIONE} NumRegistrazione^{REGISTRAZIONE}, IdG^{GIOCATORE})
NOTA: un giocatore nel caso di singoli, due nel caso di doppi, più di due nel caso di squadre (categorie)

Formulare le seguenti interrogazioni in **SQL**.

1. I/Le giocatori/rici italiani/e che hanno partecipato sia a US Open che a Australian Open nello stesso anno

```
SELECT DISTINCT IdG
FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN TORNEO
WHERE Nazione='Italia' AND NomeT='US Open'
AND (IdG, YEAR(DataI)) IN (SELECT DISTINCT IdG, YEAR(DataI)
                           FROM GIOCAIN NATURAL JOIN TORNEO
                           WHERE NomeT='Australian Open'
```

2. Il torneo con il maggior numero di teste di serie

```
SELECT DISTINCT IdT
FROM REGISTRAZIONE
WHERE TestaDiSerie
GROUP BY IdT
HAVING COUNT(*) ≥ALL(SELECT COUNT(*)
                     FROM REGISTRAZIONE
                     WHERE TestaDiSerie
                     GROUP BY IdT)
```

FILA D

Esercizio II.B - SQL

In riferimento al seguente schema relazionale relativo a tornei di tennis:

GIOCATORE(IdG, Nome, Cognome, Genere, DataN, Nazione)
TORNEO(IdT, NomeT, Luogo, DataI, DataF, NumTurni, Tipo, Terreno)
CATEGORIA(IdCat, NomeCategoria, GenereCategoria)
REGISTRAZIONE(IdT^{TORNEO}, IdCat^{CATEGORIA}, NumRegistrazione, DataRegistrazione, TestaDiSerie)
GIOCAIN(IdT^{REGISTRAZIONE}, IdCat, ^{REGISTRAZIONE} NumRegistrazione^{REGISTRAZIONE}, IdG^{GIOCATORE})
NOTA: un giocatore nel caso di singoli, due nel caso di doppi, più di due nel caso di squadre (categorie)

Formulare le seguenti interrogazioni in **SQL**.

1. I/Le giocatori/rici tedeschi/e che non hanno mai partecipato a Wimbledon

```
SELECT DISTINCT IdG
FROM GIOCATORE
WHERE Nazione='Germania'
AND IdG NOT IN (SELECT DISTINCT IdG
                FROM GIOCAIN NATURAL JOIN TORNEO
                WHERE NomeT='Wimbledon')
```

2. Il torneo con l'età media dei giocatori più alta

```
SELECT DISTINCT IdT
FROM TORNEO NATURAL JOIN GIOCAIN NATURAL JOIN
                                 GIOCATORE
GROUP BY IdT
HAVING AVG(DataN-DataI) ≥ALL
       (SELECT AVG(DataN-DataI)
        FROM TORNEO NATURAL JOIN GIOCAIN
                    NATURAL JOIN GIOCATORE
        GROUP BY IdT)
```

GIUGNO 2023

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

SPEDIZIONE(CodiceSp, Mittente[SOGGETTO], Destinatario[SOGGETTO], IndirizzoSpedizione$_0$, CodiceCat[CATEGORIA], Stato)
SOGGETTO(CodiceSogg, Nome, Indirizzo)
CATEGORIA(CodiceCat *Descrizione*, Prezzo)
PACCO(CodiceSp[SPEDIZIONE], NumPacco, Peso)
LUOGO(Coordinate, Descrizione)
POSIZIONE(CodiceSp[SPEDIZIONE], Coordinate[LUOGO], DataOra)

La spedizione viene effettuata all'indirizzo del destinatario, a meno che non sia indicato un indirizzo di spedizione. Possibili valori per Stato in SPEDIZIONE sono consegnata, rifiutata, in viaggio. Possibili valori per Descrizione in CATEGORIA sono standard, fast, express.

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare i nomi dei soggetti che non hanno mai effettuato (cioè non sono mai stati mittenti di) spedizioni in categoria "express".

```
SELECT DISTINCT Nome
FROM SOGGETTO
WHERE CodiceSogg NOT IN (SELECT Mittente
                         FROM SPEDIZIONE NATURAL JOIN CATEGORIA
                         WHERE Descrizione='express')
```

2. Determinare lo stato della spedizione con peso complessivo massimo.

```
SELECT DISTINCT Stato
FROM SPEDIZIONE NATURAL JOIN PACCO
GROUP BY CodiceSp
HAVING SUM(Peso) ≥ ALL (SELECT SUM(Peso)
                        FROM PACCO
                        GROUP BY CodiceSp)
```

FILA B

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

SPEDIZIONE(CodiceSp, Mittente[SOGGETTO], Destinatario[SOGGETTO], IndirizzoSpedizione$_0$, CodiceCat[CATEGORIA], Stato)
SOGGETTO(CodiceSogg, Nome, Indirizzo)
CATEGORIA(CodiceCat *Descrizione*, Prezzo)
PACCO(CodiceSp[SPEDIZIONE], NumPacco, Peso)
LUOGO(Coordinate, Descrizione)
POSIZIONE(CodiceSp[SPEDIZIONE], Coordinate[LUOGO], DataOra)

La spedizione viene effettuata all'indirizzo del destinatario, a meno che non sia indicato un indirizzo di spedizione. Possibili valori per Stato in SPEDIZIONE sono consegnata, rifiutata, in viaggio. Possibili valori per Descrizione in CATEGORIA sono standard, fast, express.

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare i soggetti che non hanno mai rifiutato spedizioni (cioè non sono mai stati destinatari di spedizioni con stato "rifiutata").

```
SELECT DISTINCT CodiceSogg
FROM SOGGETTO
WHERE CodiceSogg NOT IN (SELECT Destinatario
                         FROM SPEDIZIONE
                         WHERE Stato='rifiutata')
```

2. Determinare la spedizione con stato "in viaggio" che contiene il maggior numero di pacchi.

```
SELECT DISTINCT CodiceSp
FROM SPEDIZIONE NATURAL JOIN PACCO
WHERE Stato='in viaggio'
GROUP BY CodiceSp
HAVING COUNT(*) ≥ ALL (SELECT COUNT(*)
                       FROM SPEDIZIONE NATURAL JOIN PACCO
                       WHERE Stato='in viaggio'
                       GROUP BY CodiceSp)
```

LUGLIO 2023

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale, relativo ad assicurazioni sanitarie e alle relative pratiche per il rimborso delle prestazioni coperte dall'assicurazione

POLIZZA(NumP, Titolare[TITOLARE], DataIn, DataScad, Tipo, ImportoP)
PRATICA(CodicePr, NumP[POLIZZA], DataPratica)
PRESTAZIONE(CodicePr[PRATICA], NumRiga, Erogante[SOGGETTO], Beneficiario[ASSISTITO], Descrizione, DataPrest, ImportoPrest)
ASSISTITO(CodiceFisc, Nome, Cognome, Genere, DataN)
SOGGETTO(CodiceSogg, Denominazione, Indirizzo)
TITOLARE(Titolare[ASSISTITO], *Email*, Telefono, Indirizzo, Convenzione$_0$)
FAMILIARE(Familiare[ASSISTITO], Titolare[TITOLARE])
COPRE_FAM(NumP[POLIZZA], Familiare[FAMILIARE])

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare la data della pratica, il numero di polizza e nome e cognome del titolare della polizza per le pratiche presentate nell'ultimo mese che non contengono prestazioni di tipo visita (cioè tali che Descrizione include la stringa 'visita').

```
SELECT DataPratica, NumP, Nome, Cognome
FROM PRATICA NATURAL JOIN POLIZZA JOIN ASSISTITO ON Titolare=CodiceFisc
WHERE DataPratica ≥ CURRENT_DATE - INTERVAL 1 month AND DataPratica ≤ CURRENT_DATE
AND CodicePr NOT IN (SELECT CodicePr
                     FROM PRESTAZIONE
                     WHERE Descrizione LIKE '%visita%')
```

2. Determinare le polizze in corso che coprono tutti i familiari del titolare.

```
SELECT NumP
FROM POLIZZA P NATURAL JOIN COPRE_FAM
WHERE DataScadenza ≥ CURRENT_DATE
GROUP BY NumP, Titolare
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM FAMILIARE
                   WHERE Titolare = P.titolare)
```

FILA B

Esercizio II.B - SQL

In riferimento al seguente schema relazionale, relativo ad assicurazioni sanitarie e alle relative pratiche per il rimborso delle prestazioni coperte dall'assicurazione

ASSISTITO(CodiceFisc, Nome, Cognome, Genere, DataN)
TITOLARE(Titolare^{ASSISTITO}, *Email*, Telefono, Indirizzo, Convenzione$_O$)
FAMILIARE(Familiare^{ASSISTITO}, Titolare^{TITOLARE})
POLIZZA(NumP, Titolare^{TITOLARE}, DataIn, DataScad, Tipo, ImportoP)
COPRE_FAM(NumP^{POLIZZA}, Familiare^{FAMILIARE})
SOGGETTO(CodiceSogg, Denominazione, Indirizzo)
PRATICA(CodicePr, NumP^{POLIZZA}, DataPratica)
PRESTAZIONE(CodicePr^{PRATICA}, NumRiga, Erogante^{SOGGETTO}, Beneficiario^{ASSISTITO}, Descrizione, DataPrest, ImportoPrest)

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare l'email e il numero di telefono del titolare insieme al numero di polizza per le polizze scadute (=data di scadenza precedente alla data odierna) per cui non sono state presentate pratiche.

```
SELECT Email,Telefono, NumP
FROM TITOLARE NATURAL JOIN POLIZZA
WHERE DataScad ≤ CURRENT_DATE
AND NumP NOT IN (SELECT NumP
                 FROM PRATICA)
```

2. Determinare le polizze il cui importo è inferiore all'importo delle prestazioni coperte da tale polizza.

```
SELECT NumP
FROM POLIZZA NATURAL JOIN PRATICA
             NATURAL JOIN PRESTAZIONE
GROUP BY NumP, ImportoP
HAVING SUM (ImportoPrest) > ImportoP
```

importo polizza inferiore a
importo delle prestazioni

GENNAIO 2024

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale (semplificazione dello schema dell'esercizio I)

AGENZIA (NomeAg, Indirizzo, Tel, Città, Stato)
VEICOLO (Targa, CategoriaV, Modello, Marca, Alimentazione, DataImm)
CLIENTE (IdCli, Nome, Indirizzo, Tel, *Email*, DataN, *NumeroPat*, ScadenzaPat)
PRENOTAZIONE (IdP, *Targa*^{VEICOLO}, *DataRitiro*, DataConsegna, AgRitiro^{AGENZIA}, AgConsegna^{AGENZIA}, IdCli^{LIENTE}, Stato, Importo)
NOLEGGIO (IdP^{PRENOTAZIONE}, KmIniziali, KmFinali, Carburante, Costo)
GUIDATORE_AGG (IdP^{PRENOTAZIONE}, IdCli^{CLIENTE})

Formulare le seguenti interrogazioni in **SQL**.

1. Determinare le prenotazioni con stato "confermato" e data consegna precedente alla data odierna, per cui non sono presenti le informazioni del corrispondente noleggio.

```
SELECT IdP
FROM PRENOTAZIONE
WHERE Stato='confermato AND DataConsegna<CURRENT_DATE
AND IdP NOT IN (SELECT IdP
                FROM NOLEGGIO)
```

2. Determinare marca e modello del veicolo noleggiato nel noleggio in cui sono stati percorsi più km.

```
SELECT Marca, Modello
FROM VEICOLO NATURAL JOIN PRENOTAZIONE
             NATURAL JOIN NOLEGGIO
WHERE KmIniziali - KmFinali = (SELECT MAX(KmIniziali - KmFinali)
                               FROM NOLEGGIO)
```

FILA B

Su una piccola istanza, la richiesta e l'interrogazione formulata restituiscono lo stesso risultato

Esercizio II.B - SQL

In riferimento al seguente schema relazionale (semplificazione dello schema dell'esercizio I)

AGENZIA (NomeAg, Indirizzo, Tel, Città, Stato)
VEICOLO (Targa, CategoriaV, Modello, Marca, Alimentazione, DataImm)
CLIENTE (IdCli, Nome, Indirizzo, Tel, *Email*, DataN, *NumeroPat*, ScadenzaPat)
PRENOTAZIONE (IdP, *Targa*^{VEICOLO}, *DataRitiro*, DataConsegna, AgRitiro^{AGENZIA}, AgConsegna^{AGENZIA}, IdCli^{LIENTE}, Stato, Importo)
NOLEGGIO (IdP^{PRENOTAZIONE}, KmIniziali, KmFinali, Carburante, Costo)
GUIDATORE_AGG (IdP^{PRENOTAZIONE}, IdCli^{CLIENTE})

Formulare le seguenti interrogazioni in **SQL**.

1. Determinare le targhe dei veicoli elettrici che non sono mai stati noleggiati in un'agenzia e riconsegnati in un'altra.

```
SELECT DISTINCT Targa
FROM VEICOLO
WHERE Alimentazione='elettrica'
AND Targa NOT IN (SELECT DISTINCT Targa
                  FROM PRENOTAZIONE
                  WHERE AgRitiro != AgConsegna)
```

2. Determinare i dati di contatto (email e telefono) del cliente che ha effettuato il noleggio (concluso) di durata massima.

```
SELECT Email, Telefono
FROM CLIENTE NATURAL JOIN PRENOTAZIONE
             NATURAL JOIN NOLEGGIO
WHERE Stato='concluso'
AND (DataConsegna-DataRitiro) = (SELECT MAX(DataCons. - DataRit.)
                                 FROM PRENOTAZIONE
                                 NATURAL JOIN NOLEGGIO
                                 WHERE stato='concluso')
```

GIUGNO 2022

FILA A

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

UTENZE(Prefisso[DISTRETTI], Numero, Titolare[PERSONE], Indirizzo)
PERSONE(CodiceFiscale, Cognome, Nome, DataDiNascita)
DISTRETTI(Prefisso, NomeDistretto, Provincia[PROVINCE])
PROVINCE(Sigla, NomeProvincia)
BOLLETTE(CodiceBolletta, Prefisso[UTENZE], Numero[UTENZE], DataEmissione, DataScadenza, Importo)
PAGAMENTI(CodicePagamento, Bolletta[BOLLETTE], Data, Importo, Modalità)

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare per ogni utenza il prefisso, il numero, il codice fiscale del titolare, l'importo totale dovuto (cioè il totale delle bollette emesse), l'importo totale pagato (cioè il totale dei pagamenti) e il relativo saldo.

```
D = SELECT Prefisso, Numero, Titolare, SUM(Importo) AS Dovuto
    FROM BOLLETTE
    GROUP BY Prefisso, Numero, Titolare

P = SELECT Prefisso, Numero, SUM(P.Importo) AS Pagato
    FROM BOLLETTE JOIN PAGAMENTI ON Bolletta = CodiceBolletta
    GROUP BY Prefisso, Numero

SELECT prefisso, numero, Titolare, Dovuto, pagato, Dovuto - Pagato AS saldo
FROM D NATURAL LEFT OUTER JOIN P
```

2. Determinare le utenze per cui nell'ultimo mese (DataEmissione = 01/06/2022) sono state emesse bollette con importo superiore all'importo medio delle bollette emesse nel loro distretto.

```
SELECT B.Prefisso, B.Numero
FROM BOLLETTE B
WHERE B.DataEmissione = DATE '01/06/2022' AND B.Importo > (SELECT AVG(Importo)
                                                           FROM BOLLETTE
                                                           WHERE Prefisso = B.Prefisso AND
                                                           DataEmissione = DATE '01/06/2022')
```

FILA B

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

UTENZE(Prefisso[DISTRETTI], Numero, Titolare[PERSONE], Indirizzo)
PERSONE(CodiceFiscale, Cognome, Nome, DataDiNascita)
DISTRETTI(Prefisso, NomeDistretto, Provincia[PROVINCE])
PROVINCE(Sigla, NomeProvincia)
BOLLETTE(CodiceBolletta, Prefisso[UTENZE], Numero[UTENZE], DataEmissione, DataScadenza, Importo)
PAGAMENTI(CodicePagamento, Bolletta[BOLLETTE], Data, Importo, Modalità)

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare per ogni distretto in provincia di Genova il numero di utenze in quel distretto e l'importo totale delle bollette emesse negli ultimi 5 anni.

```
SELECT Prefisso, COUNT(DISTINCT Numero), SUM(Importo)
FROM PROVINCE JOIN DISTRETTI ON Sigla = Provincia NATURAL JOIN BOLLETTE
WHERE NomeProvincia = 'Genova' AND DataEmissione >= CURRENT_DATE - INTERVAL '5 YEARS'
```

2. Determinare le date di emissione delle bollette di importo superiore all'importo medio delle bollette emesse per la stessa utenza.

```
SELECT B.DataEmissione, B.Prefisso, B.Numero
FROM BOLLETTE B
WHERE B.Importo > (SELECT AVG(Importo)
                   FROM BOLLETTE
                   WHERE Prefisso = B.Prefisso AND Numero = B.Numero)
```

LUGLIO 2022

Esercizio II.B - SQL

In riferimento al seguente schema relazionale:

DIPENDENTE(ID, Cognome, Nome, Dipartimento[DIPARTIMENTO])
PROFESSORE(ID[DIPENDENTE], Qualifica, AnnoNascita)
DIPARTIMENTO(Codice, *Nome*, Indirizzo, Direttore[PROFESSORE])
CORSODISTUDIO(Codice, *Nome*, Dipartimento[DIPARTIMENTO])
COLLABORAZIONE(CorsoDiStudio[CORSODISTUDIO], Professore[PROFESSORE], Tipo)
CORSO(Codice, Materia[MATERIA], Docente[PROFESSORE], CorsoDiStudio[CORSODISTUDIO], Semestre)
MATERIA(Sigla, *Nome*)

Formulare le seguenti interrogazioni in **SQL**.
1. Determinare per ogni dipartimento il direttore, il numero di dipendenti e il numero di corsi di studio gestiti dal dipartimento.

```
SELECT dipartimento, direttore, COUNT(DISTINCT ID), COUNT (DISTINCT CORSODISTUDIO.Codice)
FROM DIPENDENTE JOIN CORSODISTUDIO USING (dipartimento) JOIN DIPARTIMENTO ON Dipartimento=DIPARTIMENTO.Codice
GROUP BY Dipartimento, Direttore
```

2. Determinare la materia che è insegnata (= ci sono corsi relativi a quella materia) nel maggior numero di corsi di studi diversi.

```
SELECT Materia
FROM CORSI
GROUP BY Materia
WHERE COUNT(DISTINCT CorsoDiStudio) ≥ ALL (SELECT COUNT(DISTINCT CorsoDiStudio)
                                           FROM CORSO
                                           GROUP BY Materia)
```

SETTEMBRE 2022

FILA A

Esercizio II.B - SQL

In riferimento al seguente (frammento di) schema relazionale:

Pezzi(CodPezzo, Titolo, CodAutore[Compositori])
Compositori(CodAutore, Cognome, Nome)
Concerti(CodConc, Titolo, Descrizione, NomeO[Orchestre], *Teatro, Data, Ora*)
Programmazione(CodPezzo[Pezzi], CodConc[Concerto], Posizione)
(nota: Posizione è un intero che indica la posizione di un pezzo nell'ambito di un concerto; ad esempio 1 è il primo pezzo o pezzo di apertura, 2 il secondo, e così via)
Orchestre(NomeO, Sede, Direttore[Musicisti])

Formulare le seguenti interrogazioni in **SQL**.

1. Per ogni orchestra, determinare il numero di concerti tenuti e il numero di pezzi in programmazione in tali concerti.

```
SELECT NomeO, COUNT (DISTINCT CodConc) AS NumeroConcerti, COUNT(*) AS NumeroPezzi
FROM CONCERTI NATURAL JOIN PROGRAMMAZIONE
GROUP BY NomeO
```

2. Determinare per ciascuna orchestra, il nome del teatro dove ha tenuto l'ultimo concerto.

```
SELECT C.NomeO, C.teatro
FROM CONCERTI C
WHERE C.Data ≥ ALL (SELECT Data
                    FROM CONCERTI
                    WHERE NomeO = C.NomeO)
```

FILA B

Esercizio II.B - SQL

In riferimento al seguente (frammento di) schema relazionale:

Pezzi(CodPezzo, Titolo, CodAutore[Compositori])
Compositori(CodAutore, Cognome, Nome)
Concerti(CodConc, Titolo, Descrizione, NomeO[Orchestre], *Teatro, Data, Ora*)
Programmazione(CodPezzo[Pezzi], CodConc[Concerto], Posizione)
(nota: Posizione è un intero che indica la posizione di un pezzo nell'ambito di un concerto; ad esempio 1 è il primo pezzo o pezzo di apertura, 2 il secondo, e così via)

Formulare le seguenti interrogazioni in **SQL**.

1. Per ogni concerto, determinare il numero di pezzi in programmazione e il numero di compositori di cui sono stati suonati pezzi.

```
SELECT CodConc, COUNT(*) AS NumPezzi, COUNT (DISTINCT CodAutore) AS NumCompositori
FROM PROGRAMMAZIONE NATURAL JOIN PEZZI
GROUP BY CodConc
```

2. Determinare per ciascun concerto codice e titolo del pezzo che viene suonato per ultimo (cioè quello con il valore massimo per la posizione).

```
SELECT P.CodConc, CodPezzo, Titolo
FROM PROGRAMMAZIONE P NATURAL JOIN PEZZI
WHERE P.Posizione = (SELECT MAX (Posizione)
                     FROM PROGRAMMAZIONE
                     WHERE CodConc = P.CodConc)
```

GENNAIO 2023

Esercizio II.B - SQL

In riferimento al seguente (frammento di) schema relazionale:

ALUNNO (cf, nome, cognome, genere, classe, sezione, esito)
PROF (idProf, nomeProf, cognomeProf, genereProf, dataN)
INSEGNA (classe, sezione, materia, idProf[PROF])
VOTO (cf[ALUNNO], materia, voto, debito)

dove i valori possibili per esito in ALUNNO sono 'ammesso', 'non ammesso' e debito in VOTO è a valori Booleani, formulare le seguenti interrogazioni **in SQL**.

1. Determinare i professori di matematica che non hanno assegnato valutazioni inferiori al sette (nota: un prof assegna una valutazione se insegna quella materia nella classe -e sezione- frequentata dall'alunno a cui è assegnata la valutazione).

```
SELECT idProf
FROM INSEGNA I
WHERE Materia='matematica' AND NOT EXIST (SELECT *
                                          FROM VOTO NATURAL JOIN ALUNNO
                                          WHERE VOTO < 7 AND materia='matematica' AND
                                                classe = I.classe AND sezione=I.sezione)
```

2. Determinare gli alunni la cui votazione di matematica è più alta della media dei voti attribuiti in matematica nella loro classe.

```
SELECT CF
FROM VOTO NATURAL JOIN ALUNNO A
WHERE Materia='matematica' AND VOTO > (SELECT AVG(VOTO)
                                       FROM VOTO NATURAL JOIN ALUNNO
                                       WHERE Materia='matematica' AND classe=A.classe
                                             AND sezione = A.sezione)
```

FEBBRAIO 2023

Esercizio II.B - SQL

In riferimento al seguente (frammento di) schema relazionale:

DOODLE(IdD, Titolo, Luogo, Descrizione, TimeZone, IdCreatore[PARTECIPANTE])
SLOT(IdD[DOODLE], NumS, Data, OraI, OraF)
PARTECIPANTE(IdP, Nome, Cognome)
RISPOSTA(IdD[SLOT], NumS[SLOT], IdP[PARTECIPANTE], Risp)

dove i valori possibili per Risp sono 'si', 'no', 'se necessario', formulare le seguenti interrogazioni in **SQL**.

1. Determinare il doodle contenente lo slot con l'ora di fine più tarda

```
SELECT IdD
FROM SLOT
WHERE OraF = (SELECT MAX(OraF)
              FROM SLOT)
```

2. Determinare i doodle che prevedono slot di durata diversa

```
SELECT S.IdD
FROM SLOT S
WHERE (OraF-OraI) ≠ ANY (SELECT (OraF-OraI)
                         FROM SLOT
                         WHERE IdD=S.IdD)
```

GIUGNO 2021

FILA A

Esercizio 4. (SQL)

In riferimento al seguente schema:

SEDE(IdSede, *NomeSede*, Indirizzo, CAP, Comune)
DIPENDENTE(Email, Nome, Cognome, Qualifica, Telefono, IdSede[SEDE])
AULA(IdAula, IdSede[SEDE], NomeAula$_O$, TipoAula, Capienza, Accessibile)
PRENOTAZIONE(IdAula[AULA], IdSede[AULA], Data, OraI, OraF, Descrizione, TipoAttività, PartecipantiAttesi, PrenotataDa[DIPENDENTE], ConfermataDa[DIPENDENTE])

Formulare le seguenti interrogazioni in SQL

a) Determinare per ogni sede il numero di prenotazioni effettuate, il numero medio di partecipanti attesi e quanti dipendenti diversi hanno confermato le prenotazioni

```
SELECT IdSede, COUNT(*), AVG(PartecipantiAttesi), COUNT (DISTINCT ConfermataDa)
FROM PRENOTAZIONE
GROUP BY IdSede
```

b) Determinare le aule la cui capienza è superiore alla capienza media delle aule dello stesso tipo nella stessa sede

```
SELECT IdAula, IdSede
FROM AULA A
WHERE Capienza > (SELECT AVG(Capienza)
                  FROM AULA
                  WHERE TipoAula=A.TipoAula AND IdSede=A.IdSede)
```

FILA B

Esercizio 4. (SQL)

In riferimento al seguente schema:

SEDE(IdSede, *NomeSede*, Indirizzo, CAP, Comune)
DIPENDENTE(Email, Nome, Cognome, Qualifica, Telefono, IdSede^{SEDE})
AULA(IdAula, IdSede^{SEDE}, NomeAula$_O$, TipoAula, Capienza, Accessibile)
PRENOTAZIONE(IdAula^{AULA}, IdSede^{AULA}, Data, OraI, OraF, Descrizione, TipoAttività, PartecipantiAttesi, PrenotataDa^{DIPENDENTE}, ConfermataDa^{DIPENDENTE})

Formulare le seguenti interrogazioni in SQL

a) Determinare per ogni sede il numero di prenotazioni effettuate per aule di tale sede, quante aule diverse sono state prenotate e il numero medio di partecipanti attesi

```
SELECT IdSede, COUNT(*), COUNT(DISTINCT IdAula), AVG(PartecipantiAttesi)
FROM PRENOTAZIONE
GROUP BY Id Sede
```

b) Determinare le aule la cui capienza è inferiore alla capienza media delle aule dello stesso tipo

```
SELECT IdAula, IdSede
FROM Aula A
WHERE capienza < (SELECT AVG(capienza)
                  FROM AULA
                  WHERE TipoAula = A.TipoAula)
```

LUGLIO 2021

FILA A

Esercizio 4. (SQL)

In riferimento al seguente schema:

QUIZ(IdQ, *Titolo*, TempoMax$_O$, PuntiTotali)
DOMANDA(IdQ^{QUIZ}, NumD, Testo, Punti)
RISPOSTA(IdQ^{DOMANDA}, NumD^{DOMANDA}, Opzione, Testo, Percentuale, Feedback$_O$)
STUDENTE(MatrS, Nome, Cognome)
COMPILAZIONE(*MatrS*^{STUDENTE}, IdQ^{QUIZ}, NumTentativo, *DataOraInizio*, DataOraFine$_O$, Punteggio$_O$)
SCELTA(MatrS^{COMPILAZIONE}, IdQ^{COMPILAZIONE,RISPOSTA}, NumTentativo^{COMPILAZIONE}, NumD^{RISPOSTA}, Opzione^{RISPOSTA})

Formulare le seguenti interrogazioni in SQL

a) Determinare il titolo del quiz di punteggio massimo tra quelli senza limite di tempo

```
SELECT Titolo
FROM QUIZ
WHERE TempoMax IS NULL AND PuntiTotali = (SELECT MAX(PuntiTotali)
                                          FROM QUIZ
                                          WHERE TempoMax IS NULL)
```

b) Determinare gli studenti che hanno effettuato almeno un tentativo per tutti i quiz

```
SELECT MatrS
FROM COMPILAZIONE
GROUP BY MatrS
HAVING COUNT(DISTINCT IdQ) = (SELECT COUNT(*)
                              FROM QUIZ)
```

FILA B

Esercizio 4. (SQL)

In riferimento al seguente schema:

QUIZ(IdQ, *Titolo*, TempoMax$_O$, PuntiTotali)
DOMANDA(IdQ^{QUIZ}, NumD, Testo, Punti)
RISPOSTA(IdQ^{DOMANDA}, NumD^{DOMANDA}, Opzione, Testo, Percentuale, Feedback$_O$)
STUDENTE(MatrS, Nome, Cognome)
COMPILAZIONE(*MatrS*^{STUDENTE}, IdQ^{QUIZ}, NumTentativo, *DataOraInizio*, DataOraFine$_O$, Punteggio$_O$)
SCELTA(MatrS^{COMPILAZIONE}, IdQ^{COMPILAZIONE,RISPOSTA}, NumTentativo^{COMPILAZIONE}, NumD^{RISPOSTA}, Opzione^{RISPOSTA})

Formulare le seguenti interrogazioni in SQL

a) Determinare l'ultima compilazione iniziata tra quelle non ancora completate

```
SELECT MatrS, IdQ, NumTentativo
FROM COMPILAZIONE
WHERE DataOraFine IS NULL AND DataOraInizio = (SELECT MAX(DataOraInizio)
                                               FROM COMPILAZIONE
                                               WHERE DataOraFine IS NULL)
```

b) Determinare i quiz per cui tutti gli studenti hanno effettuato almeno un tentativo

```
SELECT IdQ
FROM COMPILAZIONE
GROUP BY IdQ
WHERE COUNT(DISTINCT MatrS) = (SELECT COUNT(*)
                               FROM STUDENTE)
```

LUGLIO 2021

Esercizio 4. (SQL)

In riferimento al seguente schema relazionale, relativo ad un sondaggio di tipo "doodle" utilizzato per pianificare riunioni/incontri, dove i valori possibili per Risp sono 'sì', 'no', 'se necessario'.

SONDAGGIO (IdD, Titolo, Luogo, Descrizione, TimeZone, IdCreatore^{PARTECIPANTE})
QUANDO (*IdD*^{SONDAGGIO}, NumO, *Data*, *OraI*, OraF)
PARTECIPANTE (IdP, Nome, Cognome)
RISPOSTA (IdD^{QUANDO}, NumO^{QUANDO}, IdP^{PARTECIPANTE}, Risp)
NOTA (IdD^{SONDAGGIO}, IdP^{PARTECIPANTE}, TestoNota)
QUANDO_SCELTO (IdD^{QUANDO}, NumO^{QUANDO})

Formulare le seguenti interrogazioni **in SQL.**

a) Determinare i sondaggi per cui è stata data almeno una risposta e non è ancora stata effettuata una scelta su quando si terrà l'incontro

```
SELECT DISTINCT IdD
FROM RISPOSTA
WHERE IdD NOT IN (SELECT IdD
                  FROM SCELTO_QUANDO)
```

b) Determinare i sondaggi che hanno più (=almeno due) opzioni orarie alternative (=più quando) nella stessa data

```
SELECT IdD
FROM QUANDO
GROUP BY IdD, Data
HAVING COUNT(*) ≥ 2
```