

BASI DI DATI

Prof. Pistolesi

INDICE PISTOLESI-SQL

Pagina 8

- Query in SQL: SELECT, FROM, WHERE
- Tutti gli attributi: SELECT *
- Operatore logico: AND
- Operatore logico: OR

Pagina 9

- Intervalli di valori: BETWEEN, >= e <=
- Eliminazione duplicati: DISTINCT
- Valori NULL: IS NOT NULL
- Valori NULL: IS NULL

Pagina 10

- Tipi di dato data
- Alcuni formati utili
- Formattazione delle date: DATE_FORMAT(AttributoDaFormattare,'Formato')
- Giorno / Mese / Anno: DAY / MONTH / YEAR (Attributo)

Pagina 11

- Giorno / Mese / Anno, con variabile di sistema: DAY / MONTH / YEAR (CURRENT_DATE)
- Durata in giorni: DATEDIFF(DataPiùRecente,DataPiùRemota)
- Durata in mesi: PERIOD_DIFF(DataPiùRecente,DataPiùRemota)

Pagina 12

- Sintassi INTERVAL
- Sommare / Sottrarre intervalli di tempo con funzioni:
DATE_ADD / DATE SUB (PrimaData, LassodiTempo-INTERVAL)
- Sommare / Sottrarre intervalli di tempo direttamente con + e - usando INTERVAL
- Funzioni di utilità sulle date

Pagina 13

- Conteggio con ridenominazione: COUNT(*) AS
- Conteggio su attributo/i: COUNT(DISTINCT)
- Somma: SUM

Pagina 14

- Media: AVG
- Minimo e massimo: MIN / MAX
- INNER JOIN (con ALIAS)
- NATURAL JOIN

Pagina 15

- Prodotto cartesiano: CROSS JOIN
- LEFT OUTER JOIN

Pagina 16

- RIGHT OUTER JOIN
- Query con INNER JOIN e condizioni sui record
- JOIN Multiplo

Pagina 17

- Ridenominazione nel FROM
- SELF JOIN

Pagina 18

- SELF JOIN con USING
- DERIVED TABLE

Pagina 19

- NON CORRELATED SUBQUERY (con IN)
- NON CORRELATED SUBQUERY (con NOT IN)

Pagina 20

- Annidamento multiplo di NON CORRELATED SUBQUERY con versione JOIN-equivalente

INDICE PISTOLESI-SQL

Pagina 21

- NON CORRELATED SUBQUERY scalare con operatori di confronto ($>$, $<$, $=$, \geq , \leq)
- Risoluzione mista NON CORRELATED SUBQUERY-JOIN

Pagina 22

- CORRELATED SUBQUERY (IN/NOT IN)
- CORRELATED SUBQUERY scalare con operatori di confronto ($>$, $<$, $=$, \geq , \leq)

Pagina 23

- CORRELATED SUBQUERY nel SELECT
- CTE

Pagina 24

- CTE multiple
- Sintassi di WITH per CTEs (Common Table Expression)

Pagina 25

- Raggruppamento GROUP BY
- GROUP BY dentro una DERIVED TABLE
- Condizioni sui gruppi: clausola HAVING

Pagina 26

- Condizioni sui gruppi + (SUBQUERY scalare + DERIVED TABLE)
- Condizioni sui gruppi + condizione sui record

Pagina 27

- Raggruppamento su più attributi

Pagina 28

- EXISTS
- NOT EXISTS

Pagina 29

- Unione OR
- Unione senza duplicati UNION

Pagina 30

- Unione con duplicati UNION ALL

Pagina 31

- Divisione: con doppio NOT EXISTS
- Divisione: con raggruppamento

Pagina 32

- Differenza con NOT IN
- Differenza a più insiemi (NOT IN + Subquery con \neq “diverso”)

Pagina 33

- Modificatori: SINTASSI (ANY/ALL)
- Modificatori “Almeno un record del result set” \rightarrow ANY
- Modificatori “Tutti i record del result set” \rightarrow ALL

Pagina 34

- Modificatori “Tutti i record del result set + EX AEQUO”
- Query complesse

Pagina 36

- Inserimento valori statici: SINTASSI
- Inserimento valori statici (tutti i valori): INSERT INTO - VALUES
- Inserimento per mancanza di informazioni
- Inserimento valori ricavati: SINTASSI

Pagina 37

- Inserimento valori ricavati: INSERT INTO – Query
- Aggiornamento: SINTASSI

INDICE PISTOLESI-SQL

Pagina 38

- Aggiornamento UPDATE - SET – WHERE
- Cancellazione: SINTASSI
- Cancellazione DELETE FROM - WHERE

Pagina 39

- Cancellazione: Riferimento alla tabella target nella condizione

Pagina 41

- STORED PROCEDURE

Pagina 42

- Variabili locali: DECLARE – DEFAULT (SINTASSI)
- Assegnamento statico (variabile) con SET
- Assegnamento calcolato (variabile) con SELECT + INTO

Pagina 43

- Assegnamento statico (variabile) con SET
- Parametri di ingresso con (IN_nomeParametro)

Pagina 44

- Parametri di uscita con (OUT nomeParametro_) + variabile USER-DEFINED @

Pagina 45

- Istruzione IF THEN – ELSEIF THEN – ELSE – END (SINTASSI)
- Istruzione CASE – WHEN THEN – END CASE (SINTASSI)

Pagina 46

- Esempio di utilizzo di IF

Pagina 47

- Più variabili di uscita – Ritorno di un unico record (LIMIT)
- WHILE DO – END WHILE (SINTASSI)

Pagina 48

- REPEAT – UNTIL – END REPEAT (SINTASSI)
- LOOP – END LOOP (SINTASSI)
- LEAVE
- ITERATE

Pagina 49

- Esempio di LOOP con istruzioni di salto: LEAVE / ITERATE + Concatenazione CONCAT

Pagina 50

- Cursori: DECLARE – CURSOR FOR (SINTASSI)
- Apertura, prelevamento, chiusura cursori: OPEN, FETCH, CLOSE (SINTASSI)
- Cursori: NOT FOUND CONTINUE HANDLER (SINTASSI)
- ESEMPIO NOT FOUND CONTINUE HANDLER

Pagina 51

- FUNZIONAMENTO COMPLETO DEL CURSOR
- Esempio di funzionamento del cursore COMPLETO + parametri di uscita (INOUT)

Pagina 53

- Gestione degli errori con HANDLER (SINTASSI)
- Gestione degli errori con CONTINUE HANDLER-SQLEXCEPTION
- Gestione degli errori con EXIT HANDLER-ROLLBACK

Pagina 54

- Sollevare gli errori: SIGNAL SQLSTATE – SET (SINTASSI)
- Esempio sollevare gli errori: SIGNAL SQLSTATE - SET

Pagina 55

- STORED FUNCTION (SINTASSI)

Pagina 56

- Esempio di STORED FUNCTION

INDICE PISTOLESI-SQL

Pagina 57

- Esempio di utilizzo di una FUNCTION

Pagina 58

- Esempio di utilizzo di una FUNCTION + TEMPORARY TABLE

Pagina 59

- Sintassi TRIGGER
- Sintassi TRIGGER (+NEW)

Pagina 60

- Sintassi TRIGGER multi-statement
- TRIGGER multi-statement

Pagina 62

- TRIGGER (attributo ridondante)

Pagina 63

- TRIGGER (business rule)

Pagina 64

- Aggiunta ridondanza e assegnamento (con ALTER TABLE – ADD COLUMN)

Pagina 65

- Ridondanza giornaliera (e periodica)
→ Recurring (EVENT – ON SCHEDULE EVERY 1 DAY – (STARTS) – DO)

Pagina 66

- Sintassi ON SCHEDULE (Periodico)
- Sintassi ON SCHEDULE – STARTS (Periodico – Inizio)
- Sintassi ON SCHEDULE – STARTS (Periodico – Inizio/Fine)
- Sintassi ON SCHEDULE di EVENT a singolo scatto
- Sintassi CREATE TABLE

Pagina 67

- Materialized view (introduzione)
- Materialized view (esempio)

Pagina 71

- Materialized view (IMMEDIATE / ON DEMAND / DEFERRED REFRESH)

Pagina 73

- Materialized view – INCREMENTAL REFRESH (concetto)
- Materialized view (PARTIAL REFRESH)

Pagina 76

- Materialized view (PARTIAL REFRESH, COMPLETE REFRESH, REBUILD)

Pagina 78

- WINDOW FUNCTIONS: clausola OVER() su funzione aggregata
- WINDOW FUNCTIONS: clausola OVER(PARTITION BY NomePartizione) su funzione aggregata

Pagina 79

- WINDOW FUNCTIONS: clausola OVER applicazioni

Pagina 80

- Non-aggregate: funzione ROW_NUMBER()

Pagina 81

- Non-aggregate: funzione RANK()
- Non-aggregate: funzione RANK() su PARTITION + ordinamento

Pagina 82

- Non-aggregate: funzione DENSE_RANK() su PARTITION + ordinamento
- Non-aggregate: RANK multipli con ALIAS

Pagina 83

- Non-aggregate: LAG(AttributoDaConsiderarePerSpostamento, ValoreSpostamento)

INDICE PISTOLESI-SQL

Pagina 85

- Non-aggregate: LEAD(AttributoDaConsiderarePerSpostamento, ValoreSpostamento)
- Non-aggregate: CUME_DIST() + WINDOW

Pagina 86

- Gestione FRAME
- Non-aggregate: FIRST_VALUE(ValoreDaTrovare) su DEFAULT FRAME

Pagina 88

- Non-aggregate: LAST_VALUE(ValoreDaTrovare) su FRAME

Pagina 89

- Aggregate: AVG(Valore) + PRECEDING/FOLLOWING su FRAME

Pagina 90

- Aggregate: SUM(Valore) e COUNT(*) su FRAME

Pagina 91

- Aggregate: SUM(Valore) e COUNT(*) su FRAME temporale con dichiarazione di RANGE

Pagina 92

- PIVOTING STATICO
- Funzione GROUP_CONCAT(NomeAttributoDaConcatenare)

Pagina 93

- GROUP_CONCAT(NomeAttributoDaConcatenare) in SELECT
- PIVOTING DINAMICO

Da pagina 95 a pagina 128

- Esercizi riepilogativi dell'intero programma

PISTOLESI-SQL

Query in SQL: SELECT, FROM, WHERE

Indicare cognome e nome delle persone di età inferiore a 40 anni

```
SELECT Cognome, Nome  
FROM Persona  
WHERE Età < 40;
```

Tutti gli attributi: SELECT *

Indicare tutte le informazioni delle persone di età inferiore a 40 anni

```
SELECT *  
FROM Persona  
WHERE Età < 40;
```

Operatore logico: AND

Indicare il codice fiscale delle persone di età inferiore a 40 anni il cui cognome è Lepre

```
SELECT CodFiscale  
FROM Persona  
WHERE Età < 40  
      AND Cognome = 'Lepre';
```

Contemporaneamente!

→ Ciò implica che si devono verificare contemporaneamente tutte e due le condizioni, per rendere la WHERE verificata

Operatore logico: OR

Indicare codice fiscale ed età delle persone il cui cognome è Nutrie o il cui nome è Maddalena

```
SELECT CodFiscale, Età  
FROM Persona  
WHERE Cognome = 'Nutrie'  
      OR Nome = 'Maddalena';
```

Oppure!

→ Ciò implica che si deve verificare una delle due condizioni, per rendere la WHERE verificata

PISTOLESI-SQL

Intervalli di valori: BETWEEN, >= e <=

Indicare cognome e nome delle persone di età compresa fra 45 e 60 anni

```
SELECT Cognome, Nome  
FROM Persona  
WHERE Età BETWEEN 45  
      AND 60;
```

```
SELECT Cognome, Nome  
FROM Persona  
WHERE Età >= 45  
      AND Età <= 60;
```

BETWEEN considera l'intervallo chiuso di valori, quindi stessa cosa tra >= e <=

Eliminazione duplicati: DISTINCT

Indicare i cognomi delle persone di età almeno pari a 38 anni

```
SELECT DISTINCT Cognome  
FROM Persona  
WHERE Età >= 38;
```

- DISTINCT elimina i cognomi duplicati restituiti dalla query, inserendo il valore che era duplicato soltanto una volta
- DISTINCT su più attributi → Agisce su tutti gli attributi proiettati

Valori NULL: IS NOT NULL

Indicare matricola e data di laurea degli studenti laureati immatricolati fra l'anno 2001 e l'anno 2005

```
SELECT Matricola, DataLaurea  
FROM Studente  
WHERE DataLaurea IS NOT NULL  
      AND Datascrizione BETWEEN '2001-01-01'  
            AND '2005-12-31';
```

Valori NULL: IS NULL

Indicare la matricola degli studenti non ancora laureati

```
SELECT Matricola  
FROM Studente  
WHERE DataLaurea IS NULL;
```

PISTOLESI-SQL

Tipi di dato data

Tipo	Formato
DATE	YYYY-MM-DD
TIMESTAMP	YYYY-MM-DD HH:MM:SS

Alcuni formati utili

Formato	Descrizione
%Y	anno (4 cifre)
%y	anno (2 cifre)
%M	nome del mese
%m	numero del mese (2 cifre)
%d	giorno del mese (00-31)
%W	nome del giorno
%w	giorno della settimana {0,...,6}
%T	orario (hh:mm:ss)

Formattazione delle date:

DATE_FORMAT(AttributoDaFormattare, 'Formato')

Indicare matricola e data di laurea (nel formato ‘dd|mm|yyyy, nome_giorno’) degli studenti iscritti prima del 2005

```
SELECT Matricola,  
       DATE_FORMAT(DataLaurea, '%d|%m|%Y, %W')  
FROM Studente  
WHERE Dataiscrizione < '2005-01-01';
```

- Gli apici nella DATE_FORMAT ‘ ’ solo per la formattazione e, il tipo di formato è dato nel testo tranne il valore percentuale
- Gli apici nella data (nel WHERE in questo caso) si mettono quando si esplicita una data

Giorno / Mese / Anno: DAY / MONTH / YEAR (Attributo)

Indicare matricola e mese di laurea degli studenti immatricolati dopo il 2000

```
SELECT Matricola,  
       MONTH(DataLaurea)  
FROM Studente  
WHERE DataLaurea IS NOT NULL  
      AND YEAR(Dataiscrizione) > 2000;
```

PISTOLESI-SQL

**Giorno / Mese / Anno, con variabile di sistema:
DAY / MONTH / YEAR (CURRENT_DATE)**

Indicare il cognome degli studenti che si sono laureati cinque anni fa

```
SELECT DISTINCT Cognome  
FROM Studente  
WHERE DataLaurea IS NOT NULL  
      AND YEAR(DataLaurea) = YEAR(CURRENT_DATE) - 5;
```

Durata in giorni: DATEDIFF(DataPiùRecente,DataPiùRemota)

Indicare matricola e da quanti giorni risultavano iscritti gli studenti, ad oggi laureati, che non si erano ancora laureati il 15 Luglio 2005

```
SELECT Matricola,  
      DATEDIFF('2005-07-15', Datalscrizione)  
FROM Studente  
WHERE Datalscrizione < '2005-07-15'  
      AND DataLaurea > '2005-07-15';
```

- Nella verifica della condizione i NULL non si considerano, perché il confronto con il valore inesistente ci restituisce falso
- Per rafforzare la dicitura nel testo dell'esercizio "...ad oggi laureati" la seconda condizione del WHERE potremmo scriverla così:
AND (
 DataLaurea > '2005-07-15'
 AND DataLaurea <= CURRENT_DATE
);

Durata in mesi: PERIOD_DIFF(DataPiùRecente,DataPiùRemota)

Indicare matricola e da quanti mesi risultavano iscritti gli studenti, ad oggi laureati, che non si erano ancora laureati il 15 Luglio 2005

```
SELECT Matricola,  
      PERIOD_DIFF(  
          DATE_FORMAT(DataLaurea, '%Y%m'),  
          DATE_FORMAT(Datalscrizione, '%Y%m')  
      )  
FROM Studente  
WHERE Datalscrizione < '2005-07-15'  
      AND DataLaurea > '2005-07-15';
```

- Nella PERIOD_DIFF dentro le DATE_FORMAT, considero %Y%m, anno e mese per fare la differenza → Faccio così perché se devo fare un confronto fra mesi, mi serve anche sapere a quale anno un mese può riferirsi
- Stesso discorso di sopra sui valori NULL e sulla dicitura 'ad oggi laureati'

PISTOLESI-SQL

Sintassi INTERVAL

Esprime un **intervallo di tempo** espresso in giorni, mesi o anni

INTERVAL Numerolntero [YEAR | MONTH | DAY]

Sommare / Sottrarre intervalli di tempo con funzioni: **DATE_ADD / DATE SUB (PrimaData, LassodiTempo-INTERVAL)**

Indicare la matricola e il mese di iscrizione degli studenti che si sono laureati dopo cinque anni esatti dal giorno dell'iscrizione

```
SELECT Matricola,  
       MONTH(Datalscrizione)  
FROM Studente  
WHERE DataLaurea IS NOT NULL  
      AND DataLaurea = DATE_ADD(Datalscrizione, INTERVAL 5 YEAR);
```

Sommare / Sottrarre intervalli di tempo direttamente con + e - usando INTERVAL

Indicare la matricola e il mese di iscrizione degli studenti che si sono laureati dopo cinque anni esatti dal giorno dell'iscrizione

```
SELECT Matricola,  
       MONTH(Datalscrizione)  
FROM Studente  
WHERE DataLaurea IS NOT NULL  
      AND DataLaurea = Datalscrizione + INTERVAL 5 YEAR;
```

OSS: La somma e la sottrazione si fanno sempre con una data e un INTERVAL, non si fa mai fra due date!

Funzioni di utilità sulle date

Funzione	Risultato
dayname ()	nome del giorno
monthname ()	nome del mese
dayofweek ()	giorno della settimana in {1,...,7}
weekday ()	giorno della settimana in {0,...,6}
last_day ()	ultimo giorno del mese della data
dayofyear ()	mese (2 cifre)
weekofyear ()	numero della settimana {0,...,53}
yearweek ()	anno e settimana

PISTOLESI-SQL

Conteggio con ridenominazione: COUNT(*) AS

Indicare il numero di visite effettuate in data 1° Marzo 2013

```
SELECT COUNT(*) AS VisitePrimoMarzo  
FROM Visita  
WHERE Data = '2013-03-01';
```

- Conta tutte le visite di questo risultato restituendomi un solo numero, cioè il conteggio
- I valori NULLI nel conteggio non vengono considerati

Conteggio su attributo/i: COUNT(DISTINCT)

Indicare il numero di pazienti visitati nel mese di Marzo 2013

```
SELECT COUNT(DISTINCT Paziente)  
FROM Visita  
WHERE MONTH(Data) = '03'  
      AND YEAR(Data) = '2013';
```

Evito di contare più di una volta lo stesso valore dell'attributo
(in questo caso lo stesso paziente) e si prende solo una volta il doppione

Somma: SUM

Supponendo che Umberto Manzi sia il marito di Antonella Lepre, calcolare il reddito totale della famiglia Manzi.

```
SELECT SUM(Reddotto) AS ReddottoTotale  
FROM Paziente  
WHERE(  
      Cognome = 'Lepre'  
      AND Nome = 'Antonella'  
    )  
    OR  
    (  
      Cognome = 'Manzi'  
      AND Nome = 'Umberto'  
    );
```

Metto OR e non AND tra i due gruppi di condizioni, perché altrimenti verificherei che una persona si chiami contemporaneamente Umberto Manzi e Antonella Lepre

PISTOLESI-SQL

Media: AVG

Calcolare il reddito medio dei pazienti nati dopo il 1950

```
SELECT AVG(Reddito) AS RedditoMedio  
FROM Paziente  
WHERE DataNascita > '1950-12-31';
```

Minimo e massimo: MIN / MAX

Ricavare il reddito massimo/minimo fra quelli di tutti i pazienti

- ```
SELECT MAX(Reddito)
FROM Paziente;
```
- ```
SELECT MIN(Reddito)  
FROM Paziente;
```

WHERE non presente perché non voglio applicare nessuna condizione dato che voglio considerare tutti i pazienti

QUANDO SI UTILIZZANO GLI OPERATORI DI AGGREGAZIONE, ACCANTO A QUESTI NON SI PUÒ METTERE ALCUN VALORE CHE NON SIA AGGREGATO ANCH'ESSO, VISTO CHE NON SAREBBERO COLLEGATI A QUELLO AGGREGATO

INNER JOIN (con ALIAS)

Indicare nome e cognome dei medici che hanno effettuato almeno una visita

```
SELECT DISTINCT M.Nome, M.Cognome  
FROM Visita V  
    INNER JOIN  
        Medico M ON V.Medico = M.Matricola;
```

L'attributo V.Medico deve essere uguale a M.Matricola

NATURAL JOIN

Indicare nome e cognome dei medici che hanno effettuato almeno una visita

```
SELECT DISTINCT M.Nome, M.Cognome  
FROM Visita V  
    NATURAL JOIN  
        Medico M;
```

Combina i record della prima tabella con quelli della seconda tabella, aventi valori uguali sugli attributi omonimi (in questo caso come prima V.Medico = M.Matricola)

NEL CASO AVESSI ATTRIBUTI CON LO STESSO NOME SULLE DUE TABELLE DIVERSE, QUESTI ATTRIBUTI NEL RESULT SET, COMPARIREBBERO UNA VOLTA SOLA (PER IL NATURAL JOIN), NELL'INNER JOIN INVECE SI RIPETONO

PISTOLESI-SQL

Prodotto cartesiano: CROSS JOIN

Indicare il numero di pazienti aventi nome uguale ad almeno un medico della clinica

```
SELECT COUNT(DISTINCT P.CodFiscale)
FROM Paziente P
    CROSS JOIN
    Medico M
WHERE P.Nome = M.Nome;
```

- Metto il DISTINCT perché un paziente può avere nome uguale a più medici della clinica, e a me basta che sia uguale ad almeno un medico.
- Se non lo mettessi, conterei più volte lo stesso paziente perché: quando trovo che un paziente ha lo stesso nome di un medico faccio +1 al conteggio e, per il restante dei medici, appena trovo un altro medico che ha lo stesso nome del paziente, riconto un'altra volta il paziente, facendo un altro +1 al conteggio, e così via. Il tutto che mi produce il conto sbagliato su quel paziente!

LEFT OUTER JOIN

Indicare le visite effettuate da medici che non lavorano più presso la clinica

```
SELECT V.*
FROM Visita V
    LEFT OUTER JOIN
    Medico M ON V.Medico = M.Matricola
WHERE M.Matricola IS NULL;
```

- Il fatto che un medico non lavori più in una clinica, si ha sulla sua non presenza nella tabella delle visite
- Questo tipo di JOIN mi mantiene la tabella di SX (Visita) mettendomi NULL alla tabella di DX (Medico), che sono i record che non fanno JOIN, cioè quelli che non verificano la condizione V.Medico = M.Matricola
- In questo caso seleziono quelli la cui matricola del medico (dentro la tabella Medico) è NULL e, facendo così rappresento quelli che non ci sono come medici, di conseguenza che non lavorano più
- Proietto V.*, per avere tutte le informazioni diverse da NULL, perché se proiettassi * avrei anche i NULL che ci sono nella tabella di destra del Medico

PISTOLESI-SQL

RIGHT OUTER JOIN

Indicare matricola, cognome e specializzazione dei medici che non hanno ancora effettuato visite

```
SELECT DISTINCT M.Matricola, M.Cognome, M.Specializzazione  
FROM Visita V
```

RIGHT OUTER JOIN

Medico M ON V.Medico = M.Matricola

```
WHERE V.Medico IS NULL;
```

- Questo tipo di JOIN mi mantiene la tabella di DX (Medico) mettendomi NULL alla tabella di SX (Visita) che sono i record che non fanno JOIN, cioè quelli che non verificano la condizione V.Medico = M.Matricola
- In questo caso seleziono quelli la cui matricola del medico (dentro la tabella Visita) è NULL e, facendo così rappresento quelli che non ci sono nelle visite, di conseguenza che non hanno fatto visite (simile all'esercizio precedente)
- DISTINCT mi serve perché due record con la stessa Matricola, Cognome e Specializzazione, si riferiscono a due visite effettuate dallo stesso medico

Query con INNER JOIN e condizioni sui record

Indicare nome, cognome e specializzazione dei medici che hanno visitato almeno un paziente il giorno 1° Marzo 2013

```
SELECT DISTINCT M.Nome, M.Cognome, M.Specializzazione  
FROM Medico M  
INNER JOIN  
Visita V ON M.Matricola = V.Medico  
WHERE V.Data = '2013-03-01';
```

DISTINCT mi serve perché due record con la stessa Matricola, Cognome e Specializzazione, si riferiscono a due visite effettuate dallo stesso medico lo stesso giorno; quindi, se non lo mettessi avrei dei duplicati

JOIN Multiplo

Indicare nome e cognome dei pazienti visitati dal dott. Rino Neri

```
SELECT DISTINCT P.Nome, P.Cognome  
FROM Paziente P  
INNER JOIN  
Visita V ON P.CodFiscale = V.Paziente  
INNER JOIN  
Medico M ON V.Medico = M.Matricola  
WHERE M.Nome = 'Rino'  
AND M.Cognome = 'Neri';
```

DISTINCT serve perché lo stesso paziente può avere fatto più visite col dottor Rino Neri

PISTOLESI-SQL

Ridenominazione nel FROM

Indicare il numero di volte che la dottoressa Marta Rossi ha visitato il paziente Umberto Manzi

```
SELECT COUNT(*)  
FROM Paziente P  
    INNER JOIN  
        Visita V ON P.CodFiscale = V.Paziente  
    INNER JOIN  
        Medico M ON V.Medico = M.Matricola  
WHERE(  
    M.Nome = 'Marta'  
    AND M.Cognome = 'Rossi'  
)  
AND  
(  
    P.Nome = 'Umberto'  
    AND P.Cognome = 'Manzi'  
);
```

SELF JOIN

Indicare il codice fiscale dei pazienti che sono stati visitati più di una volta da almeno un medico della clinica, nel mese corrente

```
SELECT DISTINCT V1.Paziente  
FROM Visita V1  
    INNER JOIN  
        Visita V2 ON (  
            V2.Medico = V1.Medico  
            AND V2.Paziente = V1.Paziente  
            AND V2.Data <> V1.Data  
        )  
WHERE MONTH(V1.Data) = MONTH(CURRENT_DATE)  
    AND YEAR(V1.Data) = YEAR(CURRENT_DATE) //1  
    AND MONTH(V2.Data) = MONTH(CURRENT_DATE)  
    AND YEAR(V2.Data) = YEAR(CURRENT_DATE); //2 //3
```

- DISTINCT mi serve per non avere lo stesso codice fiscale del paziente se lo ritrovo tra le visite dei medici della clinica
- V1 e V2 sono due variabili che puntano ad istanze della stessa tabella (Visita)
- Vado ad affiancare a ciascuna visita, tutte le visite effettuate da ciascun paziente con lo stesso medico in date diverse //1 → Quindi se ho anche solo una visita dello stesso paziente con lo stesso medico in due date diverse ciò vuol dire che ho più di una visita, e di conseguenza va nel risultato
- Essendo che mi interessa solo il mese corrente e non l'anno, le righe a //2 e a //3 si potrebbero omettere

PISTOLESI-SQL

SELF JOIN con USING

Indicare il codice fiscale dei pazienti che sono stati visitati più di una volta nel mese corrente

```
SELECT DISTINCT V1.Paziente
FROM Visita V1
INNER JOIN
Visita V2 USING(Paziente)
WHERE MONTH(V1.Data) = MONTH(CURRENT_DATE)
AND MONTH(V2.Data) = MONTH(CURRENT_DATE)
AND (
(
V2.Data = V1.Data
AND V2.Medico <> V1.Medico
)
OR
V2.Data <> V1.Data
);
```

- USING è fatto sulla chiave primaria di entrambi quindi ($V1.Paziente=V2.Paziente$)
- Il DISTINCT lo metto perché quel paziente in questo mese potrebbe essere stato visitato più volte e quindi, andrei in contro a più di una sua presenza (duplicato)
- Più di una visita si considera dal fatto se ci sono visite di due medici differenti oppure se ci sono visite in due date diverse (le due condizioni tra l'OR)

DERIVED TABLE

Indicare la matricola dei medici che non hanno mai visitato di giovedì

```
SELECT DISTINCT V1.Medico
FROM Visita V1
LEFT OUTER JOIN
(
SELECT V2.Medico
FROM Visita V2
WHERE DAYOFWEEK(V2.Data) = 4
) AS D
USING(Medico)
WHERE D.Medico IS NULL;
```

//1

//2

//3

DERIVED TABLE →

Si
trova
nel
FROM

- USING equivarrebbe a fare $V1.Medico=D.Medico$
- La DERIVED TABLE mi dà i medici che hanno fatto visita il giovedì
- Prendo le matricole dei medici togliendo i duplicati (//1), faccio un SELF JOIN esterno SX sulla DERIVED TABLE (//2), quindi quelle che non fanno join hanno NULL a DX → Quindi sono i medici che non hanno fatto visite di giovedì, perché quelli che hanno fatto visite di giovedì fanno JOIN !
- Li proietto (//1) sfruttando la condizione nel WHERE che hanno NULL alla matricola (//3) e quindi prendo quelli che non hanno mai visitato di giovedì

PISTOLESI-SQL

NON CORRELATED SUBQUERY (con IN)

Indicare nome, cognome e parcella degli ortopedici che hanno effettuato almeno una visita nell'anno 2013

```
SELECT M.Nome, M.Cognome, M.Parcella  
FROM Medico M  
WHERE M.Specializzazione = 'Ortopedia'  
AND M.Matricola IN  
(  
    SELECT V.Medico  
    FROM Visita V  
    WHERE YEAR(V.Data) = 2013  
);
```

NON
CORRELATED
SUBQUERY

Si trova
nel
WHERE

- I record della SUBQUERY NON dipendono dalla query esterna
- La SUBQUERY restituisce un RESULT SET dove ci sono le visite di un medico nell'anno 2013
- La query esterna pesca dal RESULT SET, controllando, se la matricola del medico è tra le visite del medico fatte nel 2013 → Tramite IN
- E se questa matricola c'è, allora ciò vuol dire che il medico ha fatto almeno una visita

NON CORRELATED SUBQUERY (con NOT IN)

Indicare i cognomi dei pazienti che non appartengono anche a un medico

```
SELECT DISTINCT P.Cognome  
FROM Paziente P  
WHERE P.Cognome NOT IN  
(  
    SELECT M.Cognome  
    FROM Medico M  
);
```

NON CORRELATED
SUBQUERY

Considero i cognomi dei pazienti (senza duplicati) il cui cognome non sta fra
→ I cognomi dei medici (SUBQUERY)

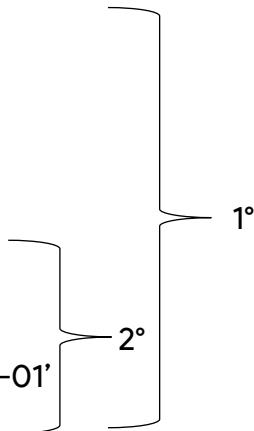
**DATA UNA QUERY CON SUBQUERY È SEMPRE POSSIBILE PASSARE ALLA
VERSIONE BASATA SU JOIN, E VICEVERSA**

PISTOLESI-SQL

Annidamento multiplo di NON CORRELATED SUBQUERY con versione JOIN-equivalente

Indicare nome, cognome e specializzazione dei medici che hanno effettuato visite eccetto che il giorno 1° Marzo 2013

```
SELECT M.Nome, M.Cognome, M.Specializzazione  
FROM Medico M  
WHERE M.Matricola IN  
(  
    SELECT V.Medico  
    FROM Visita V  
)  
AND M.Matricola NOT IN  
(  
    SELECT V.Medico  
    FROM Visita V  
    WHERE V.Data='2013-03-01'  
);
```



1°) Considero (IN) le matricole dei medici che sono presenti tra le matricole delle visite
→ Quindi i medici che hanno visitato.

Di queste matricole non considero (NOT IN) 2°) Le matricole che hanno la data della visita uguale al 1° Marzo 2013

→ Quindi considero i medici che non hanno visitato il 1° Marzo 2013

Versione JOIN-equivalente

```
SELECT M.Nome, M.Cognome, M.Specializzazione  
FROM Visita V  
    INNER JOIN  
        Medico M ON V.Medico=M.Matricola  
    LEFT OUTER JOIN  
    (  
        SELECT *  
        FROM Visita  
        WHERE Data='2013-03-01'  
    ) AS D  
        ON V.Medico=D.Medico  
WHERE D.Medico IS NULL;
```

- L'INNER JOIN mi considera i medici che hanno fatto visite
- Il LEFT OUTER JOIN, a questi medici che hanno fatto visite, affianca una tabella che contiene le visite fatte da questi medici in data 1° Marzo 2013, oppure pone a NULL quelli che non hanno fatto visite il 1° Marzo 2013
- Alla fine, prenderò i medici che hanno NULL sulla DERIVED TABLE, che saranno quelli che hanno fatto visite tranne che il 1° Marzo 2013

PISTOLESI-SQL

NON CORRELATED SUBQUERY scalare con operatori di confronto (>, <, =, >=, <=)

Indicare il numero degli otorini aventi parcella più alta della media delle parcelle dei medici della loro specializzazione.

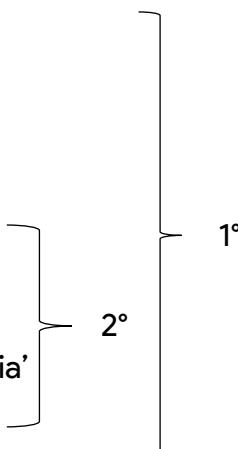
```
SELECT COUNT(*)  
FROM Medico M1  
WHERE M1.Specializzazione = 'Otorinolaringoiatria'  
AND M1.Parcella >  
(  
    SELECT AVG(M2.Parcella)  
    FROM Medico M2  
    WHERE M2.Specializzazione = 'Otorinolaringoiatria'  
);
```

Conto gli otorini la cui parcella è > della → Media delle parcelle degli otorini (SUBQUERY)

Risoluzione mista NON CORRELATED SUBQUERY-JOIN

Indicare il numero di pazienti di età superiore a 50 anni visitati dai cardiologi di Pisa avendo parcella inferiore alla media delle parcelle dei cardiologi.

```
SELECT COUNT(DISTINCT CodFiscale)  
FROM Paziente P INNER JOIN  
    Visita V ON P.CodFiscale = V.Paziente  
WHERE CURRENT_DATE > P.DataNascita + INTERVAL 50 YEAR //1  
AND V.Medico IN  
(  
    SELECT M.Matricola  
    FROM Medico M  
    WHERE M.Specializzazione = 'Cardiologia'  
    AND M.Citta = 'Pisa'  
    AND M.Parcella <  
(  
        SELECT AVG(M2.Parcella)  
        FROM Medico M2  
        WHERE M2.Specializzazione = 'Cardiologia'  
    )  
);
```



- Conto il numero di pazienti (senza duplicati) che hanno fatto una visita la cui età è: maggiore di 50 anni (ad oggi) → Oggi è maggiore di (data nascita + 50 anni) quindi oggi è almeno 51 → Di conseguenza supera i 50 anni (//1)
- Di questi pazienti, (1°) considero i medici che li hanno visitati e vedo se sono (IN): tra i medici cardiologi di Pisa, la cui parcella è minore della: (2°) media delle parcelle dei cardiologi
- La SUBQUERY 2° è annidata dentro la 1°

PISTOLESI-SQL

CORRELATED SUBQUERY (IN/NOT IN)

Indicare la matricola dei medici che hanno visitato per la prima volta almeno un paziente nel mese di Ottobre 2013.

```
SELECT DISTINCT V1.Medico  
FROM Visita V1  
WHERE YEAR(V1.Data) = 2013  
    AND MONTH(V1.Data) = 10  
    AND V1.Paziente NOT IN  
(  
    SELECT V2.Paziente  
    FROM Visita V2  
    WHERE V2.Medico = V1.Medico  
        AND V2.Data < V1.Data  
);
```

Viene eseguita per ogni tupla esterna → Quindi per ogni medico che ha visitato nel mese di Ottobre del 2013

CORRELATED SUBQUERY

Proietto la matricola dei medici (senza duplicati) che hanno fatto visite nel mese di ottobre 2013 il cui paziente non lo trovo tra:

(SUBQUERY) I pazienti che hanno fatto una visita con quel medico lì, la cui data della visita, è precedente alla visita fatta dal paziente iniziale → Se con quel medico ho una visita precedente per quel paziente → Allora non è una prima visita, e quindi deve valere il NOT IN per fare in modo che non ci sia il paziente in questa condizione

CORRELATED SUBQUERY scalare con operatori di confronto (>, <, =, >=, <=)

Indicare nome e cognome dei pazienti che sono stati visitati due volte dal dottor Paolo Verdi.

```
SELECT P.Nome, P.Cognome  
FROM Paziente P  
WHERE 2 =  
(  
    SELECT COUNT(*) AS VisiteDottorVerdi  
    FROM Visita V  
    INNER JOIN  
        Medico M ON V.Medico = M.Matricola  
    WHERE M.Nome = 'Paolo'  
        AND M.Cognome = 'Verdi'  
        AND V.Paziente = P.CodiceFiscale  
);
```

CORRELATED SUBQUERY

- Nella SUBQUERY conto le visite che il Paziente ha fatto con il dottor Paolo Verdi, e questo paziente proveniente dalla query esterna → Faccio questa operazione per tutti i pazienti della query esterna (OSS. V.Paziente=P.CodiceFiscale)
- Nella query esterna proietto Nome e Cognome dei pazienti che hanno fatto 2 visite col dottore Paolo Verdi (CORRELATED SUBQUERY)

PISTOLESI-SQL

CORRELATED SUBQUERY nel SELECT

Considerato ciascun paziente di sesso maschile, indicarne il codice fiscale e il numero di visite effettuate.

```
SELECT P.CodiceFiscale,
      (
        SELECT COUNT(*)
        FROM Visita V
        WHERE V.Paziente = P.CodFiscale
      ) AS TotaleVisite
  FROM Paziente P
 WHERE P.Sesso = 'M';
```



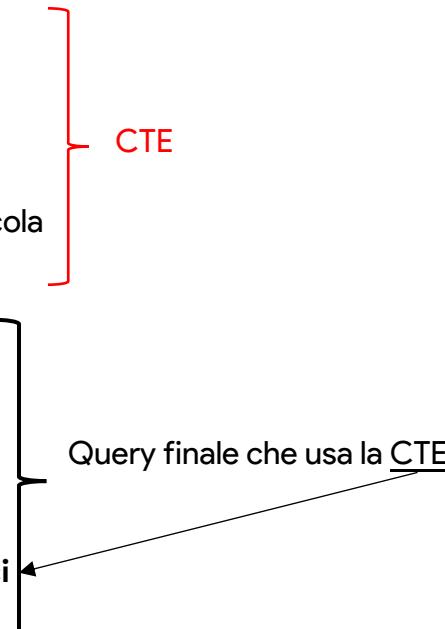
- La SUBQUERY fa il conteggio delle visite per ogni paziente di sesso maschile (proveniente dalla query esterna)
- È utile che la SUBQUERY nel select abbia una ridenominazione
- 'TotaleVisite' cambia ogni volta che cambio paziente

UNA CORRELATED SUBQUERY È ESEGUITA PER OGNI TUPLA DELLA QUERY ESTERNA, E SE VIENE INSERITA NEL SELECT, QUESTA DEVE ESSERE SCALARE!

CTE

Indicare il numero di pazienti di Siena, mai visitati da ortopedici

```
WITH pazienti_visitati_ortopedici AS
  (
    SELECT V.Paziente
    FROM Visita V
    INNER JOIN
      Medico M ON V.Medico = M.Matricola
    WHERE M.Specializzazione = 'Ortopedia'
  )
  SELECT COUNT(*)
  FROM Paziente P
  WHERE P.Citta = 'Siena'
    AND P.CodFiscale NOT IN
      (
        SELECT *
        FROM pazienti_visitati_ortopedici
      );
```



- CTE = Cerco i pazienti che sono stati visitati da ortopedici
- Query finale che usa la CTE = Conto i pazienti di Siena, il cui codice fiscale non è tra → I pazienti che sono stati visitati da ortopedici (risultato CTE)
- La tabella "pazienti_visitati_ortopedici" è l'identificatore della CTE
- OSS: Alla fine del WHERE dentro la CTE non è presente il punto e virgola!

PISTOLESI-SQL

CTE multiple

Indicare il numero di pazienti di età superiore a 50 anni visitati dai cardiologi di Pisa aventi parcella inferiore alla media delle parcelle dei cardiologi.

```

WITH parcella_media_cardio AS
(
    SELECT AVG(M2.Parcella)
    FROM Medico M2
    WHERE M2.Specializzazione = 'Cardiologia'
)
, cardiologi_pisa AS
(
    SELECT M.Matricola AS Medico, M.Parcella
    FROM Medico M
    WHERE M.Specializzazione = 'Cardiologia'
        AND M.Citta = 'Pisa'
)
SELECT COUNT(DISTINCT CodFiscale)
FROM Paziente P
    INNER JOIN
    Visita V ON P.CodFiscale = V.Paziente
    NATURAL JOIN
    cardiologi_pisa CP
WHERE CURRENT_DATE > P.DataNascita + INTERVAL 50 YEAR → SOTTOLINEATURA
    AND CP.Parcella >
    (
        SELECT *
        FROM parcella_media_cardio
    );

```

SOTTOLINEATURA

CTE1 (SOTTOLINEATURA)

CTE2 (SOTTOLINEATURA)

Pazienti visitati da cardiologi di Pisa
→ NATURAL JOIN su
V.Medico=CP.Medico (Matricola
rinominata in Medico dentro la CTE)

L'ultimo AND rappresenta le parcelle dei medici cardiologi di Pisa, che hanno ciascuno, la parcella maggiore, della media delle parcelle dei cardiologi

Sintassi di WITH per CTEs (Common Table Expression)

```
WITH  
    name1 AS (query1)  
    ,  
    name2 AS (query2)  
    ,  
    ...  
    ,  
    nameN AS (queryN)  
query finale;
```

LE CTE SONO RESULT SET CON IDENTIFICATORE, CHE COSTRUISCONO RISULTATI INTERMEDI

PISTOLESI-SQL

Raggruppamento GROUP BY

Indicare la parcella media dei medici di ciascuna specializzazione

```
SELECT Specializzazione,  
       AVG(Parcella) AS ParcellaMedia  
  FROM Medico  
 GROUP BY Specializzazione;
```

L'attributo di raggruppamento va specificato anche nella proiezione! (Specializzazione)

L'ATTRIBUTO DI RAGGRUPPAMENTO DEVE COMPARIRE NELLA PROIEZIONE, A MENO CHE NON SIA UN AGGREGATO

GROUP BY dentro una DERIVED TABLE

Per ogni specializzazione medica, indicarne il nome, la parcella minima e il cognome del medico a cui appartiene

→ OSS “D.”

```
SELECT M.Specializzazione, D.ParcellaMinima, M.Cognome  
  FROM Medico M  
    NATURAL JOIN  
    (  
      SELECT Specializzazione,  
             MIN(Parcella) AS ParcellaMinima  
        FROM Medico  
 GROUP BY Specializzazione  
    ) AS D
```

DERIVED TABLE

WHERE M.Parcella = D.ParcellaMinima;

- La DERIVED TABLE mi dà le parcelle minime dei medici raggruppate per specializzazione
- La query esterna mi proietta le parcelle minime ottenute dalla DERIVED TABLE e il cognome dei medici che hanno la parcella minima
- Il NATURAL JOIN viene fatto sull'unico attributo omonimo Specializzazione
- Il fatto che sia sicuro di scegliere i medici giusti, cioè quelli con la parcella minima, lo ottengo dal WHERE → Perché scelgo i medici (quelli esterni) che hanno parcella uguale alla parcella minima dei medici (derivante dalla DERIVED TABLE) e di conseguenza i medici saranno proprio questi

Condizioni sui gruppi: clausola HAVING

Indicare le specializzazioni della clinica con più di due medici

```
SELECT Specializzazione  
  FROM Medico  
 GROUP BY Specializzazione  
 HAVING COUNT(*) > 2;
```

Dopo aver fatto il raggruppamento, salvo i gruppi aventi un conto di righe >2

PISTOLESI-SQL

Condizioni sui gruppi + (SUBQUERY scalare + DERIVED TABLE)

Indicare le specializzazioni con la più alta parcella media

```
SELECT M.Specializzazione  
FROM Medico M  
GROUP BY M.Specializzazione  
HAVING AVG(M.Parcella) =
```

SUBQUERY SCALARE

```
(  
    SELECT MAX(D.MediaParcelle)  
    FROM (  
        SELECT M2.Specializzazione,  
              AVG(M2.Parcella) AS MediaParcelle  
        FROM Medico M2  
        GROUP BY M2.Specializzazione  
    )AS D  
)
```

DERIVED TABLE

- Proietto le specializzazioni dei medici (raggruppandole) e prendendo quelle, che hanno la parcella media dei medici = alla parcella massima tra le parcelle medie dei medici, raggruppate per specializzazione
- La parcella massima tra le parcelle medie dei medici, raggruppate per specializzazione me le restituisce la SUBQUERY (che è di tipo non correlata)
- Le parcelle medie dei medici, raggruppate per specializzazione sono dentro la DERIVED TABLE

OSS: La proiezione iniziale, considera anche i pari merito

Condizioni sui gruppi + condizione sui record

Indicare le specializzazioni con più di due medici di Pisa

```
SELECT Specializzazione  
FROM Medico  
WHERE Citta = 'Pisa'  
GROUP BY Specializzazione  
HAVING COUNT(*) > 2;
```

Per ogni Specializzazione di Pisa (condizione sui record) conto se ci sono gruppi da più di 2 elementi (condizione sui gruppi)

LE CONDIZIONI NEL WHERE SONO APPLICATE AI RECORD PRIMA DEL RAGGRUPPAMENTO; LE CONDIZIONI NELL'HAVING SONO APPLICATE AI GRUPPI DOPO IL RAGGRUPPAMENTO

PISTOLESI-SQL

Raggruppamento su più attributi

Considerati i soli pazienti di Pisa, indicarne nome e cognome, e la spesa sostenuta per le visite di ciascuna specializzazione, nel triennio 2008-2010

```
SELECT M.Specializzazione,  
       P.Nome, P.Cognome,  
       SUM(M.Parcella)  
  FROM Visita V  
        INNER JOIN  
      Paziente P ON V.Paziente = P.CodFiscale  
        INNER JOIN  
      Medico M ON V.Medico = M.Matricola  
 WHERE P.Citta = 'Pisa'  
       AND YEAR(V.Data) BETWEEN 2008  
                           AND 2010  
GROUP BY M.Specializzazione,  
        P.CodFiscale, P.Nome, P.Cognome;
```

- “Considerati i soli pazienti di Pisa” → Sotto sotto, dopo aver visto anche il resto del testo, implica che dovrò andare a considerare ciascun paziente; quindi, ci sarà un raggruppamento anche sul paziente!
- Il raggruppamento lo faccio su P.CodFiscale, P.Nome, P.Cognome, proiettando P.Nome, P.Cognome → In questo caso mi sarebbe andato bene raggruppare solo per P.CodFiscale, visto che c’è una *dipendenza funzionale* di chiave tra (CodFiscale→Nome,Cognome)
→ P.Nome, P.Cognome tra gli attributi di raggruppamento sono superflui
“...di Pisa...” e anche “...nel triennio 2009-2010” → Sono due condizioni sui record e vanno espressi prima del predicato di raggruppamento

TUTTI GLI ATTRIBUTI DI RAGGRUPPAMENTO DEVONO COMPARIRE NELLA PROIEZIONE, OPPURE ANCHE SOLO LA CHIAVE CHE INTERESSA TALI ATTRIBUTI IN QUANTO VI ESISTE UNA DIPENDENZA FUNZIONALE TRA LORO; INOLTRE VALE ANCHE QUA LO STESSO DISCORSO SUGLI AGGREGATI

PISTOLESI-SQL

EXISTS

Una visita di controllo è una visita in cui un medico visita un paziente già visitato precedentemente almeno una volta. Indicare medico, paziente e data delle visite di controllo del mese di Gennaio 2016

```
SELECT V1.Medico, V1.Paziente, V1.Data  
FROM Visita V1  
WHERE MONTH(V1.Data) = 1  
    AND YEAR(V1.Data) = 2016  
    AND EXISTS  
        (  
            SELECT *  
            FROM Visita V2  
            WHERE V2.Medico = V1.Medico  
                AND V2.Paziente = V1.Paziente  
                AND V2.Data < V1.Data  
        );
```

CORRELATED SUBQUERY

- Per le visite di controllo vedo se, dato un medico un paziente e una data, se ESISTE: (SUBQUERY) una visita con quel medico, con quel paziente e con una data precedente alla data della visita (query esterna) → Nel caso esistesse non è una prima visita, quindi è di controllo! → Quindi la prendo

NOT EXISTS

Indicare la matricola dei medici che hanno visitato per la prima volta almeno un paziente nel mese di Ottobre 2013

```
SELECT DISTINCT V1.Medico  
FROM Visita V1  
WHERE YEAR(V1.Data) = 2013  
    AND MONTH(V1.Data) = 10  
    AND NOT EXISTS  
        (  
            SELECT *  
            FROM Visita V2  
            WHERE V2.Medico = V1.Medico  
                AND V2.Paziente = V1.Paziente  
                AND V2.Data < V1.Data  
        );
```

CORRELATED SUBQUERY

- Per le visite di controllo, dato un medico un paziente e una data, vedo se NON ESISTE: (SUBQUERY) una visita di quel medico, con quel paziente, in una data precedente alla data della visita (query esterna)
→ Che sarebbe una visita di controllo
- Nel caso non esistesse una visita di controllo, allora è una prima visita e va nel

QUESTI DUE COSTRUTTI, SI USANO SOLO SU QUERY DI TIPO CORRELATO

PISTOLESI-SQL

Unione OR

Indicare la parcella media fra quella degli otorini e quella degli ortopedici

```
SELECT AVG(Parcella)
FROM Medico
WHERE Specializzazione = 'Otorinolaringoiatria'
      OR Specializzazione = 'Ortopedia';
```

Un medico è un otorino oppure un ortopedico → La 'e' è copulativa!

Unione senza duplicati UNION

I pazienti visitati dal dott. Verdi o dal dott. Rossi

```
SELECT Paziente
FROM Visita V
      INNER JOIN
            Medico M ON V.Medico = M.Matricola
WHERE M.Cognome = 'Verdi'
UNION
SELECT Paziente
FROM Visita V
      INNER JOIN
            Medico M ON V.Medico = M.Matricola
WHERE M.Cognome = 'Rossi';
```

- La prima query mi dà i pazienti visitati dal dott. Verdi, mentre la seconda dal dott. Rossi
- In questo caso mi vanno bene dal dott. Verdi, oppure dal dott. Rossi oppure anche da entrambi → Uso la UNION per questa cosa che mi prende anche l'intersezione
- I pazienti del risultato seguono l'ordine dell'unione

LA UNION ELIMINA AUTOMATICAMENTE I DUPLICATI

PISTOLESI-SQL

Unione con duplicati UNION ALL

Totale delle visite effettuate il lunedì dal dott. Verdi e il venerdì dal dott. Rossi

```
SELECT COUNT(*)  
FROM(  
    SELECT Paziente  
    FROM Visita V  
        INNER JOIN  
            Medico M ON V.Medico = M.Matricola  
    WHERE M.Cognome = 'Verdi'  
        AND DAYOFWEEK (V.Data) = 1  
UNION ALL  
    SELECT Paziente  
    FROM Visita V  
        INNER JOIN  
            Medico M ON V.Medico = M.Matricola  
    WHERE M.Cognome = 'Rossi'  
        AND DAYOFWEEK (V.Data) = 5  
) AS D;
```

- 1 equivale al lunedì, 5 al venerdì → Seguono i giorni della settimana
- UNION ALL mi mantiene i duplicati, perché in questo caso voglio il totale delle visite e non dei pazienti (di cui solitamente escluderei i duplicati)
- La prima query mi restituisce i pazienti visitati dal dottor Verdi il lunedì
- La seconda query mi restituisce i pazienti visitati dal dottor Rossi il venerdì
- Con la UNION ALL unisco (con i duplicati) i risultati delle due query e il tutto andrà nella DERIVED TABLE
- La query esterna mi fa il conteggio totale delle righe della DERIVED TABLE

LA UNION ALL MANIENE AUTOMATICAMENTE I DUPLICATI

PISTOLESI-SQL

Divisione: con doppio NOT EXISTS

Indicare i pazienti visitati da tutti i medici

```
SELECT P.CodFiscale  
FROM Paziente P } 1  
WHERE NOT EXISTS  
(  
    SELECT *  
    FROM Medico M } 2  
    WHERE NOT EXISTS  
        (  
            SELECT *  
            FROM Visita V  
            WHERE V.Medico = M.Matricola  
                AND V.Paziente = P.CodFiscale } 3  
        )  
);
```

1. "Prendi i pazienti per i quali NON ESISTONO"
2. "Medici, i quali NON HANNO"
3. "Fatto visite con quel paziente"

→ Quindi prendi i pazienti, per i quali ci sono medici che hanno fatto visite con quei pazienti

Divisione: con raggruppamento

Indicare i pazienti visitati da tutti i medici

```
SELECT V.Paziente  
FROM Visita V } 1  
GROUP BY V.Paziente  
HAVING COUNT(DISTINCT V.Medico) = } 2  
(  
    SELECT COUNT(*) } 3  
    FROM Medico  
);
```

1. Prendi i pazienti e raggruppali per paziente
2. Considerando il totale di quelli che li hanno visitati (senza duplicati) uguale al
3. Totale dei medici

→ Se valgono 2 e 3 vuol dire che sono tutti i medici che hanno fatto visite

Se il conteggio al punto 2 non è = ma è !=, allora quel paziente non è andato da tutti i medici!

LA DIVISIONE È UTILIZZATA NELLA PRESENZA DELL'AVVERBIO 'TUTTI'

PISTOLESI-SQL

Differenza con NOT IN

Indicare i pazienti visitati dal dott. Verdi ma non dal dott. Rossi

```
SELECT DISTINCT V.Paziente
FROM Visita V
    INNER JOIN
        Medico M ON V.Medico = M.Matricola
WHERE M.Cognome = 'Verdi'
    AND V.Paziente NOT IN
(
    SELECT V2.Paziente
    FROM Visita V2
        INNER JOIN
            Medico M2 ON V2.Medico = M2.Matricola
    WHERE M2.Cognome = 'Rossi'
);
```

The diagram illustrates the structure of the NOT IN query. A brace labeled '1' groups the main query part, which includes the SELECT statement, FROM clause, and the WHERE clause with the condition 'M.Cognome = 'Verdi''. A brace labeled '2' groups the subquery part, which starts with 'NOT IN' and contains the subquery itself, including its own SELECT, FROM, and WHERE clauses.

1. Prendi i pazienti, senza duplicati, visitati dal dottor Verdi, verificando che questi pazienti non siano tra:
2. I pazienti visitati dal dottor Rossi

OSS: Nella NON CORRELATED SUBQUERY è necessaria la ridenominazione alla tabella Visita e Medico, altrimenti ci riferiamo alla query esterna

Differenza a più insiemi (NOT IN + Subquery con <> “diverso”)

I pazienti visitati soltanto dal dott. Verdi

```
SELECT DISTINCT V.Paziente
FROM Visita V
    INNER JOIN
        Medico M ON V.Medico = M.Matricola
WHERE M.Cognome = 'Verdi'
    AND V.Paziente NOT IN
(
    SELECT V2.Paziente
    FROM Visita V2
        INNER JOIN
            Medico M2 ON V2.Medico = M2.Matricola
    WHERE M2.Cognome <> 'Verdi'
);
```

The diagram illustrates the structure of the NOT IN query with a different comparison operator. A brace labeled '1' groups the main query part, which includes the SELECT statement, FROM clause, and the WHERE clause with the condition 'M.Cognome = 'Verdi''. A brace labeled '2' groups the subquery part, which starts with 'NOT IN' and contains the subquery itself, including its own SELECT, FROM, and WHERE clauses, with a comparison operator '<>' used instead of 'NOT IN'.

1. Prendi i pazienti, senza duplicati, visitati dal dottor Verdi, verificando che questi pazienti non siano tra:

2. I pazienti visitati da un dottore che abbia cognome diverso da Verdi

OSS: Anche qui nella NON CORRELATED SUBQUERY è necessaria la ridenominazione alla tabella Visita e Medico, altrimenti ci riferiamo alla query esterna

PISTOLESI-SQL

Modificatori: SINTASSI (ANY/ALL)

Valore 1 [> / < / >= / <= / =] + ANY / ALL Valore 2

//Il valore 2 è ciascun record, ottenuto dal result set di una subquery

Modificatori “Almeno un record del result set” → ANY

Indicare il nome e cognome dei medici la cui parcella è superiore a quella di almeno un cardiologo di Pisa

```
SELECT Nome, Cognome }  
FROM Medico } 1  
WHERE Parcella > ANY }  
(  
    SELECT Parcella  
    FROM Medico M2  
    WHERE M2.Specializzazione = 'Cardiologia'  
        AND M2.Citta = 'Pisa'  
); } 2
```

1. Prendi il nome e cognome dei medici la cui parcella è maggiore di almeno una delle:
2. Parcalle dei medici, cardiologi di Pisa

OSS: Anche qui, nella NON CORRELATED SUBQUERY è necessaria la ridenominazione alla tabella Medico, altrimenti ci riferiamo alla query esterna

Modificatori “Tutti i record del result set” → ALL

Indicare la matricola del medico avente la parcella più bassa delle parcelle di tutti i medici

```
SELECT M1.Matricola  
FROM Medico M1  
WHERE M1.Parcella < ALL  
    (  
        SELECT M2.Parcella  
        FROM Medico M2  
        WHERE M2.Matricola <> M1.Matricola  
    );
```

- QUERY ESTERNA: Seleziona le matricole dei medici la cui parcella è più piccola
→ CORRELATED SUBQUERY delle parcelle dei medici che hanno matricola diversa dal medico (della QUERY ESTERNA)

//Il fatto del <> mi serve per non confrontare il medico con sé stesso

OSS: in caso di pari merito (EX AEQUO) la query restituirebbe un result set vuoto
→ Mi servirebbe quindi il \leq per avere i pari merito (vedi slide successiva) ← All'esame
può essere chiesto di avere un result set vuoto

PISTOLESI-SQL

Modifieri “Tutti i record del result set + EX AEQUO”

Indicare la matricola del medico avente la parcella più bassa delle parcelle di tutti i medici

```
SELECT M1.Matricola,  
FROM Medico M1  
WHERE M1.Parcella <= ALL  
(  
    SELECT M2.Parcella  
    FROM Medico M2  
);
```

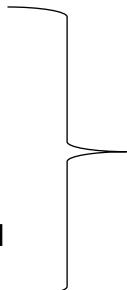
Query complesse

Indicare le specializzazioni contenenti solamente medici provenienti dalla stessa città, di cui almeno uno abbia superato nell'anno 2011 un totale di visite pari al 10% di tutte le visite della sua specializzazione nello stesso anno

PROBLEMA DIVISO IN SOTTOPROBLEMI → 3 RISULTATI A, B, C TRAMITE LE CTE, CHE ANDRANNO POI AD UNIRSI, UNO DOPO L'ALTRO FINO AD ARRIVARE ALLA QUERY FINALE

1. Considerare tutte le specializzazioni e i loro medici
2. In ogni specializzazione, contare il numero di città diverse a cui appartengono i medici che ne fanno parte
3. Imporre che ciascun conteggio precedente valga uno
4. In ogni specializzazione contenente solo medici della stessa città, contare le visite effettuate nel 2011 da ciascun medico che ne fa parte
5. In ogni specializzazione contenente solo medici della stessa città, calcolare il totale delle visite effettuate nel 2011 da tutti i medici che ne fanno parte
6. Imporre che esista almeno un medico che abbia effettuato nel 2011 più del 10% del totale delle visite effettuate nel 2011 da tutti i medici della sua stessa specializzazione

```
-----  
WITH SpecializzazioniTarget AS  
(  
    SELECT Specializzazione  
    FROM Medico  
    GROUP BY Specializzazione  
    HAVING COUNT(DISTINCT Citta) = 1  
)
```



RISULTATO A (1,2,3)

Con la condizione HAVING prendo solo le specializzazioni con le città UGUALI e posso avere due casi, per ogni specializzazione:

1. Ho tutte le città uguali in una determinata specializzazione
2. Ho solo una città in una determinata specializzazione (quindi solo una tupla), e di conseguenza va nella dicitura di ‘stessa città, essendocene solo una’

Se nel conteggio sulle città di ogni specializzazione, trovo due città diverse, quella specializzazione non va nel risultato

PISTOLESI-SQL

...CONTINUO Query complesse

```
, TotaleVisiteMedici AS  
(  
    SELECT M.Specializzazione, V.Medico,  
          COUNT(*) AS QuanteVisite  
     FROM Medico M  
           INNER JOIN  
             Visita V ON M.Matricola = V.Medico  
    WHERE M.Specializzazione IN (A) → OSS  
        AND YEAR(V.Data) = 2011  
   GROUP BY M.Specializzazione, V.Medico  
)
```

RISULTATO B (4)

- Per ogni specializzazione e per ogni medico, conto le visite dei medici la cui specializzazione sta tra → Le specializzazioni i cui medici provengono dalla stessa città (CTE A), e contemporaneamente l'anno è il 2011
- Faccio il raggruppamento per specializzazione e medico perché → Dal testo del punto 4. "In ogni specializzazione..." e "...da ciascun medico..." nascondono un GROUP BY

```
, TotaleVisiteSpecializzazioni AS  
(  
    SELECT B.Specializzazione  
          SUM(B.QuanteVisite) AS TotaleVisite  
     FROM B  
    GROUP BY B.Specializzazione  
)
```

RISULTATO C (5)

Prendo le specializzazioni, per le quali ci sono medici che provengono dalla stessa città e hanno fatto visite nel 2011 (dalla CTE B), e per ciascuna di queste specializzazioni ne faccio la somma totale

```
SELECT DISTINCT TVM.Specializzazione  
FROM TotaleVisiteMedici TVM  
      NATURAL JOIN  
    TotaleVisiteSpecializzazioni TVS  
WHERE TVM.QuanteVisite > 0.1*TVS.TotaleVisite;
```

RISULTATO FINALE (6)

- Data una specializzazione, affianco → Il numero di visite effettuate nel 2011, da un medico che ne fa parte (B) → Il totale delle visite effettuate nel 2011 da tutti i medici di quella specializzazione (C)
- Impongo che il valore che sta nel result set a SX sia il 10% superiore al valore che sta nel result set di DX
- A condizione soddisfatta, esiste almeno un medico che ha fatto, nel 2011, più del 10% del totale delle visite, di tutti i medici della sua specializzazione

PISTOLESI-SQL

Inserimento valori statici: SINTASSI

INSERT INTO Tabella (Attributo1, Attributo2,...,AttributoN)
VALUES (Valore1, Valore2, ..., ValoreN);

- **Tabella:** Tabella target
- **Attributo1, Attributo2,...,AttributoN** → Sono quelli da impostare, e può essere opzionale metterli se li voglio impostare tutti
- **Valore1, Valore2, ..., ValoreN** → Sono i valori da assegnare agli attributi

Inserimento valori statici (tutti i valori): INSERT INTO - VALUES

Inserire nel database un nuovo paziente, le cui informazioni sono:

- Nome: *Elvira*
- Cognome: *Passerotti*
- Sesso: *F*
- Data di nascita: *27 Ottobre 1965*
- Città: *Pisa*
- Reddito: *1500 €*
- Codice fiscale: *PSSLVR65R67G702U*

INSERT INTO Paziente

VALUES ('PSSLVR65R67G702U', 'Passerotti', 'Elvira', 'F', '1965-10-27', 'Pisa', 1500);

I CAMPI INSERITI DEVONO RISPETTARE L'ORDINE DELLO SCHEMA DI ATTRIBUTI

Inserimento per mancanza di informazioni

Tre giorni fa, la dottoressa Clelia Celesti ha visitato il paziente Edoardo Lepre, codice fiscale 'slq6'. La visita non è stata inserita. Lo si vuole fare in ritardo ma non si sa più se la visita era mutuata.

INSERT INTO Visita (Medico, Paziente, Data)

VALUES (010, 'slq6', CURRENT_DATE - INTERVAL 3 DAY);

Il valore di 'Mutuata' è andato perso, e viene impostata automaticamente al valore del default value, che solitamente è posto a NULL

Inserimento valori ricavati: SINTASSI

INSERT INTO Tabella (Attributo1, Attributo2,...,AttributoN)
Query_di_selezione

La query di selezione deve avere una proiezione coerente con la lista di attributi, (Attributo1, Attributo2,...,AttributoN) della tabella target

PISTOLESI-SQL

Inserimento valori ricavati: INSERT INTO - Query

Considerata la tabella VISITAVECCHIA avente lo schema riportato sotto, inserire in VISITAVECCHIA tutte le visite effettuate prima di due anni fa
VISITAVECCHIA (CognomePaziente, CognomeMedico, DataVisita, Specializzazione)

INSERT INTO VisitaVecchia

```
SELECT P.Cognome AS CognomePaziente,  
       M.Cognome AS CognomeMedico,  
       V.Data AS DataVisita,  
       M.Specializzazione AS Specializzazione  
FROM Visita V  
      INNER JOIN  
      Medico M ON V.Medico = M.Matricola  
      INNER JOIN  
      Paziente P ON V.Paziente = P.CodFiscale  
WHERE YEAR(V.Data) < YEAR(CURRENT_DATE) - 2;
```

OSS: le ridenominazioni sono coerenti con i nomi degli attributi della tabella VISITAVECCHIA

Considerando lo schema della tabella VISITAVECCHIA, posso dire che la chiave primaria è composta da tutti gli attributi, dato che sia i pazienti che i medici possono avere omonimi, quindi mi servono tutti gli attributi, per comporre PK

Aggiornamento: SINTASSI

UPDATE Tabella (Attributo1, Attributo2,...,AttributoN)

SET Attributo1 = Valore1, Attributo2 = Valore2, ..., AttributoN = ValoreN

WHERE Condizione

- **Tabella:** Tabella target
- **Attributo1, Attributo2,...,AttributoN** → Attributi da aggiornare
- **Attributo1 = Valore1, Attributo2 = Valore2, ..., AttributoN = ValoreN** → Sono i nuovi valori da assegnare agli attributi
- **Condizione** → Può essere articolata ed espressa tramite JOIN, SUBQUERY ecc...
Di condizioni ce ne possono essere anche più di una

//A volte si può anche omettere l'attributo da aggiornare, che sta accanto al nome della tabella target

PISTOLESI-SQL

Aggiornamento UPDATE – SET - WHERE

Modificare in “mutuata” tutte le visite del mese corrente, effettuate da pazienti nati prima del 1925

```
UPDATE Visita (Mutuata)
SET Mutuata = 1
WHERE MONTH(Data) = MONTH(CURRENT_DATE)
    AND YEAR(Data) = YEAR(CURRENT_DATE)
    AND Paziente IN
    (
        SELECT CodFiscale
        FROM Paziente
        WHERE YEAR(DataNascita) < 1925
    );
```

... AND il paziente sta tra → I pazienti dati prima dell'anno 1925 (SUBQUERY)

Cancellazione: SINTASSI

```
DELETE
FROM Tabella
WHERE Condizione
```

- **Tabella:** Tabella target
- **Condizione** → Può essere articolata, ed è espressa tramite, JOIN, SUBQUERY ecc..
Di condizioni ce ne possono essere anche più di una

Cancellazione DELETE FROM - WHERE

Il dottor Ettore Grigi ha lasciato la clinica. Rimuovere tutte le sue visite dal database, supponendo che il medico non avesse omonimi nella clinica.

```
DELETE
FROM Visita
WHERE Medico =
(
    SELECT Matricola
    FROM Medico
    WHERE Nome = 'Ettore'
        AND Cognome = 'Grigi'
);
```

- Detto a parole: “Cancella dalla tabella Visita, laddove la matricola del medico è uguale a → La matricola di Ettore Grigi (SUBQUERY NON CORRELATA)
- Si presuppone che ce ne sia soltanto uno di Ettore Grigi

PISTOLESI-SQL

Cancellazione: Riferimento alla tabella target nella condizione

Rimuovere dal database tutti i medici di Pisa che non hanno effettuato visite mutuate il mese scorso.

SOLUZIONE CON DERIVED TABLE:

```
DELETE  
FROM Medico  
WHERE Matricola IN
```

```
(  
    SELECT *  
    FROM (SELECT M1.Matricola  
          FROM Medico M1  
          LEFT OUTER JOIN  
            (SELECT V2.Medico AS MedicoMutuato  
             FROM Visita V2  
             INNER JOIN  
               Medico M2 ON V2.Medico = M2.Matricola  
             WHERE M2.Citta = 'Pisa'  
               AND V2.Mutuata IS TRUE  
               AND YEAR(V2.Data) = YEAR(CURRENT_DATE)  
               AND MONTH(V2.Data) = MONTH(CURRENT_DATE)-1  
            ) AS D  
          ON M1.Matricola = D.MedicoMutuato  
          WHERE D.MedicoMutuato IS NULL  
        ) AS D2  
);
```

Subquery

Derived Table (esterna)

Derived Table (interna)

- DERIVED TABLE INTERNA → Prende il medico che ha fatto visite mutuate a Pisa il mese scorso
- DERIVED TABLE ESTERNA → Considero i medici che non vengono combinati con i medici che hanno fatto visite mutuate a Pisa il mese scorso (con la DERIVED TABLE INTERNA), cioè quelli che hanno NULL a DX → Facendo così proietto le matricole dei medici che non hanno fatto visite mutuate a Pisa il mese scorso
- SUBQUERY → Seleziono tutto dal risultato precedente
- QUERY PRINCIPALE → Cancello dalla tabella Medico, le matricole dei medici che (SUBQUERY) non hanno fatto visite mutuate a Pisa il mese scorso

OSS: Viene utilizzata un'ulteriore tabella M1 per riferirsi al medico, perché la tabella Medico accanto al DELETE è bloccata in scrittura e non possiamo utilizzare dentro la subquery

PISTOLESI-SQL

...CONTINUO Cancellazione: Riferimento alla tabella target nella condizione

SOLUZIONE CON JOIN ANTICIPATO:

DELETE M1.*

FROM Medico M1

```
LEFT OUTER JOIN
(
    SELECT V2.Medico AS MedicoMutuato
    FROM Visita V2
        INNER JOIN
            Medico M2 ON V2.Medico = M2.Matricola
    WHERE M2.Citta = 'Pisa'
        AND V2.Mutuata IS TRUE
        AND YEAR(V2.Data) = YEAR(CURRENT_DATE)
        AND MONTH(V2.Data) = MONTH(CURRENT_DATE)-1
) AS D
ON M1.Matricola = D.MedicoMutuato
WHERE D.MedicoMutuato IS NULL;
```

1 – Record Target che rispecchiano la condizione (come nel punto precedente)

“**DELETE M1.*** → Guarda i record target e togili da M1

Sfrutto il fatto che la **DELETE** agisce sulla tabella **Medico**

QUANDO VIENE FATTO UNA CANCELLAZIONE O UN AGGIORNAMENTO, NON SI PUÒ METTERE LA TABELLA TARGET NEL FROM DELLA QUERY CONTENUTA NELLA CLAUSOLA WHERE → CIÒ COMPORTA UN ERRORE E SI PUÒ PORRE RIMEDIO UTILIZZANDO IL JOIN ANTICIPATO



Per l'esercizio di pagina precedente non avrei potuto scrivere:

```
DELETE
FROM Medico
WHERE Matricola IN
(
    SELECT M1.Matricola
    FROM Medico M1 ....
```

→ La tabella **Medico** è già stata bloccata in scrittura dalla **DELETE**

PISTOLESI-SQL

STORED PROCEDURE

Scrivere una stored procedure che stampi le specializzazioni mediche offerte dalla clinica

CREAZIONE PROCEDURA

```
DROP PROCEDURE IF EXISTS mostra_specializzazioni; → 1
DELIMITER $$ → 2
CREATE PROCEDURE mostra_specializzazioni() → 3
BEGIN → 4
    SELECT DISTINCT Specializzazione
    FROM Medico; } → 5
END $$ → 6
DELIMITER; → 7
```

CHIAMATA PROCEDURA

```
CALL mostra_specializzazioni(); → 8
```

- 1- Prima di creare la procedura mi assicuro di eliminarla se è già presente per non duplicarla → Nome procedura SENZA PARENTESI e PUNTO E VIRGOLA
- 2- Delimitatore \$\$ di fine statement → Finchè non ritrovo \$\$ il motore di processazione non compila nulla
- 3- Creo la procedura CON LE PARENTESI
- 4- Inizio del **BODY** della procedura
- 5- “Select Statement” equivale a stampare a video un’output (come il cout in c++) e rappresenta la query che risolve il testo dell’esercizio (con il PUNTO E VIRGOLA)
- 6- Fine del BODY della procedura con il delimitatore \$\$ di fine statement → Qua il motore di processazione compila
- 7- Reimposta il motore di processazione al PUNTO E VIRGOLA in modo da poter scrivere altre istruzioni da questo punto in poi
- 8- Chiamata che esegue la STORED PROCEDURE e ottiene il risultato restituito dall’esecuzione del BODY → Nome procedura CON LE PARENTESI e il PUNTO E VIRGOLA

PISTOLESI-SQL

Variabili locali: DECLARE – DEFAULT (SINTASSI)

```
DECLARE nome_variabile TIPO(SIZE) DEFAULT valore_default;
```

- Le variabili si dichiarano tutte insieme all'inizio del body
- **TIPO** → INT, DOUBLE, CHAR, VARCHAR, DATE ecc...
- **SIZE** → Capacità della variabile, per ES. l'intero è a 32 bit
OSS: se non è settato è il valore di default del tipo di dato
- Se non è presente il **valore_default**, il valore iniziale è NULL

Assegnamento statico (variabile) con SET

Supporre di essere nel body di una stored procedure e creare una variabile contenente il minimo numero di visite da effettuare, impostato a 20

```
DECLARE min_visite_mensili INT DEFAULT 0;
```

```
.
```

```
.
```

```
SET min_visite_mensili = 20;
```

Assegnamento calcolato (variabile) con SELECT+INTO

Supporre di essere nel body di una stored procedure e creare una variabile contenente il numero di visite effettuate nel mese in corso

1° MODO (INTO ACCANTO ALLA SELECT)

```
DECLARE visite_mese_attuale INT DEFAULT 0;
```

```
.
```

```
.
```

```
SELECT COUNT(*) INTO visite_mese_attuale
FROM Visita V
WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)
    AND YEAR(V.Data) = YEAR(CURRENT_DATE);
```

→ 2° MODO (INTO IN FONDO ALLA SELECT)

```
DECLARE visite_mese_attuale INT DEFAULT 0;
```

```
.
```

```
.
```

```
SELECT COUNT(*)
FROM Visita V
WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)
    AND YEAR(V.Data) = YEAR(CURRENT_DATE)
INTO visite_mese_attuale;
```

PISTOLESI-SQL

Assegnamento calcolato (variabile) con SET

Creare una variabile contenente il numero di visite effettuate nel mese in corso

```
DECLARE visite_mese_attuale INT DEFAULT 0;
```

```
SET visite_mese_attuale =  
(  
    SELECT COUNT(*)  
    FROM Visita V  
    WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)  
        AND YEAR(V.Data) = YEAR(CURRENT_DATE)  
);
```

Parametri di ingresso con (IN _nomeParametro)

Scrivere una stored procedure che stampi la parcella media di una specializzazione specificata come parametro

```
DROP PROCEDURE IF EXISTS parcella_media_spec;  
DELIMITER $$  
CREATE PROCEDURE parcella_media_spec(IN _specializzazione VARCHAR(100))  
BEGIN  
    SELECT AVG(M.Parcella)  
    FROM Medico M  
    WHERE M.Specializzazione = _specializzazione;  
END $$  
DELIMITER;
```

CHIAMATA

```
CALL parcella_media_spec('Ortopedia');
```

- Il parametro di ingresso IN viene letto ma non modificato → Perché se ne fa una copia di questo parametro e se ne modifica la copia → Dopodiché viene richiamata la copia con la CALL
- IN è opzionale, dato che i parametri sono in ingresso per default
- Per convenzione si mette l'underscore _ in testa al nome della variabile, per indicare che si tratta di un parametro di ingresso
- Il valore in ingresso essendo composto da caratteri lo metto tra apici singoli

PISTOLESI-SQL

Parametri di uscita con (OUT nomeParametro_) + variabile USER-DEFINED @

Scrivere una stored procedure che restituisca il numero di pazienti visitati da medici di una data specializzazione, ricevuta come parametro

```
DROP PROCEDURE IF EXISTS tot_pazienti_visitati_spec;
```

```
DELIMITER $$  
CREATE PROCEDURE tot_pazienti_visitati_spec  
(  
    IN _specializzazione VARCHAR(100),  
    OUT totale_pazienti_ INT  
)  
BEGIN  
    SELECT COUNT (DISTINCT V.Paziente) INTO totale_pazienti_  
    FROM Visita V  
        INNER JOIN  
            Medico M ON V.Medico = M.Matricola  
    WHERE M.Specializzazione = _specializzazione;  
END $$  
DELIMITER;
```

CHIAMATA

```
CALL tot_pazienti_visitati_spec('Neurologia', @quantiPazienti);
```

OUTPUT

```
SELECT @quantiPazienti;
```

- Il parametro di uscita **OUT** va messo tra i parametri
- Per convenzione si mette l'underscore _ in coda al nome della variabile, per indicare che si tratta di un parametro di uscita
- La INTO mi scrive il risultato della subquery ti tipo SCALARE dentro il parametro di uscita
- Al momento della chiamata, il valore del parametro di uscita si inserisce nella variabile USER-DEFINED @
- La variabile @:
 - Mi serve per stampare a video il risultato della procedura
 - Non è necessario dichiararla
 - Può contenere qualsiasi tipo di dato, purchè questo stia SCALARE
 - Quando si chiama si deve sempre anteporre la @ prima del loro nome

PISTOLESI-SQL

Istruzione IF THEN – ELSEIF THEN – ELSE – END (SINTASSI)

```
IF if_condition THEN
    blocco istruzioni if true; } 1
ELSEIF elseif_1_condition THEN
    blocco istruzioni elseif_1;
.
.
.
ELSEIF elseif_N_condition THEN
    blocco istruzioni elseif_N;
ELSE
    blocco istruzioni else; } 3
END IF; → 4
```

- 1) Si verifica quando è vera la prima condizione IF
- 2) Sono le sottocondizioni della prima condizione IF e se tutte queste condizioni non valgono passo a 3
- 3) Ultima condizione prima di uscire del tutto
- 4) Chiusura di tutti gli IF precedenti, compreso quello iniziale

OSS:

- 2) e 3) sono parte facoltativa
- 4) è obbligatoria
- Alla fine di ogni blocco è necessario mettere il punto e virgola

Istruzione CASE - WHEN THEN – END CASE (SINTASSI)

```
CASE
WHEN condition_1 THEN
    blocco istruzioni_1; } 1
.
.
.
WHEN condition_N THEN
    blocco istruzioni_N; } 2
ENDCASE; → 3
```

- 1) Quando si verifica la condizione 1 → Fai ‘blocco_istruzioni_1:’
.
.
.
- 2) Quando si verifica la condizione N → Fai ‘blocco_istruzioni_N:’
- 3) Chiusura dell’istruzione CASE

OSS:

- Alla fine di ogni blocco è necessario mettere il punto e virgola
- Equivarrebbe all’istruzione SWITCH del C++

PISTOLESI-SQL

Esempio di utilizzo di IF

Scrivere una stored procedure che riceva come parametro un intero `t` e una specializzazione `s` e restituisca in uscita true se il totale delle visite della specializzazione `s` nel mese in corso è superiore all'intero `t`, false se è inferiore e NULL se è uguale.

```
1  DROP PROCEDURE IF EXISTS visite_sopra_soglia;
2
3  DELIMITER $$ 
4  CREATE PROCEDURE visite_sopra_soglia(IN _t INT, IN _s VARCHAR(100), OUT passed BOOLEAN)
5  BEGIN
6      DECLARE visite_mese_attuale INT DEFAULT 0;
7      SET visite_mese_attuale = (
8          SELECT COUNT(*)
9              FROM Visita V
10             INNER JOIN
11                 Medico M ON V.Medico = M.Matricola
12            WHERE M.Specializzazione = _s
13            AND MONTH(V.`Data`) = MONTH(CURRENT_DATE)
14            AND YEAR(V.`Data`) = YEAR(CURRENT_DATE)
15      );
16
17      IF visite_mese_attuale > _t THEN
18          SET passed = TRUE;
19      ELSEIF visite_mese_attuale < _t THEN
20          SET passed = FALSE;
21      ELSE
22          SET passed = NULL;
23  END IF;
24  END $$ 
25  DELIMITER ;
26
27  CALL visite_sopra_soglia(10, 'Otorinolaringoiatria', @controllo);
```

- RIGA 4 → I parametri di ingresso sono `_t` e `_s` e il parametro di uscita è `passed`, che è di tipo BOOLEAN dato che mi deve restituire true/false
- RIGA 6 → Variabile dove ci inserirò il totale delle visite del mese in corso (RIGA 8, RIGA 9)
- Setto la variabile dichiarata prima con il conteggio (RIGA 8, RIGA 9) del totale delle visite del mese in corso (RIGA 13, RIGA 14) di un certo medico con una certa matricola (RIGA 11), ma avente la specializzazione uguale alla specializzazione presa in ingresso (RIGA 12)
- RIGA 17, RIGA 18 → Caso totale delle visite della specializzazione `_s` nel mese in corso è superiore all'intero `_t` → Imposto la variabile passed a TRUE
- RIGA 19, RIGA 20 → Caso totale delle visite della specializzazione `_s` nel mese in corso è inferiore all'intero `_t` → Imposto la variabile passed a FALSE
- RIGA 21, RIGA 22 → Altri casi di confronto tra il totale delle visite della specializzazione `_s` nel mese in corso e l'intero `_t` (cioè il caso di uguaglianza) → Imposto la variabile passed a NULL
- RIGA 27 → Chiamata alla procedura e variabile d'uscita @ (USER DEFINED), richiamabile successivamente con SELECT @controllo;

OSS: Gli apici `` su Data indicano che questa è un attributo di Visita e non una KEYWORD

PISTOLESI-SQL

Più variabili di uscita – Ritorno di un unico record (LIMIT)

Scrivere una stored procedure che restituisca la data in cui un paziente, il cui codice fiscale è passato come parametro, è stato visitato per la prima volta, e il nome e cognome del medico che lo ha visitato in tale circostanza. In caso di più medici, per semplicità, selezionarne uno

```
1  DELIMITER $$  
2  CREATE PROCEDURE primaVisitaPaziente(IN paziente VARCHAR(16),  
3                                         OUT dataPrimaVisita DATE,  
4                                         OUT cognomeMedico VARCHAR(100),  
5                                         OUT nomeMedico VARCHAR(100)  
6  BEGIN  
7      SELECT MAX(V.`Data`) INTO dataPrimaVisita  
8      FROM Visita V  
9      WHERE V.Paziente = paziente;  
10  
11     SELECT M.Cognome, M.Nome INTO cognomeMedico, nomeMedico  
12     FROM Visita V INNER JOIN Medico M ON V.Medico = M.Matricola  
13     WHERE V.Paziente = paziente  
14         AND V.`Data` = dataPrimaVisita  
15     LIMIT 1;  
16 END $$  
17 DELIMITER ;
```

- In ingresso prendo il codice fiscale di un paziente (RIGA 2)
- Da RIGA 7 a RIGA 9 faccio un assegnamento calcolato sulla variabile di uscita della data della prima visita di un paziente il cui codice fiscale è dato in ingresso (RIGA 9) → Il fatto che sia una prima visita lo noto dal fatto che considero la data massima → Quindi la data più remota e la restituisco con la proiezione
- Da RIGA 11 a RIGA 15 faccio un assegnamento calcolato sulle varabili di uscita del cognome e del nome del medico, che ha effettuato una visita che corrisponde ad una prima visita (RIGA 14) di un paziente il cui codice fiscale è dato in ingresso (RIGA 13), prendendone solamente 1 (RIGA 15)
- **LIMIT 1** → Questo mi limita il result set ad un solo record!
- Infine, tutte e tre le variabili di uscita, attraverso una CALL alla procedura → Le inserisco in 3 variabili @NomeVariabile1, @NomeVariabile2, @NomeVariabile3 (USER-DEFINED), che potrò poi visualizzare con l'utilizzo di **SELECT @NomeVariabile1, @NomeVariabile2, @NomeVariabile3**

WHILE DO – END WHILE (SINTASSI)

WHILE condition DO

 blocco istruzioni;

END WHILE;

- Finchè è vera la condizione *condition* il blocco di istruzioni viene eseguito →DO solo se la condizione è TRUE
- La condizione *condition* viene controllata prima di ogni operazione

PISTOLESI-SQL

REPEAT – UNTIL – END REPEAT (SINTASSI)

REPEAT

blocco istruzioni;

UNTIL condition

END REPEAT;

- Il *blocco istruzioni* viene eseguito finchè è vera la condizione *condition*
→ Non appena la *condition* è falsa si esce dal ciclo
- La condizione di uscita è dovuta all'UNTIL
- La condizione è controllata dopo ogni iterazione → Questo fa sì che almeno una volta ci entri nel ciclo REPEAT

OSS: Questo ciclo è simile al DO...WHILE che abbiamo nel C++

LOOP – END LOOP (SINTASSI)

loop_label: LOOP

blocco istruzioni e check di condizioni (con istruzioni di salto);

END LOOP;

- Per il LOOP ho un'etichetta → *loop_label*
- Il blocco di istruzioni viene eseguito finchè non ho istruzioni di salto che me lo fanno terminare come LEAVE e ITERATE

LEAVE

LEAVE loop_label;

- Serve a farci uscire dal ciclo LOOP → Ci porta subito dopo l'END LOOP;

OSS: In C++ è l'equivalente dell'istruzione BREAK

ITERATE

ITERATE loop_label;

- Ci permette di passare all'operazione successiva del ciclo LOOP

OSS: In C++ è l'equivalente dell'istruzione CONTINUE

PISTOLESI-SQL

Esempio di LOOP con istruzioni di salto: LEAVE / ITERATE + Cocatenazione CONCAT

Scrivere una stored procedure che riceve in ingresso un intero i e stampa a video i numeri dispari fino ad i , separati da virgola (ES. $i=5$, l'uscita deve essere → 1,3,5)

```
1  DROP PROCEDURE IF EXISTS stampa_dispari;
2  DELIMITER $$ 
3  CREATE PROCEDURE stampa_dispari(IN i INT)
4  BEGIN
5      DECLARE s VARCHAR(255) DEFAULT '';
6      DECLARE counter INT DEFAULT 1;
7
8      IF i>0 THEN
9          SET s = '1';
10     END IF;
11
12    scan: LOOP
13        SET counter = counter + 1;
14
15        IF counter = i+1 THEN
16            LEAVE scan;
17        END IF;
18
19        IF (counter % 2) = 0 THEN
20            ITERATE scan;
21        ELSE
22            SET s = CONCAT(s,',',counter);
23        END IF;
24    END LOOP;
25
26    SELECT s;
27 END $$ 
28 DELIMITER ;
```

- RIGA 3 → Gli passo l'intero i in ingresso, che rappresenta i primi i interi
- RIGA 5 → Variabile dove concatenero i numeri dispari trovati fino ad i , per default è vuota, indicata con → ''
- RIGA 6 → Variabile contatrice che mi servirà per il ciclo
- Da RIGA 8 a RIGA 10 → Se ho almeno un intero (o anche più di uno, basta che non siano 0) allora inserisco nella stringa ‘concatenatrice’ il valore 1 → Quindi almeno il primo numero dispari lo devo stampare
- RIGA 12 → Inizio il ciclo (che terminerò poi a RIGA 24) incrementando la variabile contatrice (RIGA 13) per ogni passata
- Da RIGA 15 a RIGA 17 → Se la variabile contatrice supera quella di i allora ho finito i valori i da verificare e quindi esco dal ciclo con la **LEAVE**
- RIGA 19, RIGA 20 → Se ho un numero pari, non lo considero e vado all’istruzione successiva con la **ITERATE**
- RIGA 21, RIGA 22 → Caso in cui ho un numero dispari, cioè la variabile contatrice è dispari) e lo concateno con **CONCAT** → Funzione che costruisce la stringa (convertendo gli interi in caratteri) concatenata così, in questo caso:
(valore di s che era dispari, virgola, valore dispari della variabile contatrice)
e il tutto lo inserisco in S, che avrà primi i interi dispari che stamperò nella proiezione RIGA 26

PISTOLESI-SQL

Cursori: DECLARE – CURSOR FOR (SINTASSI)

**DECLARE nomeCursore CURSOR FOR
Query**

- Il cursore viene associato ad un insieme di record → Che è rappresentato dal result set della Query
- Permette di scorrere un result set in avanti, per effettuare delle azioni, all'interno dei cicli

OSS: Si dichiara dopo aver dichiarato tutte le variabili

Apertura, prelevamento, chiusura cursori: OPEN, FETCH, CLOSE (SINTASSI)

1. **OPEN NomeCursore;**
 2. **FETCH NomeCursore INTO ListaVariabili**
 3. **CLOSE NomeCursore;**
-
1. Serve ad aprire il cursore, per poi poterlo usare
 2. La **FETCH** serve a prelevare il record, riga per riga → Ogni record prelevato si inserisce all'interno di una variabile locale della lista di tutte le variabili (**INTO ListaVariabili**)
 3. Serve a chiudere il cursore, e si fa dopo aver fatto **FETCH** su tutti i record

Cursori: NOT FOUND CONTINUE HANDLER (SINTASSI)

DECLARE CONTINUE HANDLER FOR NOT FOUND

Eseguo un'azione

- È un gestore di situazioni ed è utile per riconoscere, dopo aver scorso tutti i record, quando si è giunti alla fine di un result set
- In C++ equivrebbe ad avere, **NEXT=NULL**, nello scorrimento di una lista
- **CONTINUE** → Implica che il flusso continua, uscendo per sempre dal cursore → Interrompendo il ciclo di estrazione dei record del result set → Senza però, terminare l'esecuzione con un **ABORT!**

OSS: Si definisce dopo aver definito tutte le variabili e i cursori → Perché ogni cursore dichiarato ha bisogno del suo **HANDLER** (di fine corsa)

ESEMPIO NOT FOUND CONTINUE HANDLER

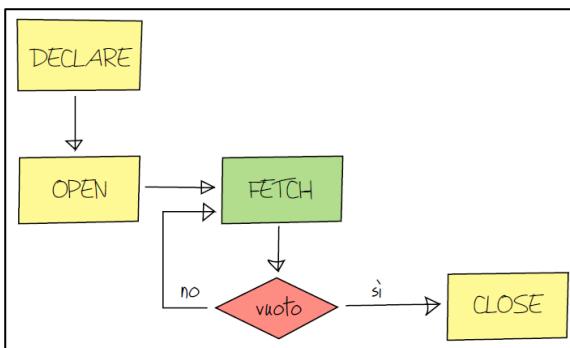
DECLARE CONTINUE HANDLER FOR NOT FOUND

SET finito=1;

Spiegazione a parole: "Dichiara l'HANDLER di tipo CONTINUE, cioè che esca senza produrre errori per l'evento NOT FOUND → Quindi non trovo un altro record da estrarre → Di conseguenza ho finito il result set → In questo caso settami la variabile *finito* a 1 (la quale è stata già dichiarata precedentemente)"

PISTOLESI-SQL

FUNZIONAMENTO COMPLETO DEL CURSOR



- Il **FETCH** fa avanzare il cursore di una posizione e se l'HANDLER non segnala la fine del result set → Preleva un record
- **NO** → Caso in cui VUOTO non si verifica e di conseguenza il NOT FOUND non si verifica → Torno su e faccio un **FETCH** al record successivo
- **SÌ** → Caso in cui VUOTO si verifica (quindi il NOT FOUND si verifica) → Non ci sono più record da estrarre

Esempio di funzionamento del cursore COMPLETO + parametri di uscita (INOUT)

Scrivere una stored procedure che restituisca il codice fiscale dei pazienti visitati da un solo medico per una data specializzazione, organizzati in una stringa formattata del tipo “codFiscale1, codFiscale2, … , codFiscaleN”

```
1  DELIMITER $$  
2  CREATE PROCEDURE PazientiSingoloMedico(IN specializzazione CHAR,  
3  INOUT codiciFiscali VARCHAR (255))  
4  BEGIN  
5      DECLARE finito INTEGER DEFAULT 0;  
6      DECLARE codiceFiscale VARCHAR(255) DEFAULT “”;  
7  
8      -- dichiarazione del cursore  
9      DECLARE cursoreCodici CURSOR FOR  
10         SELECT V.Paziente  
11             FROM Visita V INNER JOIN Medico M ON V.Medico = M.Matricola  
12                 WHERE M.Specializzazione = specializzazione  
13                 GROUP BY V.Paziente  
14                 HAVING COUNT(DISTINCT V.Medico) = 1;  
15  
16      -- dichiarazione handler  
17      DECLARE CONTINUE HANDLER  
18          FOR NOT FOUND SET finito = 1;  
19  
20      OPEN cursoreCodici;  
21  
22      -- ciclo di fetch per il prelievo  
23      preleva: LOOP  
24          FETCH cursoreCodici INTO codiceFiscale;  
25          IF finito = 1 THEN  
26              LEAVE preleva;  
27          END IF;  
28          SET codiciFiscali = CONCAT(codiceFiscale, “;”, codiciFiscali);  
29      END LOOP preleva;  
30      CLOSE cursoreCodici;  
31  END $$  
32  DELIMITER ;
```



PISTOLESI-SQL

... Continuo Esempio di funzionamento del cursore COMPLETO + parametri di uscita (INOUT)

- RIGA 3 → Parametro **INOUT** (ingresso/uscita) dove ci andrò a mettere i codici fiscali che mi richiede il testo
- RIGA 5 → Dichiaro questa variabile, con valore di DEFAULT a 0 → Questa mi rappresenta il fine corsa dell'HANDLER, il quale a sua volta, nel momento opportuno la trasformerà in 1
- RIGA 9 → Dichiaro il cursore che mi scansionerà il result set della query (da RIGA 10 a RIGA 14) che conterrà → I codici fiscali dei pazienti (RIGA 10) visitati da un solo medico (RIGA 14), di una data specializzazione (RIGA 12), cioè dal parametro in ingresso *specializzazione* dichiarato a RIGA 2
- RIGA 17 e RIGA 18 → Dichiaro l'HANDLER per l'evento NOT FOUND → Il quale HANLER è riferito al FETCH del cursore → e lo imposto a 1 quando abbiamo scorso tutto il result set senza trovare nulla
- RIGA 24 → Faccio il FETCH del cursore, il quale pescherà i record dal result set della query (da RIGA 10 a RIGA 14) e li metterà nella variabile *codiceFiscale* (RIGA 6)
- RIGA 25 → Sono nel caso in cui il FETCH non ha prodotto valori, cioè ho già estratto tutto oppure non ho più niente da estrarre → Quindi interrompo il ciclo ed esco (RIGA 26)
- RIGA 28 → Sono nel caso in cui l'HANDLER non ha trovato il fine corsa → Setto la variabile di ingresso/uscita *codiciFiscali* così → Allora prendo il codice fiscale estratto dal cursore (RIGA 24) → Ci concateno il punto e virgola → Poi ci concateno ciò che c'era già in *codiciFiscali*, presa in ingresso/uscita (RIGA 3)

OSS: Si può dire che è stato fatto una specie di inserimento in testa

CHIAMATA

```
SET @codiciPazienti = '';
CALL PazientiSingoloMedico('Ortopedia', @codiciPazienti);
SELECT @codiciPazienti;
```

- Inizializzo una variabile di tipo USER-DEFINED a stringa vuota, per non avere NULL al momento CONCAT dentro la procedura (brutto esteticamente)
- Il risultato atteso viene inserito nella variabile *codiciPazienti* dato che il parametro è di tipo INOUT
- Non mi resta altro che chiamare con la SELECT questa variabile!

PISTOLESI-SQL

Gestione degli errori con HANDLER (SINTASSI)

```
DECLARE action HANDLER FOR condition_value  
      statement(s);
```

Le azioni (action) possono essere:

- **EXIT**
- **CONTINUE**

Le condition_value possono essere:

- Codici errore MySQL
- Valori SQLSTATE
- SQLEXCEPTION
- SQLWARNING
- NOT FOUND

Lo statement:

- Blocco di istruzioni racchiuso fra BEGIN-END

Gestione degli errori con CONTINUE HANDLER-SQLEXCEPTION

Scrivere un handler che imposti a 1 una variabile d'errore qualora si verifichi un'eccezione, ma non interrompa comunque la processazione

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION  
      SET errore = 1;
```

- Il CONTINUE fa in modo che l'esecuzione non termini
- Impostare la variabile errore a 1 vuol dire che l'eccezione **SQLEXCEPTION** è verificata non appena si attiva l'HANDLER → Quindi vuol dire che il comportamento errato viene catturato senza interrompere l'esecuzione

Gestione degli errori con EXIT HANDLER-ROLLBACK

Scrivere un handler che, al verificarsi di un'eccezione, annulli tutte le precedenti operazioni, stampi un messaggio d'errore, e interrompa la processazione.

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION  
      BEGIN  
          ROLLBACK;  
          SELECT 'Si è verificato un errore!';  
      END;
```

- Il **ROLLBACK** annulla tutto quanto fatto in precedenza e ne stampa un messaggio di errore
- L'**EXIT** fa terminare l'esecuzione
- Il blocco di istruzioni tra il BEGIN e l'END → Si chiama **statement**

PISTOLESI-SQL

Sollevare gli errori: SIGNAL SQLSTATE – SET (SINTASSI)

```
SIGNAL SQLSTATE '45000'          //useremo solo questo di errore che è generico
      SET condition_information_item_name1 = value1,
      .
      .
      .
      condition_information_item_nameN = valueN;
```

- Il tipo di errore **45000** è quello generico grave!
- Le condition da 1...N sono un insieme di azioni che possono essere eseguite dopo il lancio dell'eccezione
- Come condizione dopo la SET scriviamo → **MESSAGE_TEXT** = “Errore...”

Esempio sollevare gli errori: SIGNAL SQLSTATE – SET

Scrivere una stored procedure che aumenta la parcella di un dato medico di 10 euro solo se questa non è superiore alla media delle parcelle.

Nel caso la parcella non sia da aggiornare, deve essere sollevato un errore generico (SQLSTATE '45000') che ne specifichi la causa

```
1  DELIMITER $$ 
2
3  DROP PROCEDURE IF EXISTS aumentaParcella;
4  CREATE PROCEDURE aumentaParcella(matricolaMedico VARCHAR(255))
5
6  BEGIN
7
8      DECLARE mediaParcelle DOUBLE DEFAULT 0;
9      DECLARE parcellaMedico DOUBLE DEFAULT 0;
10
11     -- calcolo della parcella media dei medici
12     SELECT AVG(Parcella) INTO mediaParcelle
13     FROM Medico;
14
15     -- ricerca della parcella del medico
16     SELECT Parcella INTO parcellaMedico
17     FROM Medico
18     WHERE Matricola = matricolaMedico;
19
20     -- controllo di flusso
21     IF parcellaMedico >= mediaParcelle THEN
22         SIGNAL SQLSTATE '45000'
23         SET MESSAGE_TEXT = 'Medico con parcella superiore alla media';
24     ELSE
25         UPDATE Medico
26         SET Parcella = Parcella + 10
27         WHERE Matricola = matricolaMedico;
28     END IF;
29
30 END $$ 
31
32 DELIMITER ;
```

- RIGA 21 → Caso in cui la parcella non sia da aggiornare → Sollevo l'errore generico (RIGA 22) → Specifico la causa con un messaggio di testo (RIGA 23)

PISTOLESI-SQL

STORED FUNCTION (SINTASSI)

DELIMITER \$\$

DROP FUNCTION function_name IF EXISTS function_name

CREATE FUNCTION function_name(parametro1, ..., parametroN)

RETURNS datatype DETERMINISTIC | NOT DETERMINISTIC

.

.

.

 RETURN function_name;

END \$\$

DELIMITER;

LE FUNCTION RESTITUISCONO UN SOLO VALORE E MAI UN RESULT SET

- Come unico valore restituito possiamo avere → Per esempio, un numero, una stringa, un booleano ecc... → Questo è dovuto al **RETURNS datatype**
- Il fatto di restituire un solo valore lo abbiamo al **RETURN function_name**
- Sono richiamabili da statement SQL
- Non hanno bisogno di una CALL
- Esempi di STORED FUNCTION sono: MIN/MAX/COUNT/AVG ecc...
→ Cioè quelle messe a disposizione di SQL
- LA DROP FUNCTION, controlla che se la funzione esiste già, di cancellarla e sostituirla
- I parametri *parametro1, ..., parametroN* → Sono solo parametri di ingresso
- Una funzione è di tipo **DETERMINISTIC** se, dando gli stessi valori in ingresso, si avrà sempre lo stesso risultato in uscita, ogni volta che si usa
→ Per esempio, se diamo in ingresso ad una funzione $2+2$ → Ogni volta che si richiama la funzione, in uscita avremo sempre 4 → Risultato che non cambia nel tempo
- Una funzione è di tipo **NOT DETERMINISTIC** se, avendo gli stessi parametri di ingresso, in uscita si può avere un valore diverso, da esecuzione ad esecuzione → Per esempio, se data la matricola di un medico, vengono restituite il numero di visite fatte → Questa è non deterministica, perché anche se la matricola del medico rimarrà la stessa, questo medico nel tempo continuerà a visitare → Di conseguenza il valore in uscita sarà diverso
- La FUNCTION può essere chiamata:
 - Dalle query SQL
 - Dalle stored procedure
 - Dai trigger
 - Dagli event

PISTOLESI-SQL

Esempio di STORED FUNCTION

Scrivere una function che, preso in ingresso un numero di visite effettuate da un medico, restituisca: ‘low’ se il numero di visite è inferiore a 20; ‘medium’ se il numero di visite è compreso fra 20 e 50; ‘high’ se il numero di visite supera 50.

```
1  DELIMITER $$  
2  DROP FUNCTION IF EXISTS rank;  
3  
4  CREATE FUNCTION rank(totaleVisite INT)  
5  RETURNS VARCHAR(6) DETERMINISTIC  
6  
7  BEGIN  
8      -- variabile risultato  
9      DECLARE ranking VARCHAR(6) DEFAULT "";  
10  
11     -- assegnamento del ranking su condizione  
12     CASE  
13         WHEN totaleVisite < 20 THEN  
14             SET ranking = 'low';  
15         WHEN totaleVisite BETWEEN 20 AND 50 THEN  
16             SET ranking = 'medium';  
17         WHEN totaleVisite > 50 THEN  
18             SET ranking = 'high';  
19     END CASE;  
20  
21     -- restituzione del risultato  
22     RETURN (ranking);  
23 END $$  
24 DELIMITER ;
```

- La function è di tipo DETERMINISTIC (RIGA 5) perché per qualsiasi valore gli si dia in ingresso, indipendentemente da esso restituirò sempre, a seconda delle casistiche i valori:
 - Low
 - Medium
 - High
- RIGA 9 → Dichiaro la variabile di uscita *ranking* come stringa vuota, di 6 caratteri perché l’output maggiore che potrei avere in uscita sarebbe la stringa *medium*, la quale è di 6 caratteri
- Questa variabile di uscita andrà a finire al valore della funzione che ritornerò di 6 caratteri come descritta a RIGA 5
- A seconda dei tre casi richiesti dal testo, gestiti da RIGA 12 a RIGA 19
→ A RIGA 22 restituirò la variabile di uscita *ranking*

PISTOLESI-SQL

Esempio di utilizzo di una FUNCTION

Scrivere una stored procedure che restituisca la posizione in classifica nel mese in corso di un medico, passato come parametro, sfruttando la function rank()

```
1 DELIMITER $$  
2 DROP PROCEDURE IF EXISTS classificaMedico;  
3 CREATE PROCEDURE classificaMedico(IN matricola VARCHAR(5), OUT classe VARCHAR(6))  
4  
5 BEGIN  
6  
7     DECLARE visiteMeseCorrente INT DEFAULT 0;  
8     DECLARE matricolaValida INT DEFAULT 0;  
9  
10    SELECT COUNT(*) INTO matricolaValida  
11    FROM Medico M  
12    WHERE M.Matricola = matricola;  
13  
14    IF matricolaValida = 0 THEN  
15        BEGIN  
16            SET classe = 'NULL';  
17            SIGNAL SQLSTATE '45000'  
18            SET MESSAGE_TEXT = 'Medico inesistente!';  
19        END;  
20    ELSE  
21        BEGIN  
22            SELECT COUNT(*) INTO visiteMeseCorrente  
23            FROM Visita V  
24            WHERE V.Medico = matricola  
25                AND MONTH(V.Data) = MONTH(CURRENT_DATE);  
26  
27            SELECT rank(visiteMeseCorrente) INTO classe;  
28        END;  
29    END IF;  
30  
31 END $$  
32 DELIMITER ;
```

- La funzione *rank* è dell'esempio precedente
- RIGA 3 → In uscita restituirò *classe* che avrà poi i valori restituiti da *rank*
- RIGA 8 → Variabile che mi farà il ‘check’ sulla matricola
- Da RIGA 10 a RIGA 12 → Inserisco nella variabile che mi servirà per il ‘check’ il conteggio (RIGA 10), laddove il medico passato tra i parametri è tra i medici della clinica (RIGA 12)
- RIGA 14 → Caso in cui non ho matricole, data quella in ingresso, uguali a quelle della tabella medici allora:
 - Imposto la variabile di uscita come stringa ‘NULL’ (RIGA 16)
 - Lancio il messaggio di errore generico (RIGA 17)
 - Lancio l'errore sottoforma di testo (RIGA 18)
- RIGA 20 → Caso in cui ho matricole, data quella in ingresso, uguali a quelle della tabella medici allora → Nella variabile *visiteMeseCorrente* inserisco il conteggio (RIGA 22) → Delle visite del medico, la cui matricola è data in ingresso (RIGA 24) → Nel mese corrente (RIGA 25)
- RIGA 27 → Utilizzo la funzione *rank* con il parametro di ingresso rappresentato dal valore di *visiteMeseCorrente* → Il valore (in stringa) ottenuto dalla *rank* → Lo inserirò nella variabile di uscita *classe*

PISTOLESI-SQL

Esempio di utilizzo di una FUNCTION + TEMPORARY TABLE

Scrivere una stored procedure che produca **una classifica di tutti i medici** della clinica sfruttando la function rank()

```
1 DELIMITER $$  
2 DROP PROCEDURE IF EXISTS classificaMedici;  
3 CREATE PROCEDURE classificaMedici()  
4  
5 BEGIN  
6   CREATE TEMPORARY TABLE IF NOT EXISTS _Classifica(  
7     Medico varchar(100) NOT NULL,  
8     Rank varchar(10) NOT NULL,  
9     PRIMARY KEY (Medico)  
10    ) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
11  
12   TRUNCATE TABLE _Classifica;  
13  
14   INSERT INTO _Classifica  
15     SELECT V.Medico, rank(COUNT(*))  
16       FROM Visita V  
17      WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)  
18        GROUP BY V.Medico;  
19 END $$  
20 DELIMITER ;
```

- Da RIGA 6 a RIGA 10 → Creo una tabella temporanea *_Classifica* con quegli attributi solo se non esisteva in precedenza
- Il comando **TRUNCATE TABLE** → Mi permette di svuotare la tabella → Che in questo caso alla RIGA 12 non ha senso, perché non ce l'avevo prima questa tabella
- Inserisco nella tabella (RIGA 14) → La matricola dei medici e la propria posizione in termini di 'high', 'medium', 'low', sfruttando la *rank*(RIGA 15) sul conto delle → Visite fatte nel mese corrente dai medici (RIGA 17) → E lo faccio per ciascuno medico (RIGA 18) → Questo accade, ogni volta che alla **SELECT** gli arriva un medico → Così si crea una specie di classifica

UNA TEMPORARY TABLE MANTIENE I RISULTATI UTILI ALL'INTERNO DELLA SESSIONE E, QUESTE TABELLE A FINE SESSIONE VENGONO CANCELLATE

CHIAMATA

```
1 CALL classificaMedici();  
2 SELECT * FROM _Classifica;
```

OUTPUT

Medico	Rank
001	high
003	low
004	low
005	low
006	medium
007	low
008	high
009	low

Ranking dei medici
rispetto alla funzione
rank

PISTOLESI-SQL

Sintassi TRIGGER

```
DROP TRIGGER IF EXISTS nome_trigger; //1  
CREATE TRIGGER nome_trigger //2  
[BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON TabellaTarget //3  
FOR EACH ROW //4  
blocco_istruzioni //5
```

- Il trigger, definito da un insieme di istruzioni, è qualcosa “che scatta” al seguito del verificarsi di una causa scatenante → La quale causa è un’istruzione DML su un determinato database
 - Rappresenta l’azione della triade: EVENTO-CONDIZIONE-AZIONE
- COMANDI:
1. Cancella il trigger di quel “nome_trigger” se esiste già
 2. Crea il trigger con quel “nome_trigger”
 3. Prima | dopo un inserimento | modifica | cancellazione di una row, relativa ad una tabella specifica che vogliamo monitorare
 4. Per ciascuna row considerata
 5. Esegui il “blocco_istruzioni”
- Il trigger di tipo BEFORE rappresenta un’azione che viene eseguita prima che un record sia inserito | modificato | cancellato
 - Il trigger di tipo AFTER rappresenta un’azione che si verifica a seguito di una esecuzione di inserimento | modifica | cancellazione

Sintassi TRIGGER (+ NEW)

Gestire un attributo ridondante nella tabella Paziente contenente la data nella quale un paziente è stato visitato l’ultima volta

```
DROP TRIGGER IF EXISTS aggiorna_ultima_visita;  
CREATE TRIGGER aggiorna_ultima_visita  
AFTER INSERT ON Visita  
FOR EACH ROW //1  
UPDATE Paziente //2  
SET UltimaVisita=CURRENT_DATE //3  
WHERE CodFiscale=NEW.Paziente; //4
```

1. Dopo ogni inserimento che è stato fatto nella tabella “Visita”
 2. Aggiorna la tabella “Paziente”
 3. Imposta l’attributo “UltimaVisita” alla data di oggi
 4. Laddove il codice fiscale della tabella paziente è uguale a quello del paziente, che ho appena inserito
- Il NEW rappresenta il nuovo record appena inserito → Con la ‘notazione a punto’ posso riferirne i suoi attributi
- OSS: Se inserisco più di un record insieme, ogni volta la NEW identificherà la row che sto tentando di inserire nella tabella target

PISTOLESI-SQL

Sintassi TRIGGER multi-statement

```
DROP TRIGGER IF EXISTS nome_trigger;
DELIMITER $$

CREATE TRIGGER nome_trigger
[BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON TabellaTarget
FOR EACH ROW
BEGIN
    Bloccostruzioni1;
    Bloccostruzioni2;
    .
    .
    .
    BloccostruzioniN;
END $$

DELIMITER;
```

- L'azione svolta dai trigger è composta da tanti blocchi di istruzioni, che utilizzano la tipologia dichiarativo procedurale, terminata con punto e virgola

TRIGGER multi-statement

Scrivere un trigger che, ogni volta che viene inserita una nuova visita, se essa è mutuata, imposti l'attributo 'Ticket' in base alle fasce di reddito annue

- Ticket pari a euro 36.15 se reddito fra 0 ed euro 15000
- Ticket pari a euro 45.25 se reddito fra euro 15000 ed euro 25000
- Ticket pari a 50.00 euro se reddito oltre 25000 euro

Se la visita non è mutuata inserire NULL

```
1  DELIMITER $$ 
2  CREATE TRIGGER ImpostaTicket
3  BEFORE INSERT ON Visita
4  FOR EACH ROW
5
6  BEGIN
7      /* variabile contenente il reddito annuo del paziente */
8  SET @redditoAnnuo = (SELECT Reddito*12
9          FROM Paziente
10         WHERE CodFiscale = NEW.Paziente
11        );
12
13     /* controllo della fascia di reddito e settaggio del ticket*/
14  IF (NEW.Mutuata IS TRUE) THEN
15  IF (@redditoAnnuo BETWEEN 0 AND 14999) THEN
16      SET NEW.Ticket = 36.15;
17  ELSEIF (@redditoAnnuo BETWEEN 15000 AND 25000) THEN
18      SET NEW.Ticket = 45.25;
19  ELSE
20      SET NEW.Ticket = 50.00;
21  END IF;
22 ELSE
23     SET NEW.Ticket = NULL;
24 END IF;
25
26 END $$ 
27
28 DELIMITER ;
```



PISTOLESI-SQL

...CONTINUO TRIGGER multi-statement

- Considerando il testo dell'esercizio → "se essa è mutuata" / "Se la visita non è mutuata" → Rappresenta l'attributo mutuata della nuova visita che sto inserendo → Cioè la NEW.mutuata
- Per i tre casi di ticket mi serve un controllo di flusso
- RIGA 1 → Metto un delimitatore perché ho tanti statement → Finchè non trovo \$\$ tratto tutto come un unico statement
- L'evento, che è l'inserimento di una nuova visita è di tipo BEFORE → Perché prima di inserire una o più tuple in Visita, per ognuna di esse devo calcolarne il reddito annuo che faccio da RIGA 8 a RIGA 11 e lo metto in una variabile user-defined, la quale variabile viene deallocata a fine codice:
 - RIGA 8 → Prendo il reddito di un paziente calcolato su 12 mesi, visto che il reddito è mensile, laddove questo paziente è uguale al paziente che sto cercando di inserire (RIGA 10)
- Da RIGA 14 a RIGA 24 → Faccio il controllo sulla fascia di reddito e setto il ticket
- RIGA 14 → Verifico se la visita che ho inserito è mutuata
- Riga 15 → Controllo se il reddito annuo sta tra 0 e 15000 → Se lo è imposto il ticket a 36.15 (RIGA 16), altrimenti vado all'istruzione successiva
- Riga 17 → Controllo se il reddito annuo sta tra 15000 e 25000 → Se lo è imposto il ticket a 45.25 (RIGA 18), altrimenti vado all'istruzione successiva
- Riga 19 → Caso in cui il reddito annuo supera i 2500 → Se lo è imposto il ticket a 50.00 (RIGA 20), altrimenti vado all'istruzione successiva
- RIGA 22 → Caso in cui la visita che ho inserito non è mutuata
→ Se lo è imposto il ticket a NULL (RIGA 23), altrimenti vado all'istruzione successiva

OSS: Prima della riga 1 ci vuole **DROP TRIGGER IF EXISTS ImpostaTicket**

PISTOLESI-SQL

TRIGGER (attributo ridondante)

Implementare il trigger che mantiene aggiornato l'attributo ridondante nella tabella Paziente, contenente il numero di visite mutuate effettuate

```
1  DELIMITER $$  
2  
3  CREATE TRIGGER AggiornaVisiteMutuate  
4  AFTER INSERT ON Visita  
5  FOR EACH ROW  
6  BEGIN  
7  IF NEW.Mutuata IS TRUE THEN  
8      UPDATE Paziente  
9      SET VisiteMutuate = VisiteMutuate + 1  
10     WHERE CodFiscale = NEW.Paziente;  
11    END IF;  
12  END $$  
13  
14  DELIMITER ;
```

- Suppongo di aver già inserito l'attributo ridondante (perché può essere calcolato) VisiteMutuate
- Detto a parole: "Dopo aver fatto un inserimento in Visita (RIGA 4), per ogni riga inserita (RIGA 5) → Se ho inserito una visita mutata (RIGA 7) → Aggiorno la tabella Paziente (RIGA 8), impostando l'attributo VisiteMutuate a ciò che c'era prima + 1 che rappresenta questa appena inserita (RIGA 9), laddove il codice fiscale è quello del paziente di cui ho inserito la visita (RIGA 10)
OSS: Prima della riga 1 ci vuole **DROP TRIGGER IF EXISTS AggiornaVisiteMutuate**

QUANDO AGGIUNGO UN ATTRIBUTO RIDONDANTE (CALCOLATO) CI DEVE ESSERE SEMPRE DI MEZZO UN TRIGGER CHE LO AGGIORNA → QUESTO PERCHÉ IL TRIGGER GESTISCE LA RIDONDANZA

PISTOLESI-SQL

TRIGGER (business rule)

Ogni mese le visite non mutuate di un medico non devono superare quelle mutuate

```
1  DROP TRIGGER IF EXISTS checkValiditaVisita;
2
3  DELIMITER $$ 
4  CREATE TRIGGER checkValiditaVisita BEFORE INSERT ON Visita
5  FOR EACH ROW
6  BEGIN
7      DECLARE visite_mutuate_mese INTEGER DEFAULT 0;
8      DECLARE visite_non_mutuate_mese INTEGER DEFAULT 0;
9
10     SET visite_mutuate_mese =
11         ( SELECT COUNT(*) + IF(NEW.Mutuata = 1, 1, 0)
12             FROM Visita V
13             WHERE V.Medico = NEW.Medico
14                 AND V.Mutuata = 1
15                 AND MONTH(`Data`) = MONTH(CURRENT_DATE)
16                 AND YEAR(`Data`) = YEAR(CURRENT_DATE)
17         );
18
19     SET visite_non_mutuate_mese =
20         ( SELECT COUNT(*) - visite_mutuate_mese + IF(NEW.Mutuata = 0, 1, 0)
21             FROM Visita V
22             WHERE V.Medico = NEW.Medico
23                 AND MONTH(`Data`) = MONTH(CURRENT_DATE)
24                 AND YEAR(`Data`) = YEAR(CURRENT_DATE)
25
26     ▼ IF visite_non_mutuate_mese >= visite_mutuate_mese THEN
27         SIGNAL SQLSTATE '45000'
28         SET MESSAGE_TEXT = 'Limite massimo visite mutuate superato';
29     ▲ END IF;
30 END $$ 
31 DELIMITER :
```

- La BUSINESS RULE mi deve rappresentare un vincolo di integrità generico, in questo caso temporale → Il vincolo, sotto alcune condizioni, deve poter bloccare l'inserimento della visita

- RIGA 10 → Imposto le visite mutuate del mese (in corso) così:

- RIGA 12, RIGA 13 → Prendo le visite dei medici, il cui medico è quello che sto inserendo, che è gestito dal trigger (NEW)
- RIGA 14 → Che ha fatto visite mutuate
- RIGA 15, RIGA 16 → Nel mese in corso (dell'anno corrente)
- RIGA 11 → Ne proietto il conteggio sotto le condizioni precedenti e → Ci sommo 1 se la nuova visita inserita, che è gestita dal trigger (NEW), è mutuata, altrimenti → Ci sommo 0, che mi rappresenta che la nuova visita inserita non è mutuata

OSS: Questa somma, mi permette di inserire nel conto delle visite mutuate anche quelle che sto inserendo

- RIGA 19 → Imposto le visite non mutuate del mese (in corso) così:

- RIGA 21, RIGA 22 → Prendo le visite dei medici, il cui medico è quello che sto inserendo, che è gestito dal trigger (NEW)
- RIGA 23, RIGA 24 → Nel mese in corso (dell'anno corrente)

PISTOLESI-SQL

...CONTINUO TRIGGER (business rule)

- RIGA 20 → Ne proietto il conteggio sotto le condizioni precedenti e → Ci sottraggo le visite mutuate calcolate da RIGA 10 a RIGA 17, per escluderle e poi → Ci sommo 1 se la nuova visita inserita, che è gestita dal trigger (NEW), non è mutuata, altrimenti → Ci sommo 0, che mi rappresenta che la nuova visita inserita è mutuata
- Da RIGA 26 a RIGA 29 ho la **BUSINESS RULE** che mi rappresenta il fatto che → Se con le visite non mutuate che sto inserendo vado ad egualare quelle mutuate → Allora non posso andare più ad inserire, perché non posso superare quelle mutuate → La INSERT viene abortita dal comando SIGNAL...

Aggiunta ridondanza e assegnamento (con ALTER TABLE - ADD COLUMN)

Aggiungere un attributo ridondante alla tabella Medico che contenga il numero di visite effettuate, di ciascun medico

```
ALTER TABLE Medico
ADD COLUMN TotaleVisite INT DEFAULT 0 AFTER Citta
}
1
UPDATE Medico
SET TotaleVisite =
(
    SELECT COUNT(*)
    FROM Visita V
    WHERE V.Medico = Matricola
);
```

2

1. Si altera la tabella Medico aggiungendo una nuova colonna TotaleVisite, di tipo intero e con valore di default 0, e la piazzo dopo l'attributo Citta
2. In Totale visite ci metterò il conteggio delle visite di ciascun medico

OSS:

- All'inserimento di una nuova visita, la ridondanza non è più aggiornata
- Si intende come attributo ridondante, perché può essere calcolato, tramite una query

PISTOLESI-SQL

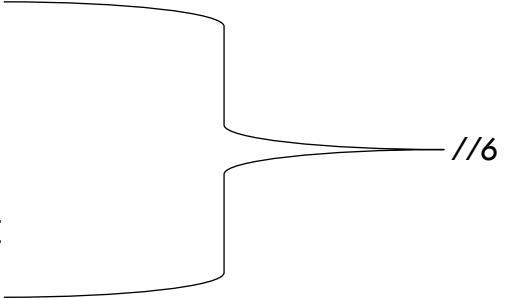
Ridondanza giornaliera (e periodica) → Recurring (EVENT - ON SCHEDULE EVERY 1 DAY -(STARTS) - DO)

Creare e mantenere giornalmente aggiornata una ridondanza nella tabella Medico contenente, per ciascuno, il numero totale di visite effettuate

CASO PERIODICO

```
DROP EVENT IF EXISTS AggiornaTotaliVisite          //0
CREATE EVENT AggiornaTotaliVisite                   //1
ON SCHEDULE EVERY 1 DAY
DO
    UPDATE Medico
    SET TotaleVisite = TotaleVisite +
        (
            SELECT COUNT (*)
            FROM Visita V
            WHERE V.Medico = Matricola
            AND
            V.Data = CURRENT_DATE
        );

```



- Suppongo di aver già inserito l'attributo ridondante TotaleVisite (pag. precedente)
 - Si crea un evento (//1) verificando prima che non esista già (//0)
 - L'evento viene eseguito ogni giorno a partire da quando viene creato (//2)
 - Ogni giorno l'evento esegue (//3) → L'aggiornamento della tabella Medico (//4) modificando il numero totale di visite con la somma tra quelle precedenti (//5) + quelle attuali, cioè le visite dei medici che hanno visitato in data odierna (//6)

CASO PERIODICO CON INIZIO PERIODICITÀ

```
DROP EVENT IF EXISTS AggiornaTotaliVisite
CREATE EVENT AggiornaTotaliVisite
ON SCHEDULE EVERY 1 DAY
STARTS '2020-04-20 23:55:00'                      //1
DO
    UPDATE Medico
    SET TotaleVisite = TotaleVisite +
        (
            SELECT COUNT (*)
            FROM Visita V
            WHERE V.Medico = Matricola
            AND
            V.Data = CURRENT_DATE
        );

```

//1 → In questo caso ho che l'evento ha la sua prima esecuzione in un certo orario di una certa data

PISTOLESI-SQL

Sintassi ON SCHEDULE (Periodico)

ON SCHEDULE EVERY numero DAY | MONTH | YEAR | SECOND | MINUTE | HOUR

Il 'numero' rappresenta la periodicità con cui avviene l'esecuzione, relativa a:

- giorni / mesi / anni / secondi / minuti / ore

Sintassi ON SCHEDULE - STARTS (Periodico - inizio)

ON SCHEDULE EVERY numero DAY | MONTH | YEAR | SECOND | MINUTE | HOUR

STARTS "AAAA-MM-GG hh:mm:ss"

STARTS... → Indica che da quel momento in poi si ripete l'evento con la periodicità scelta

Sintassi ON SCHEDULE - STARTS - ENDS (Periodico - inizio/fine)

ON SCHEDULE EVERY numero DAY | MONTH | YEAR | SECOND | MINUTE | HOUR

STARTS "AAAA-MM-GG hh:mm:ss" ENDS "AAAA-MM-GG hh:mm:ss"

STARTS...ENDS... → Indica che l'evento viene eseguito per quell'intervallo di tempo regolare, ed è ripetuto con la periodicità scelta → Si chiama RECURRING EVENT

Sintassi ON SCHEDULE di EVENT a singolo scatto

ON SCHEDULE AT "AAAA-MM-GG hh:mm:ss" +

INTERVAL numero DAY | MONTH | YEAR | SECOND | MINUTE | HOUR

[ON COMPLETION PRESERVE]

- ON SCHEDULE AT... + INTERVAL ... → Istante di esecuzione dell'evento
- ON COMPLETION PRESERVE → Indica che il codice viene stoccatto nel DBMS; quindi, viene mantenuto
- Questo tipo di evento scatta una sola volta da un certo istante → Quindi non ricorre periodicamente → Si chiama EVENT A SINGOLO SCATTO

Sintassi CREATE TABLE

CREATE TABLE NomeTabella(

Attributo 1,

.

.

.

Attributo n,

)ENGINE=InnoDB DEFAULT CHARSET=latin1;

PISTOLESI-SQL

Materialized view (introduzione)

- La Materialized view è una tabella (ridondante) stoccatà in memoria, che contiene un risultato pre-calcolato di una query
- A differenza di una view, questa non è ricalcolata ogni volta che viene usata
- A differenza di una temporary table non viene distrutta al momento del logout
- Ci sono due modalità di aggiornamento per le materialized view:
 1. **FULL REFRESH** → Si aggiorna la materialized view da 0
 2. **INCREMENTAL REFRESH** → Della materialized view si aggiorna solo la parte ‘non più aggiornata’
- Le politiche di full refresh sono di 3 tipi e sono:
 1. **ON DEMAND REFRESH** → Viene rinnovato il contenuto della materialized view “su richiesta”, in base alla chiamata di una stored procedure → La materialized view viene cancellata e ricostruita dalla stored procedure
 - o Si mettono in SYNC i dati al momento della richiesta
 2. **IMMEDIATE REFRESH** → Viene aggiornata immediatamente la materialized view a seguito della modifica delle tabelle su cui si basa la materialized view → Si fa tramite l’utilizzo dei trigger
 - o Si mettono in SYNC i dati immediatamente
 3. **DEFERRED REFRESH** → Viene rinnovata la materialized view rispettando una periodicità, dettata da un event
 - o Si posticipa SYNC dei dati, in base ad un event

Materialized view (esempio)

Creare una materialized view MV_RESOCOMTO avente funzione di reporting, contenente, per ogni specializzazione medica della clinica, il numero di visite effettuate, il numero di pazienti visitati, l’incasso totale relativo al mese in corso, e la matricola del medico che ha visitato più pazienti.

Implementare l’on demand refresh

1-CREAZIONE DELLA MATERIALIZED VIEW (COME TABELLA)

```
1 ▼ CREATE TABLE MV_RESOCOMTO (
2   Specializzazione CHAR(100) NOT NULL,
3   Visite INT(11) NOT NULL,
4   Pazienti INT(11) NOT NULL,
5   IncassoMese DOUBLE NOT NULL,
6   MedicoPiuPazienti CHAR(100) NOT NULL,
7   PRIMARY KEY (Specializzazione)
8 ▲ ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- Dopo aver creato la tabella procederò al suo popolamento, tramite delle viste, dalle quali prenderò il risultato e lo inserirò negli attributi della tabella stessa
- Per gli attributi ‘IncassoMese’ e ‘MedicoPiuPazienti’ creerò due viste separate, mentre per il restante degli attributi, utilizzerò dei semplici operatori di aggregazione



PISTOLESI-SQL

... CONTINUO Materialized view (esempio)

2-CALCOLO DELL'INCASSO TOTALE RELATIVO AL MESE IN CORSO

```
10 -- Incasso mensile di ciascuna specializzazione
11
12 CREATE OR REPLACE VIEW IncassoMese AS
13 SELECT
14     M.Specializzazione,
15     SUM(M.Parcella) AS Incasso
16 FROM
17     Visita V
18     INNER JOIN
19     Medico M ON V.Medico = M.Matricola
20 WHERE
21     MONTH(V.`Data`) = MONTH(CURRENT_DATE)
22     AND YEAR(V.`Data`) = YEAR(CURRENT_DATE)
23 GROUP BY
24     M.Specializzazione;
```

RIGA 15 → Calcolo l'incasso di tutti i medici → Che hanno visitato (da RIGA 17 a RIGA 19) → In questo mese, dell'anno corrente (RIGA 21, RIGA 22) → E l'incasso lo prendo relativo ad ogni specializzazione (RIGA 23, RIGA 24)

3-CALCOLO DELLA MATRICOLA DEL MEDICO CHE HA VISITATO PIÙ PAZIENTI

```
17 -- Medico con più pazienti visitati
18
19 CREATE OR REPLACE VIEW MediciPiuPazienti AS
20 SELECT
21     M.Matricola AS MedicoMaxPazienti,
22     M.Specializzazione
23 FROM
24     Visita V
25     INNER JOIN
26     Medico M ON V.Medico = M.Matricola
27 GROUP BY
28     M.Matricola,
29     M.Specializzazione
30 HAVING
31     COUNT(DISTINCT V.Paziente) > ALL
32     (
33         SELECT
34             COUNT(DISTINCT V2.Paziente)
35         FROM
36             Visita V2
37             INNER JOIN
38             Medico M2 ON V2.Medico = M2.Matricola
39             WHERE
40                 M2.Specializzazione = M.Specializzazione AND M2.Matricola <> M.Matricola
41             GROUP BY
42                 M2.Matricola
43     )
44 ;
```

RIGA 29 → Per ciascuna specializzazione e per ciascun medico (RIGA 28) → Considero la matricola del medico (RIGA 21) → Che ha fatto visite (da RIGA 24 a RIGA 26) → E che ha il conteggio sui pazienti visitati, senza duplicati (RIGA 31) maggiore → Del conteggio di tutte le visite dei pazienti, fatte da un medico della stessa specializzazione, ma che sia diverso (da RIGA 32 a RIGA 43) → Con <> non mi riferisco allo stesso medico

PISTOLESI-SQL

... CONTINUO Materialized view (esempio)

4-INSERIMENTO NELLA MV DEGLI ATTRIBUTI CALCOLATI PRIMA + GLI ATTRIBUTI CHE RAPPRESENTATNO IL NUMERO DI VISITE E IL NUMERO DI PAZIENTI VISITATI

```
28 INSERT INTO MV_RESOCOMTO
29 SELECT
30     M.Specializzazione,
31     COUNT(*) AS Visite,
32     COUNT(DISTINCT V.Paziente) AS Pazienti,
33     IM.Incasso AS IncassoMeseCorrente,
34     MPP.MedicoMaxPazienti AS MedicoPiuPazienti
35 FROM
36     Visita V
37     INNER JOIN
38         Medico M ON M.Matricola = V.Medico
39     NATURAL JOIN
40         IncassoMese IM
41     NATURAL JOIN
42         MediciPiuPazienti MPP
43 GROUP BY
44     M.Specializzazione,
45     IM.Incasso,
46     MPP.MedicoMaxPazienti;
```

Da RIGA 30 a RIGA 34 → Inserisco negli attributi della MV, gli attributi trovati precedentemente ai punti 2 e 3, e ci inserisco anche degli attributi che calcolo qua (RIGA 31, RIGA 32) → Che rappresentano il numero di viste effettuate e il numero di pazienti visitati dai medici che hanno visitato più pazienti per ogni specializzazione, con il relativo incasso mensile

Da RIGA 36 a RIGA 42 → Ho il JOIN con le due view che funziona così:

- Il DMBS cerca tra le view memorizzate quelle che hanno l'ID IncassoMese e MediciPiuPazienti
- Prende queste view e le compila
- Poi al posto di esse ci mette i relativi result set

→ A fianco ad ogni visita ci metto l'incasso mensile e il medico con più pazienti (se è presente)

ON DEMAND REFRESH (FULL)

- Per questo tipo di REFRESH mi basterà creare una procedura che:

- Prende la MV vecchia
- La cancella
- Ci mette i nuovi dati, semplicemente inserendoli tramite ciò che è stato fatto al punto 4

OSS:

- Una volta creata la procedura basterà semplicemente chiamarla e il codice verrà ricompilato ed eseguito
- I punti 1, 2, 3 non ci sono perché si sostituisce solamente il punto 4



PISTOLESI-SQL

... CONTINUO Materialized view (esempio)

```
1 ▼DROP PROCEDURE IF EXISTS refresh_MV_Resoconto;
2
3 DELIMITER $$  

4
5 CREATE PROCEDURE refresh_MV_Resoconto (OUT esito INTEGER)
6 ▼BEGIN
7
8     -- esito vale 1 se si verifica un errore
9     DECLARE esito INTEGER DEFAULT 0;  

10
11    DECLARE EXIT HANDLER FOR SQLEXCEPTION ← in caso di errori gravi esegue il
12    BEGIN                                         rollback riportando la materialized view
13        ROLLBACK;                                allo stato precedente alla chiamata
14        SET esito = 1;
15        SELECT 'Si è verificato un errore: materialized view non aggiornata.';  

16 ▲ END;  

17
18    -- flushing della materialized view
19    TRUNCATE TABLE MV_RESOCOMTO;
20    -- full refresh
21    INSERT INTO MV_RESOCOMTO
22    SELECT
23        M.Specializzazione,
24        COUNT(*) AS Visite,
25        COUNT(DISTINCT V.Paziente) AS Pazienti,
26        IM.Incasso AS IncassoMeseCorrente,
27        MMP.MedicoMaxPazienti AS MedicoPiuPazienti
28
29    FROM
30        Visita V
31        INNER JOIN
32        Medico M ON V.Medico = M.Matricola
33        NATURAL JOIN
34        IncassoMese IM
35        NATURAL JOIN
36        MediciPiuPazienti MMP
37        GROUP BY
38            M.Specializzazione;  

39
40 ▲ END $$  

41
42 DELIMITER;
```

- Il flushing a RIGA 19 mi serve per gettare il contenuto della materialized view e ricalcolarlo from scratch, cioè da 0
- Da RIGA 22 a RIGA 38 è lo stesso codice del punto 4, solamente che qua è inserito dentro la procedura
OSS: Dopo RIGA 38 mancherebbe la parte di codice
IM.Incasso,
MMP.MediciPiuPazienti;

PISTOLESI-SQL

Materialized view (IMMEDIATE / ON DEMAND / DEFERRED REFRESH)

Inserimento di un paziente con il numero di visite fatte e la data dell'ultima visita
(con le tre tipologie di refresh)

CREAZIONE DELLA MATERIALIZED VIEW

```
CREATE TABLE MATERIALIZED_VIEW(
    Paziente CHAR(100) NOT NULL,
    NumVisite INT(11) NOT NULL DEFAULT 0,
    UltimaVisita DATE,
    PRIMARY KEY (Paziente)
) ENGINE = InnoDB DEFAULT CHARSET=latin1;
```

Nell'attributo UltimaVisita non metto nessun vincolo → Perché se voglio inserire un paziente senza che esso faccia visite → Lo inserirò nella MV con l'attributo UltimaVisita=NULL

IMMEDIATE REFRESH (SYNC)

```
DELIMITER $$
```

```
DROP TRIGGER IF EXISTS immediate_refresh_mv1 $$  
CREATE TRIGGER immediate_refresh_mv1  
AFTER INSERT ON Visita  
FOR EACH ROW  
BEGIN  
  
    UPDATE MATERIALIZED_VIEW  
    SET NumVisite = NumVisite + 1,  
        UltimaVisita = CURRENT_DATE  
    WHERE Paziente = NEW.Paziente;  
  
END $$  
DELIMITER ;
```

1

- Dopo ogni inserimento in Visita il trigger mi aggiornerà la MV:
 - Aumentando di 1 il numero di visite
 - Aggiornando la data dell'ultima visita a quella dell'inserimento
- L'aggiornamento viene fatto laddove il paziente della visita è quello che sto cercando di inserire (NEW.Paziente)

```
DROP TRIGGER IF EXISTS immediate_refresh_mv2 $$  
CREATE TRIGGER immediate_refresh_mv2  
AFTER INSERT ON Paziente  
FOR EACH ROW  
BEGIN  
  
    INSERT INTO MATERIALIZED_VIEW(Paziente)  
    VALUES (NEW.CodFiscale);  
  
END $$  
DELIMITER ;
```

2

- Dopo ogni inserimento in Paziente il trigger mi aggiornerà la MV:
 - Mettendo nel codice fiscale relativo al paziente della MV → Il codice fiscale del paziente che sto cercando di inserire (NEW.CodFiscale)

Il REFRESH oltre che sull'inserimento di una nuova visita [1] lo devo fare anche sull'inserimento di un nuovo paziente [2] → Faccio così per fini statistici perché
→ Se inserisco una nuova visita di un paziente, so che devo andare a vedere sulla MV i suoi dati, anche se magari non ha fatto visite → Quindi li ho lo stesso sulla MV



PISTOLESI-SQL

... CONTINUO Materialized view (IMMEDIATE / ON DEMAND / DEFERRED REFRESH)

ON DEMAND REFRESH (FULL)

```
DROP PROCEDURE IF EXISTS on_demand_refresh_mv $$  
CREATE PROCEDURE on_demand_refresh_mv()  
BEGIN  
  
    TRUNCATE MATERIALIZED_VIEW; → 0  
  
    INSERT INTO MATERIALIZED_VIEW  
    SELECT P.CodFiscale, → 1  
          IF(V.Paziente IS NULL, 0, COUNT(*)), → 2  
          IF(V.Paziente IS NULL, NULL, MAX(V.Data)) → 3  
    FROM Visita V  
    RIGHT OUTER JOIN  
        Paziente P ON V.Paziente = P.CodFiscale  
    GROUP BY P.CodFiscale; → 4  
  
END $$
```

- Per questo tipo di refresh si butta via il contenuto vecchio tramite (0), e attraverso la stored procedure si aggiorna la MV (rifacendola da capo)
- 4 → Faccio il JOIN esterno DX → Quindi i pazienti che non hanno fatto visite avranno la parte, dei campi di Visita, posta a NULL, e lo faccio raggruppando per ciascun paziente
- 1 → Nell'attributo Paziente della MV ci andrà il codice fiscale del paziente all'inserimento di una nuova visita
- 2 → Nell'attributo NumVisite della MV ci andrà 0 se non ho la visita, altrimenti ci andrà COUNT(*) → Che rappresenta il conteggio sulle visite di quel paziente
- 3 → Nell'attributo UltimaVisita della MV ci andrà NULL se non ho la visita, altrimenti ci andrà MAX(V.Data) → Cioè la visita massima della data che rappresenta quella più recente → Di conseguenza l'ultima visita

DEFERRED REFRESH (FULL)

```
DROP EVENT IF EXISTS deferred_refresh_mv $$  
CREATE EVENT deferred_refresh_mv  
ON SCHEDULE EVERY 1 WEEK → 1  
DO  
BEGIN  
  
    CALL on_demand_refresh_mv(); → 2  
  
END $$  
  
DELIMITER ;
```

Per questo tipo di refresh, basta che richiami la procedura scritta nell'ON DEMAND REFRESH, tramite la CALL (2), con cadenza settimanale (1) → Che è il periodo scelto per il deferred, in questo caso

PISTOLESI-SQL

Materialized view – INCREMENTAL REFRESH (concetto)

- Questa tipologia di refresh permette di avere i dati di una MV aggiornati fino ad un certo istante di tempo, perché cancellando, modificando e inserendo roba sulla MV → Questa si disallinea con la realtà

- Meccanismo:

- Fra un aggiornamento e l'altro tutte le modifiche alle tabelle devono essere inserite nel **LOG** → Che è a tutti gli effetti una tabella, e che dovrà contenere le informazioni necessarie utili per l'aggiornamento della MV
- Poi dobbiamo combinare i dati della MV con i dati che saranno presenti nella log table, in modo da aggiornare e mantenere in sync tutto quanto
- Per la processazione del log ci sono più modi:
 - **PARTIAL REFRESH** → Processo una parte del log e lo trasferisco sulla MV
 - **COMPLETE REFRESH** → Processo tutto il log e lo trasferisco sulla MV
 - **REBUILD** → Faccio un ricalcolo completo del log (una specie di full refresh)

Materialized view (PARTIAL REFRESH)

Inserimento di un paziente con il numero delle visite fatte e la data dell'ultima visita

CREAZIONE DELLA MATERIALIZED VIEW

```
CREATE TABLE MATERIALIZED_VIEW(  
    Paziente CHAR(100) NOT NULL,  
    NumVisite INT(11) NOT NULL DEFAULT 0,  
    UltimaVisita DATE,  
    PRIMARY KEY (Paziente)  
) ENGINE = InnoDB DEFAULT CHARSET=latin1;
```

- Come nell'esempio precedente, solamente che qua sto lavorando sull'incremental refresh
- Anche qua, come nell'esempio precedente mi creo la materialized view

CREAZIONE DELLA LOG TABLE

```
CREATE TABLE LOG_TABLE(  
    Paziente CHAR(100) NOT NULL,  
    DataVisita DATE  
) ENGINE = InnoDB DEFAULT CHARSET=latin1;
```

- Il paziente coinvolto e la data della sua visita sono le informazioni minimali scelte
- Queste informazioni sono quelle necessarie per aggiornare la MV



PISTOLESI-SQL

...CONTINUO Materialized view (PARTIAL REFRESH)

CREAZIONE DEL TRIGGER DI PUSH NELLA LOG TABLE

```
DELIMITER $$  
  
DROP TRIGGER IF EXISTS push_1 $$  
CREATE TRIGGER push_1  
AFTER INSERT ON Visita  
FOR EACH ROW  
BEGIN  
  
    INSERT INTO LOG_TABLE  
    VALUES(NEW.Paziente, CURRENT_DATE);  
  
END $$
```

1


```
DROP TRIGGER IF EXISTS push_2 $$  
CREATE TRIGGER push_2  
AFTER INSERT ON Paziente  
FOR EACH ROW  
BEGIN  
  
    INSERT INTO LOG_TABLE  
    VALUES(NEW.CodFiscale, NULL);  
  
END $$
```

2

- Dopo ogni inserimento in Visita il trigger di push aggiornerà la LOG table:

- Inserendoci i dati del nuovo paziente (NEW.Paziente), con la data al momento dell'inserimento a (CURRENT_DATE)

- È come se propagassi un sottoinsieme delle informazioni di una visita

- Dopo ogni inserimento in Paziente il trigger di push aggiornerà la LOG table:

- Inserendoci i dati del nuovo paziente (NEW.CodFiscale), con la data della visita attuale al momento dell'inserimento a NULL
→ Perché sto inserendo nel LOG un paziente, non una visita (che ovviamente ha una data)

L'aggiornamento della LOG table oltre che sull'inserimento di una nuova visita [1] lo devo fare anche sull'inserimento di un nuovo paziente [2] sulla MV

PARTIAL REFRESH PARTE 1

```
DROP PROCEDURE IF EXISTS on_demand_refresh_mv $$  
CREATE PROCEDURE on_demand_refresh_mv(_up_to DATE) → 0  
BEGIN  
  
    REPLACE INTO MATERIALIZED_VIEW  
    -- aggregazione del log  
    WITH aggregated_log AS (  
        SELECT LT.Paziente,  
            -- con SUM(IF), anziché COUNT(*), scarto le insert di nuovi pazienti  
            SUM(IF(LT.DataVisita IS NULL, 0, 1)) AS NuoveVisite, → 1  
            MAX(LT.DataVisita) AS NuovaUltima  
        FROM LOG_TABLE LT  
        WHERE LT.DataVisita <= _up_to → 2  
        GROUP BY LT.Paziente  
    )
```

"Replace into" esegue un inserimento
se non collide sulla chiave primaria, altrimenti
esegue un update

0 → Il parametro d'ingresso '_up_to' indicherebbe "fino a" una certa data, e ciò rappresenta che processo solo un pezzo del LOG, laddove prenderò i dati a me necessari fino a che non supero la data presa in ingresso (2)

- Per l'inserimento della MV utilizzo la CTE 'aggregated_log'

1 → Controllo se l'inserimento che ho fatto è quello di un paziente → Cioè verifico se la data della visita è NULL → Per come ho impostato il trigger di push, la DataVisita=NULL ce l'ha il paziente → Se così fosse ci metto 0, perché non devo sommare i pazienti ma le visite, altrimenti ci metto 1, cioè rappresenta l'inserimento di una nuova visita

PISTOLESI-SQL

...CONTINUO Materialized view (PARTIAL REFRESH)

PARTIAL REFRESH PARTE 2

```
-- segue immediatamente la CTE aggregated_log
SELECT MV.Paziente, _____ → 1
    IF(MV.Paziente IS NULL,
       AL.NuoveVisite,
       MV.NumVisite + IF(AL.NuovaUltima IS NULL, 0, AL.NuoveVisite))
    ),
    IF(MV.Paziente IS NULL,
       IFNULL(AL.NuovaUltima, NULL),
       AL.NuovaUltima
    )
FROM MATERIALIZED_VIEW MV
RIGHT OUTER JOIN
aggregated_log AL USING(Paziente);
-- se un record di MV fa join, vuol dire che nel log
-- c'è un aggiornamento che coinvolge quel record;
-- se un record di MV non fa join, vuol dire che
-- in MV quel record (paziente) ancora non c'era

-- flushing della parte di log processata
DELETE FROM LOG_TABLE LT
WHERE LT.DataVisita <= _up_to; → 5

END $$
```

Questa Query è quella che segue la CTE

1 → In 'Paziente' dentro la MV, ci metto il codice fiscale del paziente ottenuto

2 → Caso in cui mi riferisco ad una nuova visita, visto che inserirò nell'attributo 'NumVisite' della MV, e controllo se MV.Paziente è NULL:

- Caso NULL: Vuol dire che non ho fatto JOIN con i record della LOG table
→ Il che significa che quello è un nuovo paziente, quindi di lui prendo AL.NuoveVisite, cioè le sue nuove visite visto che mi riferisco a visita
- Caso non NULL: Vuol dire che ho fatto JOIN con i record della LOG table
→ Il che significa che quello non è un nuovo paziente ma è un paziente da aggiornare → Allora aggiungo all'attributo MV.NumVisite:
 - 0, se l'ultima visita che ha fatto non è presente
 - AL.NuoveVisite, se è presente un ultima visita fatta, quindi ne prendo le sue nuove visite visto che mi sto riferendo a visita

3 → Caso in cui mi riferisco ad un nuovo paziente, visto che inserirò nell'attributo 'UltimaVisita' della MV, e controllo se MV.Paziente è NULL:

- Caso NULL: Vuol dire che non ho fatto JOIN con i record della LOG table
→ Quindi quello è un nuovo paziente → Controllo poi, se AL.NuovaUltima è NULL, e vuol dire che vale questo (cioè NULL), visto che non ho jointato ed è un nuovo paziente → Altrimenti vale NULL → E di conseguenza va bene perché mi sto riferendo ad un paziente, altrimenti ci metto NULL
- Caso non NULL: Vuol dire che ho fatto JOIN con i record della LOG table
→ Il che significa che quello non è un nuovo paziente ma è un paziente da aggiornare → Ci metto AL.NuovaUltima

4 → Spiegazione del JOIN presente sotto di esso!

5 → Elimino la parte del log processato visto che la MV viene aggiornata fino alla data scelta (**COMPLETE REFRESH** → Metto CURRENT_DATE al posto di _up_to)

PISTOLESI-SQL

Materialized view (PARTIAL REFRESH, COMPLETE REFRESH, REBUILD)

Creare una materialized view MV_RESOCOMTO avente funzione di reporting, contenente, per ogni specializzazione medica della clinica, il numero di visite effettuate, il numero di nuovi pazienti visitati, l'incasso totale relativo al mese in corso, e la matricola del medico che ha visitato più pazienti. Implementare: i) il complete incremental refresh; ii) il partial incremental refresh; iii) il rebuild. Per semplicità, implementare solamente il push relativo all'inserimento

LOG TABLE

```
1 CREATE TABLE RESOCOMTO_LOGC
2
3     Istante TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
4     Medico CHAR(50) NOT NULL,
5     Paziente CHAR(50) NOT NULL,
6     NuovoPaz TINYINT(1) NOT NULL,
7     PRIMARY KEY (Istante)
8
9 ^) ENGINE = InnoDB DEFAULT CHARSET = latin1;
```

Per semplicità viene esclusa la creazione della MV

TRIGGER DI PUSH

```
1 DELIMITER $$
2
3 CREATE TRIGGER Push_Resoconto_Log
4 AFTER INSERT ON Visita
5 FOR EACH ROW
6 BEGIN
7     DECLARE visite_prec INTEGER DEFAULT 0;
8
9     SELECT COUNT(*) - 1
10    FROM Visita V
11   WHERE V.Medico = NEW.Medico
12     AND V.Paziente = NEW.Paziente;
13
14    INSERT INTO MV_RESOCOMTO_LOG
15    VALUES (
16        CURRENT_TIMESTAMP, NEW.Paziente,
17        NEW.Medico, IF(visite_prec > 0, 0, 1)
18    );
19 END $$
20 DELIMITER ;
```



PISTOLESI-SQL

...CONTINUO Materialized view (PARTIAL REFRESH, COMPLETE REFRESH, REBUILD)

REBUILD

```
1 DROP PROCEDURE IF EXISTS incremental_refresh;
2
3 DELIMITER $$
```

4

```
5 CREATE PROCEDURE incremental_refresh(
6     IN metodo VARCHAR(20),
7     IN istante_soglia TIMESTAMP,
8     OUT esito INTEGER)
```

9

```
10 BEGIN
11     IF metodo = 'rebuild' THEN
12         BEGIN
13             CALL refresh_MV_Resoconto(@es);
14             IF @es = 1 THEN
15                 SET esito = 1;
16             END IF;
17         END;
```

COMPLETE REFRESH /PARTIAL REFRESH

```
18 ELSEIF metodo = 'full refresh' OR metodo = 'partial refresh'
19 BEGIN
20     REPLACE INTO MV_Resoconto
21     SELECT D.Specializzazione,
22             SUM(Visite),
23             SUM(Pazienti),
24             SUM(Incasso),
25             MPP.MedicoMaxPazienti
26     FROM
27     (
28     SELECT M.Specializzazione,
29             COUNT(*) AS Visite,
30             SUM(NuovoPaz) AS Pazienti,
31             SUM(M.Parcella) AS Incasso
32     FROM RESOCOMTO_LOG RL
33             INNER JOIN
34             Medico M ON RL.Medico = M.Matricola
35             WHERE Istante <= IF(metodo='full refresh',
36                                 CURRENT_TIMESTAMP,
37                                 istante_soglia)
38             GROUP BY M.Specializzazione
39             UNION ALL
40             SELECT Specializzazione,
41                     Visite,
42                     Pazienti,
43                     Incasso
44             FROM MV_RESOCOMTO
45             ) AS D
46             NATURAL JOIN
47             MediciPiuPazienti MPP
48             GROUP BY D.Specializzazione;
```

49

```
50
51     IF metodo = 'full refresh' THEN
52         TRUNCATE TABLE RESOCOMTO_LOG;
53     ELSE
54         DELETE FROM RESOCOMTO_LOG
55             WHERE Istante <= istante_soglia;
56     END IF;
```

57

```
58
59     END;
60     ELSE
61         SET esito = 1;
62     END IF;
63 END $$ DELIMITER ;
```

PISTOLESI-SQL

WINDOW FUNCTIONS: clausola OVER() su funzione aggregata

Scrivere una query che indichi, per ogni cardiologo, la matricola, la parcella, e la parcella media della sua specializzazione

```
SELECT M.Matricola, M.Parcella  
      AVG (M.Parcella) OVER()  
FROM Medico M  
WHERE M.Specializzazione = 'Cardiologia';
```

ESEMPIO APPLICATO

Matricola	Parcella	Avg(M.Parcella)
014	180	230.0000
015	230	230.0000
016	220	230.0000
017	260	230.0000
018	260	230.0000

- La clausola OVER affianca ad ogni medico cardiologo, con la relativa parcella → La media delle parcelle di tutti i medici cardiologi → I quali medici cardiologi fanno parte di un result set → Quindi la media viene chiamata ogni per tutti i record del result set
- AVG (M.Parcella) OVER() → Equivale a:

```
SELECT AVG(M.Parcella)  
FROM Medico M  
WHERE Specializzazione='Cardiologia';
```
- OVER() viene affiancata ad ogni record di un determinato result set, ottenuto escludendo la clausola OVER()

WINDOW FUNCTIONS: clausola OVER(PARTITION BY NomePartizione) su funzione aggregata

Scrivere una query che indichi, per ogni medico, la matricola, la specializzazione, e la parcella media della sua specializzazione

```
SELECT M.Matricola, M.Specializzazione, M.Parcella  
      AVG(M.Parcella) OVER(  
          PARTITION BY M.Specializzazione  
          )  
FROM Medico M;
```

- La media della parcella è calcolata sulla **PARTITION** → Che è l'insieme dei record che hanno la stessa specializzazione del medico proiettato → Ogni volta che mi arriva il record da proiettare per quella specializzazione → Ne calcolo la media

ESEMPIO APPLICATO

Matricola	Specializzazione	Parcella	Avg(M.Parcella) OVER(
017	Cardiologia	260	230.0000
014	Cardiologia	180	230.0000
016	Cardiologia	220	230.0000
018	Cardiologia	260	230.0000
015	Cardiologia	230	230.0000
005	Gastroenterologia	130	120.0000
013	Gastroenterologia	110	120.0000
001	Otorinolaringoiatria	100	175.0000
003	Otorinolaringoiatria	200	175.0000
002	Otorinolaringoiatria	150	175.0000
004	Otorinolaringoiatria	250	175.0000

**LA PARTITION È L'INSIEME DI RECORD SUI QUALI SI APPLICANO LE FUNZIONI DI AGGREGAZIONE (E NON).
LA PARTITION PUÒ ANCHE ESSERE DEFINITA SU ATTRIBUTI NON PROIETTATI, BASTA CHE FACCIANO PARTE AL RESULT SET PRODOTTO DALLA QUERY SENZA LA CLAUSOLA OVER**

PISTOLESI-SQL

WINDOW FUNCTIONS: clausola OVER applicazioni

- Questa clausola applica funzioni di tipo **aggregate e non aggregate** ad una partition
- Le funzioni non aggregate:

- Associano a ciascun record di un result set un valore ottenuto dalla partition senza fondere i record in un'informazione riepilogativa → Cioè, senza comprimere nulla, dato che il loro risultato non è una conseguenza di una aggregazione
- Lavorano sull'intera partition → Considerandone tutti i record
- Lavorano su **frame** → Che sono sottoinsiemi della partition → Considerandone per default i record dall'inizio della partition fino alla current row → Specificando una ORDER BY all'interno della clausola OVER

AGGREGATE FUNCTIONS UTILIZZABILI CON OVER

Name	Description
<u>AVG()</u>	Return the average value of the argument
<u>BIT_AND()</u>	Return bitwise AND
<u>BIT_OR()</u>	Return bitwise OR
<u>BIT_XOR()</u>	Return bitwise XOR
<u>COUNT()</u>	Return a count of the number of rows returned
<u>COUNT(DISTINCT)</u>	Return the count of a number of different values
<u>GROUP_CONCAT()</u>	Return a concatenated string
<u>JSON_ARRAYAGG()</u>	Return result set as a single JSON array
<u>JSON_OBJECTAGG()</u>	Return result set as a single JSON object
<u>MAX()</u>	Return the maximum value
<u>MIN()</u>	Return the minimum value
<u>STD()</u>	Return the population standard deviation
<u>STDDEV()</u>	Return the population standard deviation
<u>STDDEV_POP()</u>	Return the population standard deviation
<u>STDDEV_SAMP()</u>	Return the sample standard deviation
<u>SUM()</u>	Return the sum
<u>VAR_POP()</u>	Return the population standard variance
<u>VAR_SAMP()</u>	Return the sample variance
<u>VARIANCE()</u>	Return the population standard variance



PISTOLESI-SQL

...CONTINUO WINDOW FUNCTIONS: clausola OVER applicazioni

NON-AGGREGATE FUNCTIONS UTILIZZABILI CON OVER

Name	Description
CUME_DIST()	Cumulative distribution value
DENSE_RANK()	Rank of current row within its partition, without gaps
FIRST_VALUE()	Value of argument from first row of window frame
LAG()	Value of argument from row lagging current row within partition
LAST_VALUE()	Value of argument from last row of window frame
LEAD()	Value of argument from row leading current row within partition
NTH_VALUE()	Value of argument from N-th row of window frame
NTILE()	Bucket number of current row within its partition.
PERCENT_RANK()	Percentage rank value
RANK()	Rank of current row within its partition, with gaps
ROW_NUMBER()	Number of current row within its partition

■ usano l'**intera partition**

■ lavorano **su frame**, cioè sottoinsiemi della partition

- FIRST_VALUE() → Prende il primo valore della partition
- LAST_VALUE() → Prende l'ultimo valore della partition
- RANK() → FA una classifica dei record dentro la partition e restituisce la posizione di un record dentro la classifica
- ROW_NUMBER() → Restituisce il numero della riga dentro la partition

Non-aggregate: funzione ROW_NUMBER()

Assegnare un numero ad ogni medico nella sua specializzazione

```
SELECT M.Matricola, M.Cognome, M.Specializzazione
      ROW_NUMBER OVER(
          PARTITION BY M.Specializzazione
          )
FROM Medico M;
```

“Il numero di ogni riga ad ogni medico che ti arriva, daglielo sulla partizione della sua specializzazione” → Conta i record e azzerà il conto quando cambia la specializzazione

ESEMPIO APPLICATO

Matricola	Cognome	Specializzazione	ROW_NUMBER
014	Indachi	Cardiologia	1
015	Ciani	Cardiologia	2
016	Verdolini	Cardiologia	3
017	Amaranti	Cardiologia	4
018	Terra di Siena	Cardiologia	5
005	Neri	Gastroenterologia	1
013	Blu	Gastroenterologia	2
001	Rossi	Otorinolaringoiatria	1
002	Verdi	Otorinolaringoiatria	2
003	Gialli	Otorinolaringoiatria	3
004	Turchesi	Otorinolaringoiatria	4

PISTOLESI-SQL

Non-aggregate: funzione RANK()

*Effettuare una classifica della convenienza dei medici dipendentemente dalla loro parcella.
Restituire matricola, cognome e posizione nella classifica*

```
SELECT M.Matricola, M.Cognome,
       M.Specializzazione, M.Parcella,
       RANK() OVER(
           ORDER BY M.Parcella
       )
FROM Medico M;
```

ESEMPIO APPLICATO

Matricola	Cognome	Specializzazione	Parcella	RANK()
012	Grigi	Medicina generale	50	1
001	Rossi	Otorinolaringoiatria	100	2
019	Rossi	Oculistica	100	2
013	Blu	Gastroenterologia	110	4
020	Acquamarina	Oculistica	110	4
008	Gialli	Nefrologia	120	6
005	Neri	Gastroenterologia	130	7
002	Verdi	Otorinolaringoiatria	150	8

- Verranno proiettati Matricola, Cognome, Specializzazione e parcella dei Medici secondo un ordine di classifica (a partire dalla posizione 1)
 - La classifica è fatta sulla base della Parcella di ciascun Medico → Dalla Parcella più piccola a quella più grande → Di conseguenza si rispetta il testo dell'esercizio, visto che per convenienza si considera 'dove si spende meno'
- OSS: Salto sulla posizione 3

LA FUNZIONE RANK() STILA UNA CLASSIFICA SULLA BASE DI UN CRITERIO, APPLICATO SU UN ATTRIBUTO.

SI ASSOCIA UNO SCORE IN CLASSIFICA AD OGNI RECORD.

PIÙ ALTO È LO SCORE MIGLIORE È IL RANK.

NEL CASO DI PARI MERITO, IN CLASSIFICA, IL RANK() RIPETE LA STESSA POSIZIONE PER ENTRAMBI I PARI MERITO → SALTANDO LE REALI POSIZIONE PER TANTI POSTI, QUANTI PARI MERITO CI SONO → QUINDI PER I PARI MERITO È COME SE NON ESISTESSE LA REALE POSIZIONE

Non-aggregate: funzione RANK() su PARTITION + ordinamento

Effettuare una classifica dei medici di ogni specializzazione dipendentemente dalla loro parcella, partendo dalla più alta. Restituire matricola, cognome, specializzazione e posizione nella classifica

```
SELECT M.Matricola, M.Cognome, M.Specializzazione
      RANK() OVER(
          PARTITION BY M.Specializzazione
          ORDER BY M.Parcella DESC
      )
FROM Medico M;
```

ESEMPIO APPLICATO

Matricola	Cognome	Specializzazione	RANK()
017	Amaranti	Cardiologia	1
018	Terra di Siena	Cardiologia	1
015	Ciani	Cardiologia	3
016	Verdolini	Cardiologia	4
014	Indachi	Cardiologia	5
005	Neri	Gastroenterologia	1
013	Blu	Gastroenterologia	2
012	Grigi	Medicina generale	1
009	Arancioni	Nefrologia	1
008	Gialli	Nefrologia	2

- A ciascun medico vado ad associare la sua posizione nella classifica, della sua specializzazione (PARTITION BY M.Specializzazione) → Ordinando in base alla loro parcella in maniera decrescente (ORDER BY M.Parcella DESC)
- OSS: Attenzione ai pari merito

PISTOLESI-SQL

Non-aggregate: funzione DENSE_RANK() su PARTITION + ordinamento

Effettuare una classifica senza gap dei medici di ogni specializzazione dipendentemente dalla loro parcella, partendo dalla più alta. Restituire matricola, cognome, specializzazione e posizione nella classifica

```
SELECT M.Matricola, M.Cognome, M.Specializzazione  
      DENSE_RANK() OVER(  
          PARTITION BY M.Specializzazione  
          ORDER BY M.Parcella DESC  
      )  
FROM Medico M;
```

Stessa funzione della RANK(), solamente che in caso di pari merito non esegue il salto di numero → Mantiene gli ex-aequo con la stessa posizione in classifica, e i successivi seguono la numerazione ‘senza salti’ → Detta “rank senza gap”

ESEMPIO APPLICATO

Matricola	Cognome	Specializzazione	DENSE_RANK()
017	Amaranti	Cardiologia	1
018	Terra di Siena	Cardiologia	1
015	Ciani	Cardiologia	2
016	Verdolini	Cardiologia	3
014	Indachi	Cardiologia	4
005	Neri	Gastroenterologia	1
013	Blu	Gastroenterologia	2
012	Grigi	Medicina generale	1
009	Arancioni	Nefrologia	1
008	Gialli	Nefrologia	2

Non-aggregate: RANK multipli con ALIAS

Effettuare una classifica dei medici in base al numero di visite effettuate. Restituire cognome, specializzazione, numero di visite effettuate, posizione nella classifica generale, e la posizione nella classifica per specializzazione

```
WITH visite AS //1  
( //2  
    SELECT M.Cognome, M.Specializzazione, //3  
          V.Medico, //4  
          COUNT(*) AS Visite //5  
    FROM Visita V //6  
    INNER JOIN //7  
        Medico M ON V.Medico=M.Matricola //8  
    GROUP BY V.Medico, M.Cognome, M.Specializzazione //9  
) //10  
SELECT VV.Cognome, VV.Specializzazione, //11  
       VV.Visite, //12  
       RANK() OVER( //13  
           ORDER BY VV.Visite DESC //14  
       ) AS GlobalRank, //15  
       RANK() OVER( //16  
           PARTITION BY VV.Specializzazione //17  
           ORDER BY VV.Visite DESC //18  
       ) AS SpecRank //19  
FROM visite VV; //20
```



PISTOLESI-SQL

...CONTINUO Non-aggregate: RANK multipli con ALIAS

- Da RIGA 1 a RIGA 10 → Faccio una CTE dove, per ogni medico di una specializzazione con la propria matricola associata (RIGA 9), che ha fatto visite → JOIN da RIGA 6 a RIGA 8 → Ne prendo il cognome e la specializzazione (RIGA 3), poi la matricola (RIGA 4) e anche il conteggio di visite che ha fatto (RIGA 5)
- Da RIGA 11 a RIGA 20 → Faccio una query finale considerando la CTE (RIGA 20)
 - Dalla quale ne prendo cognome e specializzazione (RIGA 11), poi il numero delle visite fatte (RIGA 12) e faccio:
 - Da RIGA 13 a RIGA 15 → Una classifica generale per ogni medico in quanto a numero di visite, considerando tutto l'ospedale
 - Da RIGA 16 a RIGA 19 → Una classifica specifica per ogni medico in quanto a numero di visite, ristretta però alla propria specializzazione, tramite la PARTITION a RIGA 17

OSS: Alias per distinguere le due classifiche a RIGA 15 e a RIGA 19

ESEMPIO APPLICATO

Cognome	Specializzazione	Visite	GlobalRank
Indachi	Cardiologia	19	13
Terra di Siena	Cardiologia	19	13
Ciani	Cardiologia	17	15
Verdolini	Cardiologia	17	15
Amaranti	Cardiologia	11	18
Neri	Gastroenterologia	36	3
Blu	Gastroenterologia	17	15
Grigi	Medicina generale	25	11
Arancioni	Nefrologia	34	7
Gialli	Nefrologia	25	11
Celesti	Neurologia	36	3
Rossi	Oculistica	4	19
Acquamarina	Oculistica	3	20
Bianchi	Ortopedia	40	1
Rosi	Ortopedia	40	1
Verdi	Otorinolaringoiatria	35	6
Gialli	Otorinolaringoiatria	33	8
Rossi	Otorinolaringoiatria	28	9
Turchesi	Otorinolaringoiatria	27	10
Marroni	Psichiatria	36	3

Non-aggregate: LAG(AttributoDaConsiderarePerSpostamento, ValoreSpostamento)

Considerare le visite otorinolaringoiatriche dal 2010 al 2019, restituire per ciascuna, matricola del medico, codice fiscale paziente, data, e data della visita precedente del paziente con un medico della stessa specializzazione

```

SELECT V.Medico, V.Paziente, V.'Data',
       LAG(V.'Data', 1) OVER(
           PARTITION BY V.Paziente
           ORDER BY V.'Data'
       )
FROM Visita V
    INNER JOIN
        Medico M ON V.Medico = M.Matricola
WHERE M.Specializzazione = 'Otorinolaringoiatria'
    AND
        YEAR(V.'Data') BETWEEN 2010 AND 2019;
  
```



PISTOLESI-SQL

...CONTINUO Non-aggregate:
LAG(AttributoDaConsiderarePerSpostamento,
ValoreSpostamento)

- Da RIGA 6 a RIGA 8 → Considero le visite fatte dai medici, specializzati in otorinolaringoiatria (RIGA 9), che hanno visitato tra il 2010 e il 2019 (RIGA 11)
- Del risultato precedente ne prendo il medico, il paziente e la data della visita (RIGA 1) ai quali affianco → La data della visita precedente di ciascun paziente (da RIGA 5 a RIGA 9), rispetto a quella data della visita (considerando sempre che sia fatta con i medici otorini, che hanno visitato dal 2010 al 2019)
- RIGA 2 → LAG:
 - Il parametro V.'Data', rappresenta l'attributo rispetto al quale devo considerarci sopra uno spostamento → Andando all'indietro, rispetto alle row della PARTITION fatta a RIGA 3, la quale PARTITION è ordinata dalla data più remota a quella più recente (RIGA 4)
 - Il fatto che mi sposti di una visita precedente lo devo al valore 1 presente tra i parametri → Quindi ogni volta che arriva una visita, per essa si va a vedere la data di una riga precedente nella partizione di visite di quel paziente, ordinata dalla più vecchia a quella più nuova

OSS:

- Gli apici su Data per riferirmi all'attributo del database, vanno messi perché DATA è una parola riservata

ESEMPIO APPLICATO

Medico	Paziente	Data	LAG(V.'Data', 1) OVER (
003	aaa1	2012-05-23	[Null]
001	aaa1	2012-12-01	2012-05-23
001	aaa1	2013-03-01	2012-12-01
004	bbc4	2010-10-03	[Null]
004	bbe1	2010-05-30	[Null]
003	bbe1	2011-11-27	2010-05-30
001	ccc2	2012-07-27	[Null]
002	ddd6	2010-10-16	[Null]
004	eyy9	2010-10-16	[Null]
003	eyy9	2011-01-23	2010-10-16
002	eyy9	2012-02-16	2011-01-23
004	erv4	2012-06-07	[Null]
004	ffq8	2012-09-03	[Null]
002	ffq8	2013-01-26	2012-09-03
001	ffq8	2013-02-01	2013-01-26
002	fqa8	2010-11-16	[Null]
001	gwp5	2010-01-19	[Null]
004	hho1	2011-01-29	[Null]
003	hho1	2011-08-31	2011-01-29
002	hho1	2012-02-15	2011-08-31



Dove c'è NULL come data di visita precedente
→ Rappresenta che la visita precedente non esiste

QUANDO SI USA LA FUNZIONE LAG SI VA A VEDERE QUALCOSA CHE È PASSATO SOTT'OCCHIO IN PRECEDENZA, CONTROLLANDO DOVE QUEL 'QUALCOSA' SI TROVI DENTRO UN PARTIZIONAMENTO CHE LO RIGUARDI.

PISTOLESI-SQL

Non-aggregate:

**LEAD(AttributoDaConsiderarePerSpostamento,
ValoreSpostamento)**

Considerare le visite otorinolaringoiatriche dal 2010 al 2019, restituire per ciascuna, matricola del medico, codice fiscale paziente, data, e data della visita successiva del paziente con un medico della stessa specializzazione

```
SELECT V.Medico, V.Paziente, V.'Data',
       LEAD(V.'Data', 1) OVER(
           PARTITION BY V.Paziente
           ORDER BY V.'Data'
       )
FROM Visita V
    INNER JOIN
        Medico M ON V.Medico = M.Matricola
WHERE M.Specializzazione = 'Otorinolaringoatria'
    AND
        YEAR(V.'Data') BETWEEN 2010 AND 2019;
```

Per LEAD stesso discorso della funzione LAG, solamente che qua non si va a vedere la riga precedente, ma quella successiva

LEAD FA IL CONTRARIO DELLA FUNZIONE LAG, ANTICIPANDOSI SU CIÒ CHE AVVERRÀ

Non-aggregate: CUME_DIST() + WINDOW

Per ogni medico, restituire la sua matricola, il cognome, la parcella, e la percentuale di medici con parcella minore o uguale

```
SELECT M.Matricola, M.Cognome, M.Parcella,
       CUME_DIST() OVER w
FROM Medico M
WINDOW w AS(
    ORDER BY M.Parcella
);
```

- In poche parole, il testo chiede: "Dato un medico, quanti medici prendono meno o al più quanto lui?"
- Data una riga e considerato il valore dell'attributo, in questo caso M.Parcella, la funzione **CUME_DIST()** → Vede quante righe ci sono nella partizione w che hanno il valore di M.Parcella minore o uguale ad esso → E lo fa per tutte le righe della partizione
- La partizione è rappresentata tramite **WINDOW w**, la quale è ordinata per M.Parcella, in ordine crescente
- Il risultato che rappresenta i valori minori o uguali → Viene espresso in percentuale

PISTOLESI-SQL

...CONTINUO Non-aggregate: CUME_DIST() + WINDOW

ESEMPIO APPLICATO

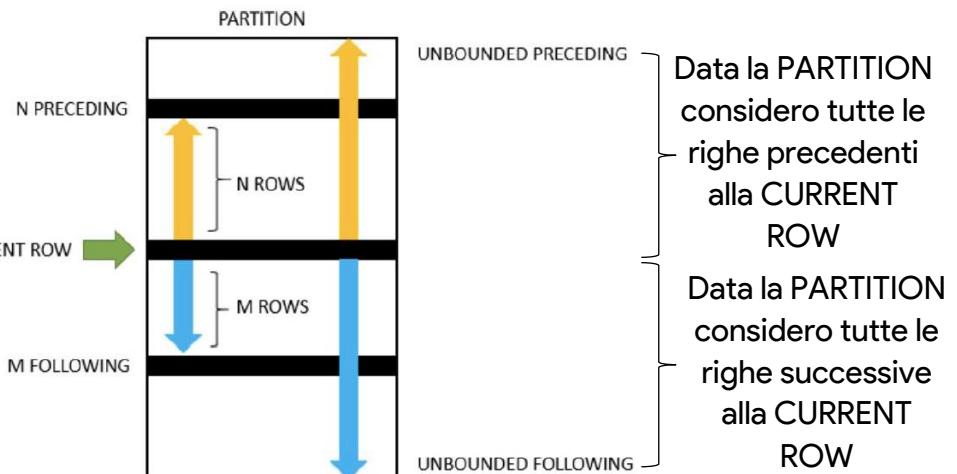
Matricola	Cognome	Parcella	CUME_DIST()
012	Grigi	50	0.05
001	Rossi	100	0.15
019	Rossi	100	0.15
013	Blu	110	0.25
020	Acquamarina	110	0.25
008	Gialli	120	0.3
005	Neri	130	0.35
002	Verdi	150	0.45
011	Marroni	150	0.45
006	Bianchi	160	0.5
009	Arancioni	170	0.55
007	Rosi	180	0.65
014	Indachi	180	0.65
003	Gialli	200	0.75
010	Celesti	200	0.75
016	Verdolini	220	0.8
015	Ciani	230	0.85
004	Turchesi	250	0.9
017	Amaranti	260	1
018	Terra di Siena	260	1

- Ciani ha l'85% dei medici della clinica che hanno la parcella minore o uguale a lui
- Amaranti e Terre di Siena hanno il 100% dei medici della clinica che prendono parcella minore o uguale a loro → Sono le più care

Gestione FRAME

Data la PARTITION
considero un certo
numero N di righe
precedenti alla
CURRENT ROW

Data la PARTITION
considero un certo
numero M di righe
successive alla
CURRENT ROW



Data la PARTITION
considero tutte le
righe precedenti
alla CURRENT
ROW

Data la PARTITION
considero tutte le
righe successive
alla CURRENT
ROW

Non-aggregate: FIRST_VALUE(ValoreDaTrovare) su DEFAULT FRAME

Date le visite cardiologiche dei pazienti 'aaa1', 'bbc4' e 'ccc2' nel triennio 2012-2014, restituire, per ciascuna, matricola del medico, codice fiscale del paziente, data, e data della prima visita effettuata dal paziente con quel medico

```

SELECT V.Medico, V.Paziente, V.'Data'                                //1
      FIRST_VALUE(V.'Data') OVER w                                     //2
  FROM Visita V                                                       //3
 WHERE V.Paziente IN( 'aaa1', 'bbc4', 'ccc2')                         //4
   AND YEAR(V.'Data') BETWEEN 2012                                      //5
   AND 2014                                                               //6
WINDOW w AS(                                                          //8
    PARTITION BY V.Medico, V.Paziente                                 //9
    ORDER BY V.'Data'                                                 //10
  );                                                                    //11
  
```



PISTOLESI-SQL

...CONTINUO Non-aggregate: FIRST_VALUE(ValoreDaTrovare) su DEFAULT FRAME

- RIGA 4 → Vuol dire “Quando il paziente sta tra quelli forniti dal testo”
- RIGA 6 e RIGA 7 → Visite dal 2012 al 2014
- RIGA 2 → Controllo una visita in ordine di tempo sulla PARTITION, creata da RIGA 8 a RIGA 11, e ne prendo quella il valore più remoto, cioè il primo valore dettato dal FIRST_VALUE
- La PARTITION è data dalla WINDOW → Che a sua volta è partizionata per medico e paziente e ordinata per data → Ciò rappresenta il FRAME, su cui ci lavorerà la funzione FIRST_VALUE

OSS: Sono nel caso in cui nella partizione non specifico il FRAME → Quindi sono nel **DEFAULT FRAME** → Considero fino alla CURRENT_ROW → Ciò va bene lo stesso, perché data una visita, vederla in ordine di tempo sia che guardi la metà superiore della

CASO 1 DI DEFAULT FRAME → OVER SU PARTIZIONE CON ORDER BY E ASSENZA DI SPECIFICA DI FRAME → SI LAVORA DALL'INZIO FINO ALLA CURRENT ROW.

CASO 2 DI DEFAULT FRAME → OVER SU PARTIZIONE SENZA ORDER BY E ASSENZA DI SPECIFICA DI FRAME → SI LAVORA DALL'INZIO FINO ALLA FINE DELLA PARTITION

ESEMPIO APPLICATO

Medico	Paziente	Data	FIRST_VALUE(V.'Data')
001	aaa1	2013-03-01	2012-12-01
001	aaa1	2012-12-01	2012-12-01
003	aaa1	2012-05-23	2012-05-23
018	aaa1	2013-02-16	2013-02-16
007	bcb4	2012-09-25	2012-09-25
010	bcb4	2012-01-25	2012-01-25
011	bcb4	2012-02-23	2012-02-23
001	ccc2	2012-07-27	2012-07-27
007	ccc2	2012-01-04	2012-01-04
010	ccc2	2012-06-27	2012-06-27
011	ccc2	2012-12-22	2012-12-22

- In questo esempio si può notare l'applicazione di FIRST_VALUE solo nelle prime due righe, visto che il paziente aaa1 è stato visitato più volte dal cardiologo (in questo caso 001) tra il 2012 e il 2014 → Di conseguenza, ai fini delle richieste dell'esercizio, posso considerarne la data della prima visita

PISTOLESI-SQL

Non-aggregate: LAST_VALUE(ValoreDaTrovare) su FRAME

Date le visite cardiologiche dei pazienti ‘aaa1’, ‘bbc4’ e ‘ccc2’ nel triennio 2012-2014, restituire, per ciascuna, matricola del medico, codice fiscale del paziente, data, e data dell’ultima visita effettuata dal paziente con quel medico

```

SELECT V.Medico, V.Paziente, V.'Data'                                //1
      LAST_VALUE(V.'Data') OVER w                                     //2
  FROM Visita V                                                       //3
 WHERE V.Paziente IN( 'aaa1', 'bbc4', 'ccc2')                         //4
   AND
     YEAR(V.'Data') BETWEEN 2012                                         //5
       AND 2014                                                       //6
 WINDOW w AS(
    PARTITION BY V.Medico, V.Paziente                                 //9
    ORDER BY V.'Data'                                                 //10
    ROWS BETWEEN CURRENT ROW                                         //11
      AND UNBOUNDED FOLLOWING                                       //12
  )
)                                                               //13
ORDER BY V.Paziente, V.Medico;                                         //14

```

- Stesso discorso dell'esercizio precedente, solamente che qua non posso fermarmi alla CURRENT ROW, dato che voglio sapere l'ultimo valore → Di conseguenza quello più recente, che sta in fondo alla partizione → Allora, devo specificare il FRAME

- RIGA 11 e RIGA 12 → Estendo il FRAME dalla CURRENT ROW fino alla fine della PARTITION → Perché mi serve prendere l'ultimo valore e, fermandomi alla CURRENT ROW, il LAST_VALUE non lo potrei ottenere

OSS: Anche se dentro la PARTITION l'ordinamento viene fatto per data, nell'output non viene così, perché fuori viene nuovamente riordinato per paziente e medico (RIGA 14)

ESEMPIO APPLICATO

partition (

Medico	Paziente	Data	LAST_VALUE(V.'Data')
001	aaa1	2013-03-01	2013-03-01
001	aaa1	2012-12-01	2013-03-01
001	aaa1	2012-05-23	2012-05-23
001	aaa1	2013-02-16	2013-02-16
007	bbc4	2012-09-25	2012-09-25
010	bbc4	2012-01-25	2012-01-25
011	bbc4	2012-02-23	2012-02-23
001	ccc2	2012-07-27	2012-07-27
007	ccc2	2012-01-04	2012-01-04

) default frame
frame

QUANDO SI LAVORA CON FIRST_VALUE E LAST_VALUE VANNO DEFINITI I FRAME, STANDO ATTENDI ALL'UTILIZZO DEL DEFAULT FRAME

PISTOLESI-SQL

Aggregate: AVG(Valore) + PRECEDING/FOLLOWING su FRAME

Scrivere una funzione analytics che, per ogni terapia conclusa del paziente 'ttw2', proietti il farmaco, la durata e la durata media rispetto alla terapia precedente e successiva con lo stesso farmaco

```

WITH durate AS //1
(
    SELECT T.Farmaco, T.DatalnizioTerapia, //2
           DATEDIFF(T.DataFineTerapia, DatalnizioTerapia) AS Durata //3
    FROM Terapia T //4
    WHERE T.Paziente = 'ttw2' //5
          AND T.DataFineTerapia IS NOT NULL //6
)
SELECT D.Farmaco, D.Durata, D.DatalnizioTerapia, //7
       AVG(D.Durata) OVER w //8
FROM durate D //9
WINDOW w AS( //10
    ORDER BY F.DatalnizioTerapia //11
    ROWS BETWEEN 1 PRECEDING //12
            AND 1 FOLLOWING //13
);
//14
//15
//16

```

- Dato che non ho né la durata delle terapie, né la durata media di tutte le terapie per quel paziente 'ttw2' → Incomincio a calcolarmi la prima delle durate, che sarà a presente a RIGA 4 (in giorni) dentro la CTE da RIGA 1 a RIGA 8
- RIGA 7 → Impongo che la data di fine terapia non sia NULL → Perché il testo dell'esercizio mi chiede le terapie concluse per quel paziente
- RIGA 9 → Proietto i dati necessari richiesti, prendendoli dalla CTE (RIGA 11) e inoltre mi calcolo la media delle durate (RIGA 10) su una partizione (da RIGA 12 a RIGA 16):
 - Guardando la durata media rispetto alla terapia precedente → Cioè alla riga precedente (RIGA 14), e rispetto alla terapia successiva → Cioè alla riga successiva (RIGA 15)
 - RIGA 13 → Ordino in modo crescente su DatalnizioTerapia per avere la terapia ordinata dalla più vecchia alla più recente per farci poi il confronto

OSS: Non faccio nessun PARTITION BY dentro la WINDOW → Considero tutta la tabella perché, di quella terapia media, ne voglio sapere sempre la precedente e la successiva → È la **moving average** (media mobile)

ESEMPIO APPLICATO

Farmaco	Durata	DatalnizioTerapia	AVG(D.Durata) OVER w
Gaviscon	23	1994-12-03	23.0000
Seglor	23	1995-10-19	24.3333
Efferalgan	27	1997-11-03	34.3333
Inderal	53	1998-12-14	35.3333
Protargolo	26	1999-02-23	44.0000
Quait	53	2002-02-21	28.0000
EN	5	2004-02-10	28.0000

PISTOLESI-SQL

Aggregate: SUM(Valore) e COUNT(*) su FRAME

Considerare le visite ortopediche di ogni paziente, scrivere una query analytics che restituisca codice fiscale del paziente, matricola del medico, la sua parcella, il numero di visite ortopediche effettuate fino a quel momento, e la spesa sostenuta dal paziente per tali visite

```

SELECT V.Paziente, V.Medico,
       M.Parcella,
       COUNT(*) OVER w,-----> 1
       SUM(M.Parcella) OVER w-----> 2
FROM Visita V
  INNER JOIN
    Medico M ON V.Medico = M.Matricola
WHERE M.Specializzazione = 'Ortopedia'
WINDOW w AS(
    PARTITION BY V.Paziente
    ORDER BY V.'Data'
    ROWS BETWEEN UNBOUNDED PRECEDING
            AND CURRENT ROW
)
  
```

- 1) Il conteggio rappresenta il numero di visite ortopediche effettuato fino a quel momento (che sarà dentro la WINDOW)
- 2) La somma rappresenta la spesa sostenuta dal paziente, rispetto al conteggio del punto precedente
- 3) La WINDOW è definita su Paziente ed è ordinata per Data → Considera le righe dall'inizio, fino a quella corrente, andando all'indietro illimitatamente, finché ce ne sono

ESEMPIO APPLICATO

Paziente	Medico	Parcella	COUNT(*) OVER w	SUM(M.Parcella)
aaa1	006	160	0	160
aaa1	007	180	1	340
aaa1	006	160	2	500
aaa1	007	180	3	680
bbc4	006	160	0	160
bbc4	006	160	1	320
bbc4	007	180	2	500
bbc4	007	180	3	680
bbe1	007	180	0	180

row di esempio

Il FRAME si blocca fino alla CURRENT ROW dell'esempio, anche se la PARTITION continua fino a qua (*)
 → Quindi la somma la fa con i record precedenti andando all'indietro

PISTOLESI-SQL

Aggregate: SUM(Valore) e COUNT(*) su FRAME temporale con dichiarazione di RANGE

Considerare le visite ortopediche di ogni paziente, scrivere una query analytics che restituisca codice fiscale del paziente, matricola del medico, la sua parcella, il numero di visite ortopediche effettuate nei sei mesi precedenti, e la spesa sostenuta dal paziente per tali visite

```

SELECT V.Paziente, V.Medico,
       M.Parcella,
       COUNT(*) OVER w - 1, -----> 1
       SUM(M.Parcella) OVER w
FROM Visita V
    INNER JOIN
        Medico M ON V.Medico = M.Matricola
WHERE M.Specializzazione = 'Ortopedia'
WINDOW w AS(
    PARTITION BY V.Paziente
    ORDER BY V.'Data'
    RANGE BETWEEN INTERVAL 6 MONTH PRECEDING
              AND CURRENT ROW
)

```

- Esercizio simile al precedente solamente che qua voglio i 6 mesi precedenti e lo devo fare tramite la partizione al punto 2) → Dove si dichiara un RANGE con un intervallo di tempo dai 6 mesi precedenti alla CURRENT ROW in cui ci troviamo
- 1) Ci metto -1 perché se vado fino alla CURRENT ROW la visita stessa devo toglierla → Perché la somma sostenuta è riferita a quelle dei 6 mesi precedenti rispetto a quella a cui mi riferisco (CURRENT ROW)

ESEMPIO APPLICATO

Paziente	Medico	Parcella	COUNT(*) OVER w	SUM(M.Parcella)
aaa1	006	160	0	160
aaa1	007	180	0	180
aaa1	006	160	0	160
aaa1	007	180	1	340
bbc4	006	160	0	160
bbc4	006	160	1	320
bbc4	007	180	0	180
bbc4	007	180	0	180

LA DICHIARAZIONE DI ROWS CI SERVE PER FARE PASSI IN AVANTI O INDIETRO IN TERMINI DI RIGHE.

LA DICHIARAZIONE DI RANGE CI SERVE PER LIMITARCI TEMPORALMENTE

PISTOLESI-SQL

PIVOTING STATICO

Dati i pazienti nati dopo il 1970, costruire una tabella pivot che per ogni città ne mostri il numero per ciascun sesso, e il totale

```
SELECT P.Città,
       SUM(
           IF(P.Sesso='M', 1, 0)
       ) AS M,
       SUM(
           IF(P.Sesso='F', 1, 0)
       ) AS F,
       COUNT(*) AS Tot
  FROM Paziente P
 WHERE YEAR(P.DataNascita) > 1970
 GROUP BY P.Città;
```

- Le condizioni 1) e 2) prendono una riga alla volta e sommano 1 se trovano il sesso M/F, 0 altrimenti → Gli attributi M e F che prima potevano essere dentro le celle → Tramite il Pivoting sono nell'intestazione
- Si intende come **PIVOTING STATICO** → Perché i valori dell'attributo Sesso sono pochi e noti → Una persona ha genere maschile o femminile

ESEMPIO APPLICATO

PIVOT TABLE

Città	M	F	TOT
Grosseto	1	0	1
Milano	1	0	1
Pisa	2	3	5
Prato	0	1	1
Roma	2	0	2

UNA TABELLA PIVOT CAMBIA IL ODO IN CUI È RAPPRESENTATA UN'INFORMAZIONE, NON L'INFORMAZIONE STESSA

Funzione GROUP_CONCAT(NomeAttributoDaConcatenare)

Per ciascuna specializzazione, indicarne il nome, la parcella media e tutte le parcelle dei medici che ne fanno parte come stringa concatenata

```
SELECT M.Specializzazione,
       AVG(M.Parcella) AS ParcellaMedia,
       GROUP_CONCAT(M.Parcella) AS ListaParcelle
  FROM Medico M;
```

ESEMPIO APPLICATO

Specializzazione	ParcellaMedia	ListaParcelle
Cardiologia	228.0000	260,180,230,220,250
Gastroenterologia	120.0000	130,110
Medicina generale	50.0000	50
Nefrologia	145.0000	120,170
Neurologia	200.0000	200
Ortopedia	170.0000	180,160
Otorinolaringoiatria	175.0000	250,200,150,100
Psichiatria	150.0000	150

- La funzione **GROUP_CONCAT** contiene un result set concatenato sul valore di ParcellaMedia

QUESTA È UNA FUNZIONE CHE RESTITUISCE UNA STRINGA CONCATENATA CONTENENTE I VALORI DIVERSI DA NULL ASSUNTI DA UN ATTRIBUTO NEI VARI RECORD DEL RESULT SET.

LA CONCATENAZIONE AVVIENE TRAMITE UN SEPARATORE CHE È LA VIRGOLA. È COME SE CI SI PORTASSE DA UNA REALTÀ VERTICALE AD UNA ORIZZONTALE.

PISTOLESI-SQL

GROUP_CONCAT(NomeAttributoDaConcatenare) in SELECT

Concatenare le matricole dei cardiologi all'interno di un unico attributo

```
SELECT GROUP_CONCAT(Matricola)
FROM Medico
Where Specializzazione = 'Cardiologia'
```



ESEMPIO APPLICATO

GROUP_CONCAT(Matricola)
014,015,016,017,018

PIVOTING DINAMICO

Considerati i cardiologi, generare una tabella pivot che riepiloghi, per ogni paziente (riga), il numero di visite effettuate con ogni cardiologo (colonna)

```
WITH cardiologi AS //1
(
    SELECT Matricola AS Medico //2
    FROM Medico //3
    WHERE Specializzazione = 'Cardiologia' //4
)
SELECT GROUP_CONCAT( //5
    CONCAT( //6
        'COUNT(IF(Medico = ''', C.Medico, '''', 1, NULL))', //7
        ')AS ''', C.Medico, '''', //8
        ')
    ) //9
    )
FROM Cardiologi C //10
INTO @pivot_query; //11
```

- Da RIGA 1 a RIGA 6 → CTE in cui considero i medici cardiologi
- Da RIGA 7 a RIGA 12 → La funzione SELECT prende ogni cardiologo che viene dal FROM (RIGA 13) un record alla volta e, per ognuno di essi ne dovrebbe creare una colonna → Ma non sapendo quanti sono i cardiologi per fare il numero di colonne → Questa cosa la devo fare dinamicamente passandolo alla GROUP_CONCAT che a sua volta lo passa alla CONCAT
- Da RIGA 8 a RIGA 12 → La CONCAT mi genera il pezzo di SQL dinamicamente e lo inserisce nella variabile pivot_query (RIGA 14) → Qua è presente una parte evidenziata che è statica perché è una stringa e una parte in grassetto che è dinamica, in cui si controlla se:
 - Un determinato medico è uguale al medico pescato dalla CTE → Allora si aggiunge 1 al conteggio, altrimenti NULL (RIGA 9)
 - Poi si rinomina l'attributo che rappresenta il conteggio, proprio con la matricola di quel determinato medico pescato dalla CTE (RIGA 10) → In modo da formare tante colonne, quanti sono i medici cardiologi, con il conteggio attribuito a ciascun medico, che si combinerà poi con i pazienti al passo successivo dopo l'inserimento in pivot_query → Perché voglio generare la tabella pivot che riepiloghi per ogni paziente il numero di visite effettuate con ogni medico

PISTOLESI-SQL

...CONTINUO PIVOTING DINAMICO

QUERY DI PIVOTING

```
SET @pivot_query=CONCAT(
```

```
    'SELECT Paziente, ',  
    '@pivot_query,  
    ' FROM Visita GROUP BY Paziente;'
```

```
);
```

3
parametri
della
CONCAT

- Supponendo che i medici siano 014, 015, 016, 017, 018
- Il contenuto di pivot_query è la parte dinamica ed è il seguente blocco:

```
COUNT(IF(Medico = '014', 1, NULL)) AS '014', COUNT(IF(Medico = '015', 1, NULL)) AS '015',  
COUNT(IF(Medico = '016', 1, NULL)) AS '016', COUNT(IF(Medico = '017', 1, NULL)) AS '017',  
COUNT(IF(Medico = '018', 1, NULL)) AS '018'
```

- Il resto (evidenziato) è la parte statica
- Il risultato di questa SET è il seguente

```
SELECT Paziente, COUNT(IF(Medico = '014', 1, NULL)) AS '014', COUNT(IF(Medico = '015', 1, NULL)) AS  
'015', COUNT(IF(Medico = '016', 1, NULL)) AS '016', COUNT(IF(Medico = '017', 1, NULL)) AS '017',  
COUNT(IF(Medico = '018', 1, NULL)) AS '018' FROM Visita GROUP BY Paziente;
```

→ è una query che contiene il raggruppamento sul paziente, poi ha come primo attributo restituito la lista dei pazienti, e come altre colonne un conteggio condizionale che conterrà 1, se ho una visita di un paziente con uno dei medici (da 014 a 018), 0 altrimenti → Alla fine di tutte le visite tutti i COUNT colllasseranno e conteranno tutte le visite

PREPARAZIONE

```
PREPARE sql_statement
```

```
FROM @pivot_query;
```

ESECUZIONE

```
EXECUTE sql_statement;
```

ESEMPIO APPLICATO

PAZIENTI	MEDICI				
	014	015	016	017	018
aaa1	1	1	1	1	1
bbc4	1	1	0	1	1
bbe1	1	2	0	1	1
ccc2	1	1	1	0	1
ddd6	1	1	1	0	1
eey9	1	1	1	0	1
erv4	1	1	1	1	0
ffq8	1	1	1	1	1
fqa8	1	1	1	1	1
gox9	2	0	1	0	1
gwp5	1	1	2	1	1
hho1	1	1	1	0	1
kkw1	1	1	1	0	1
kap6	1	1	1	1	1

```

1  /* Indicare matricola e da quanti giorni risultavano iscritti gli studenti, che
2     non si erano ancora laureati il 15 luglio 2005 */
3
4  SELECT Matricola,
5      DATEDIFF('2005-07-15', DataIscrizione) AS GiorniIscrizioneStudenti
6  FROM Studente
7  WHERE DataIscrizione<'2005-07-15'
8      AND
9          DataLaurea>'2005-07-15'
10         OR DataLaurea IS NULL
11     );
12
13 /*
14 - "...che non si erano ancora laureati" --> Sta ad indicare gli studenti che
15 a quella data stavano ancora studiando, e cioè:
16   • RIGA 7 è una condizione obbligatoria che indica che, gli studenti che hanno
17     la data di iscrizione prima del 15 luglio 2005 e contemporaneamente
18     1) RIGA 9 gli studenti che hanno la data di laurea dopo il 15 luglio 2005
19     oppure
20     2) RIGA 10 gli studenti che non hanno la data di laurea
21 - "...da quanti giorni risultavano iscritti" --> Rispetto al 15 luglio 2005
22 quindi, per indicare il numero dei giorni si usa a RIGA 5 il DATEFDIFF tra
23 il 15 Luglio 2005 che è la data più recente e DataIscrizione che è quella
24 più remota (con opportuna ridefinizione)
25 */

```

```

1  /* Indicare matricola e cognome degli studenti il cui percorso di studi è
2   durato (o dura da) oltre sei anni.
3   Risolvere con e senza l'uso di INTERVAL
4   -- SENZA INTERVAL -- */
5
6  SELECT Matricola,Cognome
7  FROM Studente
8  WHERE (
9      DataLaurea IS NOT NULL
10     AND PERIOD_DIFF(
11         DATE_FORMAT(DataLaurea, '%Y%m') ,
12         DATE_FORMAT(DataIscrizione, '%Y%m')
13         )>72
14     )
15     OR
16     (
17         DataLaurea IS NULL
18         AND PERIOD_DIFF(
19             DATE_FORMAT(CURRENT_DATE, '%Y%m') ,
20             DATE_FORMAT(DataIscrizione, '%Y%m')
21             )>72
22     );
23
24 /*
25 - "...è durato (o dura da) oltre sei anni" --> è come se avessi due
26 condizioni in una e posso riscriverla come :
27 1) Da RIGA 8 a RIGA 14 é durato oltre sei anni --> Ci indica che
28    si è laureato e il percorso di studi è finito, quindi la durata
29    la devo calcolare dalla data di iscrizione alla data di laurea e
30    controllo se la differenza in mesi, tra queste due date è maggiore di
31    72 che sono 6 anni da RIGA 10 a RIGA 13
32    (VA SPECIFICATO che la persona si sia laureata DataLaurea IS NOT NULL
33    che è a RIGA 9)
34    • Oppure RIGA 15
35  2) Da RIGA 16 a RIGA 22 dura da oltre sei anni --> Ci indica che sta
36    ancora studiando, e qui la durata la devo calcolare dalla data di
37    iscrizione alla data di oggi e controllo se la differenza in mesi è
38    maggiore di 72 da quando si sono iscritti ad oggi da RIGA 18 a RIGA 21
39    (VA SPECIFICATO che la persona non si sia laureata DataLaurea IS NULL
40    che è a RIGA 17)
41 OSS: In entrambe le PERIOD_DIFF estrapolo le date secondo questo formato
42   '%Y%m' per fare poi, dopo aver fatto la sottrazione, il confronto con
43   le mensilità che sono 72
44 */

```

```

1  /* Indicare matricola e cognome degli studenti il cui percorso di studi è durato
2   (o dura da) oltre sei anni.
3   Risolvere con e senza l'uso di INTERVAL
4   -- CON INTERVAL -- */
5
6  SELECT Matricola, Cognome
7  FROM Studente
8  WHERE (
9      DataLaurea IS NOT NULL
10     AND DataLaurea > DataIscrizione + INTERVAL 6 YEAR
11 )
12 OR
13 (
14     DataLaurea IS NULL
15     AND CURRENT_DATE > DataIscrizione + INTERVAL 6 YEAR
16 );
17
18 /*
19 - "...è durato (o dura da) oltre sei anni" --> è come se avessi due
20 condizioni in una e posso riscriverla come :
21 1) Da RIGA 8 a RIGA 11 è durato oltre sei anni --> Ci indica che
22   si è laureato e il percorso di studi è finito, quindi la durata
23   la devo calcolare dalla data di iscrizione alla data di laurea e
24   controllo se, a partire dalla data di iscrizione + un intervallo di
25   6 anni, supero la data di laurea, quindi mi ci sono voluti più di 6
26   anni per laurearmi RIGA 10
27   (VA SPECIFICATO che la persona si sia laureata DataLaurea IS NOT NULL
28   che è a RIGA 9)
29   ° Oppure RIGA 12
30 2) Da RIGA 13 a RIGA 16 dura da oltre sei anni --> Ci indica che sta
31   ancora studiando, e qui la durata la devo calcolare dalla data di
32   iscrizione alla data di oggi e controllo se, a partire dalla data di
33   iscrizione + un intervallo di 6 anni, supero la data attuale, quindi
34   ancora non mi sono laureato e sono passati + di 6 anni RIGA 15
35   (VA SPECIFICATO che la persona non si sia laureata DataLaurea IS NULL
36   che è a RIGA 14)
37 */

```

```

1  /* Indicare nome, cognome ed età degli studenti laureati quest'anno in Lettere
2   (durata standard 5 anni a ciclo unico), non fuori corso e come minimo con un
3   anticipo di sei mesi rispetto alla durata standard */
4
5  SELECT Nome, Cognome,
6    (YEAR(CURRENT_DATE) - YEAR(DataNascita)) AS Età
7  FROM Studente
8  WHERE Facolta = 'Economia'
9    AND YEAR(DataLaurea) = YEAR(CURRENT_DATE)
10   AND DataLaurea BETWEEN (DataIscrizione + INTERVAL 4 YEAR + INTERVAL 6 MONTH)
11                           AND (DataIscrizione + INTERVAL 5 YEAR);
12 /*
13 - "...laureati quest'anno" RIGA 9
14 - "...in lettere" RIGA 8
15 - "...non fuori corso e come minimo con un anticipo di sei mesi" RIGA 10, RIGA 11:
16   ° Considero che la data di laurea sia compresa tra 4 anni e 6 mesi dopo la
17     data iscrizione RIGA 10 (per l'anticipo dei sei mesi dal ciclo unico di 5 anni),
18     e 5 anni dopo la data iscrizione RIGA 11 (per il ciclo unico di 5 anni)
19 - RIGA 6 calcolo dell'età al millesimo che non è l'età esatta e, facendo così
20   si può calcolare, però potrei ancora non averli compiuti gli anni
21 */

```

```

1  /* Indicare matricola e durata in mesi del percorso di studi degli studenti
2   laureati fuori corso, cioè oltre il mese di Aprile del 6° anno, nell'anno
3   accademico 2009-2010 */
4
5  SELECT Matricola,
6      PERIOD_DIFF(
7          DATE_FORMAT(DataLaurea, '%Y%m'),
8          DATE_FORMAT(DataIscrizione, '%Y%m')
9      ) AS PeriodoDiStudiTotaleInMesi
10 FROM Studente
11 WHERE DataLaurea IS NOT NULL
12     AND DataLaurea > DataIscrizione + INTERVAL 6 YEAR + INTERVAL 4 MONTH
13     AND DataLaurea BETWEEN '2009-09-01'
14                 AND '2010-08-31';
15
16 /*
17 - Da RIGA 6 a RIGA 9 calcolo la durata in mesi del percorso di studio
18 - RIGA 12 considero che la data laurea sia durata dalla data di iscrizione più
19   (usando il >) di 6 anni e 4 mesi
20 - RIGA 13 e RIGA 14 rappresentano l'anno accademico 2009-2010
21 */

```

```

1  /* Indicare codice fiscale, nome e cognome ed età del paziente più anziano
2   della clinica, e il numero di medici dai quali è stato visitato. */
3
4  WITH paziente_piu_anziano AS
5  (
6      SELECT P.CodFiscale, P.Nome, P.Cognome,
7          YEAR(CURRENT_DATE)-YEAR(P.DataNascita) AS Eta
8      FROM Paziente P
9      WHERE NOT EXISTS(
10          SELECT *
11              FROM Paziente P2
12              WHERE P2.DataNascita < P.DataNascita
13      )
14  )
15  SELECT PPA.CodFiscale, PPA.Nome, PPA.Cognome, PPA.Eta,
16  (
17      SELECT COUNT(DISTINCT V.Medico)
18      FROM Visita V
19      WHERE V.Paziente = PPA.CodFiscale
20  ) AS QuantiMedici
21  FROM paziente_piu_anziano PPA;
22
23 /*
24 - Da RIGA 4 a RIGA 14 ho la CTE che rappresenta il paziente più anziano
25 della clinica e funziona così:
26     ° Considero il codice fiscale, il nome e il cognome di un paziente RIGA 6,
27     ° Ne calcolo anche l'età RIGA 7
28     ° Verifichio che questo paziente non sia tra:
29         I pazienti più vecchi di lui --> CORRELATED SUBQUERY di tipo EXISTS
30         (che va dalla RIGA 9 alla RIGA 13) --> Lo faccio per tutti i pazienti
31         della query esterna (sempre all'interno della CTE), fino a quando non
32         trovo quello più anziano, che sarà poi il risultato della CTE, tramite
33         il suo codice fiscale, il suo nome, il suo cognome e la sua età
34 - Da RIGA 15 a RIGA 21 ho la Query Finale che mi rappresenta la soluzione
35     del testo per il calcolo dei medici dai quali, il paziente più anziano
36     della clinica è stato visitato --> Lo faccio tramite una CORRELATED
37     SUBQUERY che va da RIGA 16 a RIGA 20 e, funziona così:
38     ° Calcolo il numero delle visite dei medici (togliendo i duplicati) RIGA 17,
39     ° del paziente più anziano della clinica RIGA 19 --> Qua ho la correlazione
40     con la query esterna!
41 - OSS: La query completa funziona se avessimo più pazienti anziani, considerando
42     i pari merito (ex-aequo)
43 */

```

```

1  /* Indicare l'età media dei pazienti mai vistati da ortopedici */
2
3  SELECT AVG(YEAR(CURRENT_DATE) - YEAR(P.DataNascita)) AS EtàMediaPazienti
4  FROM(
5      SELECT V.paziente
6      FROM visita V
7          INNER JOIN
8              medico M ON V.medico = M.Matricola
9          WHERE M.Specializzazione = 'Ortopedia'
10     )AS D
11     RIGHT OUTER JOIN
12         paziente P ON D.paziente = P.CodFiscale
13     WHERE D.paziente IS NULL;
14
15 /*
16 - RIGA 1 calcolo l'età media dei pazienti
17 mai visitati da ortopedici da RIGA 4 a RIGA 13
18 - Per il "...mai visitati da ortopedici" faccio una DERIVED TABLE da
19    RIGA 4 a RIGA 10 dove si trovano le visite dei medici ortopedici e la metto
20    in JOIN ESTERNO DX con il paziente e, laddove il JOIN non avviene
21    --> Pongo NULL a SX, il che vuol dire che le visite non sono di pazienti
22    con i medici ortopedici
23 - Considero i pazienti NULL RIGA 13, cosicchè mi vengono dati
24    quelli che non hanno il medico ortopedico che li ha visitati
25 */

```

```

1  /* Indicare nome e cognome dei pazienti che sono stati visitati non meno
2      di due volte dalla dottoressa Gialli Rita (NEL DB è Gialli Margherita) */
3
4  WITH visite_rita_gialli_per_paziente AS
5  (
6      SELECT V.Paziente,
7          COUNT(*)
8      FROM Visita V
9      WHERE V.Medico =
10         (
11             SELECT M.Matricola
12             FROM Medico M
13             WHERE M.Nome = 'MargheRita'
14                 AND M.Cognome = 'Gialli'
15         )
16     GROUP BY V.Paziente
17     HAVING COUNT(*) >= 2
18 )
19 SELECT P.Nome, P.Cognome
20 FROM Paziente P
21 WHERE P.CodFiscale IN
22   (
23       SELECT VRGPP.Paziente
24       FROM visite_rita_gialli_per_paziente VRGPP
25   );
26 /*
27 - Da RIGA 4 a RIGA 18 costruisco una CTE che avrà i pazienti RIGA 6 con
28 associato :
29 il numero di visite effettuate RIGA 7, le cui visite sono associate
30 al medico che è uguale al RIGA 9 --> Medico Margherita Gialli, risultato
31 della SUBQUERY NON CORRELATA SCALARE da RIGA 10 a RIGA 15
32 - Per la CTE impongo di raggruppare per il paziente visitato RIGA 16 e che il
33 conteggio del numero di visite di margherita gialli sia >=2 RIGA 17, che
34 implica che ci sono almeno 2 visite della dottoressa per i vari pazienti
35 trovati
36 - Da RIGA 19 a RIGA 25 ho la query finale che mi prende il cognome e il nome
37 dei pazienti RIGA 19 e, controlla se il codice fiscale è tra RIGA 21 --> i
38 pazienti pescati dalla CTE, cioè tra quelli visitati almeno due volte dalla
39 dottoressa Rita Gialli, da RIGA 22 a RIGA 25
40 - A parole: "Prendo il paziente laddove il codice fiscale è fra i codici
41 fiscali dei pazienti visitati non meno di due volte dalla Dottoressa
42 Gialli Margherita"
43 - REGOLA 1: "Non Meno di x" vuol dire: >= di x oppure > di (x-1) cioè
44 del precedente
45 - REGOLA 2: Il GROUP BY rappresenta: "Per ogni..., Per ciascun...,
46 Da ciascun..., ecc..."
```

```

1  /* Indicare nome e cognome dei pazienti che sono stati visitati non meno di due
2   volte dalla dottoressa Gialli Rita (NEL DB è Gialli Margherita) */
3
4  SELECT DISTINCT P.Nome, P.Cognome
5  FROM Paziente P
6    INNER JOIN
7      Visita V1 ON P.CodFiscale = V1.Paziente
8    INNER JOIN
9      Medico M ON M.Matricola = V1.Medico
10   INNER JOIN
11     Visita V2 ON (
12       V1.Medico = V2.Medico
13       AND V1.Paziente = V2.Paziente
14       AND V1.Data <> V2.Data
15     )
16   WHERE M.Nome = "Margherita"
17     AND M.Cognome = "Gialli";
18
19  /*
20  - "Non meno di 2 volte" -> Condizione 1 cioè più di una volta
21  - Margherita Gialli -> Condizione 2
22  - Combino tutte e 3 le tabelle Paziente, Visita e Medico da RIGA 5 a RIGA 9
23  perchè mi serviranno tutte e 3
24  - Affianco alle combinazioni precedenti la tabella Visita tramite un SELF JOIN
25  da RIGA 11 a RIGA 15
26  - Proietterò i valori dei Cognomi e dei Nomi dei pazienti RIGA 4, visitati
27  (JOIN con Visita da RIGA 5 a RIGA 7) da un medico (JOIN con Medico da RIGA 7
28  a RIGA 8) laddove, il medico è lo stesso RIGA 12, il paziente pure RIGA 13 ma
29  la data della visita è diversa RIGA 14 --> Questo fatto che la data della
30  vistia sia diversa implica che, c'è stata un'altra visita e quindi rispetto
31  la condizione 1
32  - La condizione 2 la avrò nel WHERE RIGA 16 e RIGA 17
33  - A parole: "Affianco a ciascuna visita fatta dalla dottoressa Margherita Gialli,
34  tutte le visite fatte da Margherita Gialli con lo stesso paziente ma con
35  data diversa"
36 */

```

```

1  /* Indicare il reddito medio dei pazienti che sono stati visitati solo da
2     medici con parcella superiore a 100€. */
3
4  WITH pazienti_solo_sopra_cento_euro AS
5  (
6      SELECT V.Paziente
7      FROM Visita V
8          INNER JOIN
9              Medico M ON V.Medico = M.Matricola
10     WHERE NOT EXISTS
11         (
12             SELECT *
13             FROM Visita V2
14             WHERE V2.Paziente = V.Paziente
15                 AND V2.Medico IN
16                     (
17                         SELECT M2.Matricola
18                         FROM Medico M2
19                             WHERE M2.Parcella <= 100
20                     )
21         )
22     )
23     SELECT AVG(P.Reddito)
24     FROM Paziente P
25     WHERE P.CodFiscale IN
26         (
27             SELECT *
28             FROM pazienti_solo_sopra_cento_euro
29         );
30
31 /*
32 - CTE da RIGA 4 a RIGA 22
33 - Da RIGA 11 a RIGA 21 SUBQUERY CORRELATA perché la visita
34     del paziente V2 dipende dalla visita del paziente V (esterno)
35 - Da RIGA 16 a RIGA 20 SUBQUERY NON CORRELATA
36 - Scelgo i pazienti RIGA 6 visitati dai medici RIGA 9, laddove non esistono
37     RIGA 10: altre visite di quei pazienti da RIGA 12 a RIGA 14 che abbiano il
38     medico tra RIGA 15, quelli con parcella minore di 100 da RIGA 16 a RIGA 20
39     --> Prendo quindi, solo i pazienti che sono stati visitati da medici con
40     parcella >100
41 - Tutto il risultato precedente sarà della CTE
42 - Da RIGA 23 a RIGA 29 query finale
43 - RIGA 23, RIGA 24 calcolo il reddito medio dei pazienti, il cui codice fiscale
44     sta tra RIGA 25: i pazienti che sono stati visitati da medici con parcella >100
45     SUBQUERY NON CORRELATA da RIGA 26 a RIGA 29
46 */

```

```

1  /* Considerata ogni specializzazione indicarne il nome e l'incasso totale
2   degli ultimi due anni */
3
4  WITH visite_ultimi_due_anni AS
5  (
6      SELECT V.Medico
7      FROM Visita V
8      WHERE YEAR(V.Data) BETWEEN YEAR(CURRENT_DATE)-2
9          AND YEAR(CURRENT_DATE)-1
10         AND V.Mutuata IS FALSE
11 )
12 , incassi_medici_ultimi_due_anni AS
13 (
14     SELECT M.Matricola, M.Specializzazione,
15        M.Parcella *
16     (
17         SELECT COUNT(*)
18         FROM visite_ultimi_due_anni VUDA
19         WHERE VUDA.Medico = M.Matricola
20     ) AS IncassoMedico
21     FROM Medico M
22 )
23 SELECT DISTINCT IMUDA.Specializzazione,
24 (
25     SELECT SUM(IMUDA0.IncassoMedico)
26     FROM incassi_medici_ultimi_due_anni IMUDA0
27     WHERE IMUDA0.Specializzazione = IMUDA.Specializzazione
28 ) AS IncassoSpec
29 FROM incassi_medici_ultimi_due_anni IMUDA;
30
31 /*
32 - Da RIGA 4 a RIGA 11, prima CTE rappresenta i medici che hanno visitato gli
33 ultimi 2 anni
34 - Considero i medici che hanno fatto visite RIGA 6 e RIGA 7 :
35     • Tra l'anno corrente-2 RIGA 8 e l'anno corrente-1 RIGA 9 (ultimi due anni)
36     (ES 2022-2 = 2020 e 2022-1=2021 --> Visite tra il 2020 e il 2021 compresi
37     che sono 2 anni, cioè gli ultimi 2, tranne l'anno corrente!)
38     • Con la visita mutuata a FALSE RIGA 10, per non avere medici che sono con
39     il SSN, altrimenti non potremmo calcolare la parcella, per l'incasso totale
40 - Da RIGA 12 a RIGA 22, seconda CTE dove ottengo i medici, la loro specializzazione e
41 l'incasso ottenuto da ciascuno di essi, negli ultimi due anni ottenuto così:
42     1) Medico per medico, ne considero la parcella RIGA 15
43     2) Moltiplico questa parcella (*) per il result set della --> CORRELATED SUBQUERY
44     da RIGA 16 a RIGA 20 per ogni medico proveniente dalla query esterna, conto
45     quante
46     visite ha fatto negli ultimi 2 anni (che è ottenuto andando a vedere nella prima
47     CTE RIGA 26)
48     --> Questa moltiplicazione sulla parcella, mi da esattamente quanto ha incassato
49     ciascun medico negli ultimi due anni, e lo rinomino come IncassoMedico
50 - Da RIGA 23 a RIGA 29 query finale
51 - Per ogni riga della seconda CTE RIGA 29, ne proietto senza duplicati:
52     • La Specializzazione RIGA 23
53     • L'incasso totale RIGA 25 ottenuto dalla --> Somma su tutti gli incassi dei medici
54     che
55     hanno fatto visite negli ultimi 2 anni (da RIGA 16 a RIGA 20, quindi mi riferisco
56     alla seconda CTE RIGA 26), per ogni specializzazione (distinta) RIGA 27
      --> La SUBQUERY è CORRELATA da RIGA 24 a RIGA 28
*/
```

```

1  /* Considerata ogni specializzazione indicarne il nome e l'incasso totale
2   degli ultimi due anni */
3
4  Select M.Specializzazione,
5      SUM(M.Parcella) AS Incasso
6  FROM Visita V
7      INNER JOIN
8          Medico M ON V.Medico = M.Matricola
9  WHERE YEAR(V.Data) BETWEEN YEAR(CURRENT_DATE)-2
10    AND YEAR(CURRENT_DATE)-1
11    AND V.Mutuata IS FALSE
12 GROUP BY M.Specializzazione;
13
14 /*
15 - La parola "ogni" nasconde un raggruppamento
16 - "...ultimi due anni" --> RIGA 9 e RIGA 10
17   ° Visita tra l'anno corrente-2 RIGA 9 e l'anno corrente-1 RIGA 10
18   (quindi rappresenta gli ultimi due anni)
19   (ES 2022-2 = 2020 e 2022-1=2021 --> Visite tra il 2020 e il 2021 compresi
20     che sono 2 anni, cioè gli ultimi 2 anni, tranne il corrente!)
21 - RIGA 11 metto la visita mutuata a FALSE, per non avere medici che sono con
22   il SSN, altrimenti non potremmo calcolarne la parcella, per l'incasso totale
23   (si potrebbe mettere "= 0" al posto di "IS FALSE", può essere una convenzione)
24 - RIGA 12 nel raggruppamento non metto la SUM... perchè è un attributo aggregato
25 */

```

```

1  /* Indicare il reddito medio dei pazienti che non sono stati visitati da otorini
2   nel trentesimo anno d'età */
3
4  WITH pazienti_visitati_otorini AS
5  (
6      SELECT V.Paziente
7      FROM Visita V
8      WHERE V.Paziente IN
9          (
10         SELECT P.CodFiscale
11         FROM Paziente P
12         WHERE YEAR(V.Data) >= (YEAR(DataNascita) + INTERVAL 29 YEAR)
13         AND YEAR(V.Data) <= (YEAR(DataNascita) + INTERVAL 30 YEAR)
14     )
15     AND V.Medico IN
16     (
17         SELECT M.Matricola
18         FROM Medico M
19         WHERE M.Specializzazione = 'Otorinolaringoiatria'
20     )
21 )
22 SELECT AVG(P.Reddito) AS RedditoMedio
23 FROM Paziente P
24 WHERE P.CodFiscale NOT IN
25     (
26         SELECT *
27         FROM pazienti_visitati_otorini
28     );
29
30 /*
31 - Da RIGA 4 a RIGA 21 ho la CTE che mi rappresenta i pazienti visitati da
32   otorini nel trentesimo anno d'età
33   A) Il "trentesimo anno d'età" del paziente rappresenta l'età tra 29 e 30 anni
34   e la ottengo dalla SUBQUERY CORRELATA da RIGA 9 a RIGA 14
35   --> Prendo i codici fiscali dei pazienti e verifico che abbiano:
36       1) RIGA 12 l'anno della data della visita >=
37           l'anno della data di nascita + 29 anni
38       2) RIGA 13 l'anno della data della visita <
39           l'anno della data di nascita + 30 anni
40   ES. Anno data nascita 1990 + 29 anni = 2019, 1990+30 = 2020, quindi l'anno
41   della data della visita deve stare tra il 2019 e il 2020 (estremi inclusi)
42   B) "...da otorini" lo ottengo dalla SUBQUERY NON CORRELATA da RIGA 16 a RIGA 20
43 - Da RIGA 22 a RIGA 28 ho la query finale dove, calcolo il reddito medio del
44   paziente RIGA 22 il cui codice fiscale non è RIGA 24 nel result set della CTE
45   --> SUBQUERY NON CORRELATA da RIGA 25 a RIGA 28, cioè non è tra i pazienti che
46   sono stati visitati da otorini nel trentesimo anno di età
47 */

```

```

1  /* Indicare codice fiscale, nome, cognome, ed età del paziente più anziano della
2   clinica, e il numero di volte in cui è stato visitato da ogni medico */
3
4  SELECT V.Medico, V.Paziente,
5      D2.Nome, D2.Cognome, D2.Eta,
6      COUNT(*) AS NumeroVisite
7  FROM Visita V
8      NATURAL JOIN
9      (
10         SELECT P.CodFiscale AS Paziente,
11             P.Nome, P.Cognome,
12             D.EtaMax AS Eta
13     FROM Paziente P
14     CROSS JOIN
15     (
16         SELECT MAX(YEAR(CURRENT_DATE) - YEAR(P.DataNascita)) AS EtaMax
17         FROM Paziente P
18     ) AS D
19     WHERE (YEAR(CURRENT_DATE) - YEAR(P.DataNascita)) = D.EtaMax
20 ) AS D2
21 GROUP BY V.Medico,
22     V.Paziente,
23     D2.Nome, D2.Cognome, D2.Eta;
24
25 /*
26 - "...da ogni" --> Nasconde un GROUP BY
27 - Da RIGA 15 a RIGA 18 ho una DERIVED TABLE_1 dove calcolo l'età massima tra i
28 pazienti
29 - Da RIGA 9 a RIGA 20 ho una DERIVED TABLE_2 dove, si prendono il codice fiscale
30 con alias RIGA 10, il nome e cognome RIGA 11, l'età massima RIGA 12 (pescata dalla
31 DERIVED TABLE_1) del paziente RIGA 13 e faccio il PRODOTTO CARTESIANO (SELF) con la
32 DERIVED TABLE_1 imponendo che l'età del paziente sia l'età massima RIGA 19
33 (dove l'età massima è il risultato della DERIVED TABLE_1)
34 --> Quindi costruisco una tabella intermedia riguardante il paziente più anziano
35 - OSS Considerando che l'età massima per politiche di DBMS è 100 per ESEMPIO:
36     P1 | 85 | 100
37     P2 | 45 | 100
38     P3 | 100 | 100
39     P4 | 86 | 100
40     ecc... --> P3 è quello che ha l'età massima
41 - Proietto:
42     • Il medico che ha visitato, e il paziente visitato RIGA 4
43     • Il nome, il cognome e l'età massima (ricordo 'Eta' è un alias dell'età massima
44       RIGA 12) del paziente visitato (che sarà frutto del NATURAL JOIN tra Visita e
45       DERIVED TABLE_2, sull'attributo omonimo V.Paziente = Paziente
46       (Alias di P.Paziente RIGA 10)
47     • Faccio il conteggio di queste visite per ogni medico RIGA 6
48       --> Ottenuto dal raggruppamento su V.Medico RIGA 21, RIGA 22 e RIGA 23 e aggiungo
49       gli altri attributi nel predicato di raggruppamento poichè non sono costanti come
50       V.Medico, perchè li devo proiettare
51 */

```

```

1  /* Indicare nome e cognome dei pazienti visitati almeno una volta da tutti
2   i cardiologi di Pisa nel primo trimestre del 2015 */
3
4  WITH Cardiologi AS
5  (
6      SELECT M.Matricola
7      FROM Medico M
8      WHERE M.Specializzazione = 'Cardiologia'
9          AND M.Citta = 'Pisa'
10 )
11 , visite_target AS
12 (
13     SELECT V.Paziente, V.Medico
14     FROM Visita V
15     INNER JOIN
16         Cardiologi C ON V.Medico = C.Matricola
17     WHERE YEAR(V.Data) = 2015
18         AND MONTH(V.Data) BETWEEN 1
19                         AND 3
20 )
21 SELECT VT.Paziente, P.Cognome, P.Nome
22 FROM visite_target VT
23     INNER JOIN
24         Paziente P ON VT.Paziente = P.CodFiscale
25 GROUP BY VT.Paziente, P.Cognome, P.Nome
26 HAVING COUNT(DISTINCT VT.Medico) =
27 (
28     SELECT COUNT(*)
29     FROM Cardiologi
30 );
31
32 /*
33 - OSS: Ricordo che la parola "TUTTI" -> Viene rappresentato con la divisione
34 - Prendo il problema e lo suddivido in 3 categorie: a, b, c
35     a) Trovo i cardiologi di Pisa
36     b) Trovo i pazienti visitati da medici in a) nel primo trimestre 2015
37     c) Controllo che i pazienti in b) siano stati visitati da tutti di a)
38 - La 1° CTE da RIGA 4 a RIGA 10 mi corrisponde ad a)
39 - La 2° CTE da RIGA 11 a RIGA 20 mi Corrisponde a b)
40 - RIGA 13 Faccio la selezione anche sul medico perchè per ogni paziente
41     visitato dai medici provenienti da a), questi medici me li devo portare
42     dietro poichè mi servirà per certezza sapere che fanno parte di a)
43 - Da RIGA 14 a RIGA 16 faccio il JOIN con la 1° CTE
44 - Per fare il punto c) procedo così nella QUERY FINALE da RIGA 21 a RIGA 30:
45     Paziente per paziente da RIGA 21 a RIGA 25, controllo se il numero di cardiologi
46     di Pisa diversi dai quali ogni paziente è stato visitato il primo trimestre 2015
47     RIGA 26 è uguale ai cardiologi di Pisa, da RIGA 27 a RIGA 30.
48     Se ciò vale il paziente è stato visitato da tutti i cardiologi di Pisa
49 - Nel punto c) per la divisione si utilizza la SUBQUERY DI CONTEGGIO NELLA
50     HAVING CLAUSE da RIGA 26 a RIGA 30
51 */

```

```

1  /* Indicare nome e cognome dei pazienti visitati almeno una volta da tutti
2   i cardiologi di Pisa nel primo trimestre del 2015 */
3
4  SELECT P.Cognome, P.Nome
5  FROM Paziente P
6  WHERE NOT EXISTS
7  (
8      SELECT *
9      FROM Medico M
10     WHERE M.Specializzazione = 'Cardiologia'
11        AND M.Citta = 'Pisa'
12        AND NOT EXISTS
13        (
14            SELECT *
15            FROM Visita V
16            WHERE YEAR(V.Data) = 2015
17              AND MONTH(V.Data) BETWEEN 1
18                                AND 3
19              AND V.Paziente = P.CodFiscale
20              AND V.Medico = M.Matricola
21        )
22    );
23
24 /*
25 - Per risolvere il testo considero "DA TUTTI..." che mi rappresenta il
26 costrutto della divisione utilizzo la funzione del doppio NOT EXISTS
27 e quindi posso interpretare il testo così:
28 Trovo i pazienti per i quali NON ESISTE nemmeno un cardiologo di Pisa da cui
29 NON SIANO STATI VISITATI nel primo trimestre 2015
30 - Le due negazioni per la divisione si rappresentano con il doppio NOT EXISTS
31   • RIGA 4 e RIGA 5 confronto paziente per paziente per il quale
32   • RIGA 6 non esiste
33   • Da RIGA 8 a RIGA 11 un cardiologo di Pisa
34   • RIGA 12 tale per cui non esiste
35   • Da RIGA 13 a RIGA 21 una visita di quel paziente con quel cardiologo nel
36     primo trimestre del 2015 --> Utilizzo delle CORRELATED SUBQUERY
37 - Ottengo quindi, paziente per paziente, i pazienti visitati da tutti i
38   cardiologi di Pisa, almeno una volta nel primo trimestre 2015
39 */

```

```

1  /* Selezionare cognome e specializzazione dei medici la cui parcella è superiore
2   alla media delle parcelle della loro specializzazione e che, nell'anno 2011,
3   hanno visitato almeno un paziente che non avevano mai visitato prima */
4
5  WITH parcelle_mie_spec AS
6  (
7      SELECT M.Specializzazione,
8          AVG(M.Parcella) AS MediaParcella
9      FROM Medico M
10     GROUP BY M.Specializzazione
11 )
12 , medici_costosi AS
13 (
14     SELECT M.Matricola, M.Specializzazione, M.Cognome
15     FROM Medico M
16        NATURAL JOIN
17     parcelle_mie_spec PMS
18     WHERE M.Parcella > PMS.MediaParcella
19 )
20 SELECT DISTINCT MC.Cognome, MC.Specializzazione
21 FROM Visita V2011
22    INNER JOIN
23     medici_costosi MC ON V2011.Medico = MC.Matricola
24 WHERE YEAR(V2011.Data) = 2011
25    AND NOT EXISTS
26    (
27        SELECT *
28        FROM VISITA V_PREC
29        WHERE V_PREC.Data < V2011.Data
30            AND V_PREC.Medico = V2011.Medico
31            AND V_PREC.Paziente = V2011.Paziente
32    );
33
34 /*
35 - Il problema si suddivide in 3 punti: a, b, c
36 - a) Media delle parcelle di tutte le specializzazioni
37 - b) Medici(target) la cui parcella è > dei medici di a)
38 - c) Vengono controllati se i medici in b) hanno visitato almeno 1 paziente
39   nel 2011, che non avevano mai visitato prima
40 - La 1° CTE da RIGA 5 a RIGA 11 rappresenta il punto a)
41 - OSS: per il punto a) ho che 'tutte' nasconde un raggruppamento
42 - La 2° CTE da RIGA 12 a RIGA 19 rappresenta il punto b)
43 - OSS: il NATURAL JOIN da RIGA 15 a RIGA 18 viene fatto sull'attributo
44   omonimo che è Specializzazione
45 - OSS: La dicitura "ALMENO UN" non implica un conteggio ma il costrutto EXISTS
46 - La c) viene rappresentata nella QUERY FINALE da RIGA 20 a RIGA 32 e si può
47   tradurre così -> Proietto nome e cognome dei medici (senza duplicati) RIGA 20
48   che hanno la media > della media delle parcelle della loro specializzazione
49   JOIN da RIGA 21 a RIGA 23 e che hanno visitato un paziente X nel 2011 RIGA 24
50   per il quale NON ESISTE RIGA 25 una visita precedente a quella del 2011
51   RIGA 29 con quel medico RIGA 30 e con quel paziente X RIGA 31 --> Utilizzo
52   la CORRELATED SUBQUERY da RIGA 26 a RIGA 32 per riferirmi ai dati della visita
53   nella QUERY ESTERNA
54 - Quest'ultimo fatto rappresenta che quel paziente è stato visitato la prima volta
55   da quel medico
56 - OSS: RIGA 20 il DISTINCT si mette perchè un medico nel 2011 può aver visitato per
57   la prima volta più di un paziente
58 */

```

```

1  /* Impostare come mutuate (attributo omonimo pari a 1) tutte le visite
2   ortopediche che coinvolgono pazienti già visitati precedentemente almeno
3   due volte dallo stesso medico */
4
5  UPDATE Visita V
6  SET Mutuata = 1
7  WHERE 2 <=
8  (
9    SELECT D.Precedenti
10   FROM(
11     SELECT V1.Medico, V1.Paziente, V1.Data,
12       COUNT(*) AS Precedenti
13     FROM Visita V1
14      INNER JOIN
15        Medico M ON V1.Medico = M.Matricola
16      INNER JOIN
17        Visita V2 ON(
18          V2.Medico = V1.Medico
19          AND V2.Paziente = V1.Paziente
20        )
21      WHERE V2.Data < V1.Data
22      AND M.Specializzazione = 'Ortopedia'
23    GROUP BY V1.Medico, V1.Paziente, V1.Data
24  ) AS D
25  WHERE D.Paziente = V.Paziente
26  AND D.Medico = V.Medico
27  AND D.Data = V.Data
28 );
29
30 /*
31 - OSS: "Impostare" implica cambiare un valore di un attributo -> UPDATE
32 Devo mettere quindi mutuata a TRUE oppure a 1 RIGA 6
33 - Il senso del testo rappresenta che: Per ogni visita V se riesco a trovare
34 almeno 2 visite precedenti con uno stesso paziente P e uno stesso medico M,
35 allora quella visita V deve essere oggetto di aggiornamento
36 (sull' attributo mutuata)
37 - Devo inserire le mie condizioni in una derived table per poi selezionarla
38 e confrontarla per l'aggiornamento
39 - Utilizzo del SELF JOIN da RIGA 13 a RIGA 20: Si affiancano le stesse visite
40 dello stesso paziente RIGA 19 con lo stesso medico RIGA 18 per confrontarle
41 per avere le visite precedenti RIGA 21
42 - RIGA 12 Faccio il COUNT(*) poichè ho fatto il SELF JOIN affiancando a V1 le
43 visite di V2
44 - RIGA 18 e RIGA 19 Le due condizioni dentro la parentesi dell'ON si potrebbero
45 sostituire con -> USING(Medico,Paziente), visto che va a considerare gli
46 attributi omonimi
47 - RIGA 23 Raggruppo per gli attributi su V1 poichè devo farlo per ogni visita
48 che non sia precedente ma che abbia le precedenti, che le ha solo V1 e V2
49 rappresenta le precedenti
50 - In poche parole dentro la DERIVED TABLE da RIGA 10 a RIGA 24 ho preso tutte
51 le visite dentro Visita e ne ho contate quante sono state fatte precedentemente
52 da un medico ortopedico
53 - Impongo fuori dalla DERIVED TABLE l'aggiornamento proiettando il conteggio delle
54 visite precedenti derivanti dalla DERIVED TABLE RIGA 9 solo se ne esistano almeno
55 2 di visite precedenti RIGA 7, di quel medico RIGA 26 con quel paziente RIGA 25
56 - OSS: Quando prediamo una tabella e la blocchiamo in scrittura(aggiornamento)
57 -> Non la si può più leggere perchè non posso scrivere il seguente codice:
58 es( UPDATE Visita V
59   SET Mutuata = 1
60   WHERE 2 <=
61   (
62     "Conto delle visite V2 precedenti fatte da V.Medico con V.Paziente"
63     .
64     .
65     .
66     AND V2.Medico = V.Medico
67     AND V2.Paziente = V.Paziente
68   )
69 -> I due AND impongono che per ogni record di V devo calcolare un RESULT SET
70 L'errore che mi viene fuori sul database è il 1093 "You can't specify target
71 table 'V' for update in where clause -> Sto cercando di leggere la tabella
72 che sto modificando
73 */

```

```

1  /* Scrivere una query che restituisca il codice fiscale dei pazienti che sono
2   stati visitati sempre dallo stesso medico avente la parcella più alta,
3   in tutte le specializzazioni. Se, anche per una sola specializzazione, non
4   vi è un unico medico avente la parcella più alta, la query non deve
5   restituire alcun risultato. */
6
7  WITH ParcellaMassima AS
8  (
9      SELECT M.Specializzazione,
10         MAX(M.Parcella) AS ParcellaMax
11    FROM Medico M
12   GROUP BY M.Specializzazione
13 )
14 , MediciCostosi AS
15 (
16     SELECT M.Specializzazione, M.Matricola
17       FROM Medico M
18      NATURAL JOIN
19        ParcellaMassima PM
20     WHERE M.Parcella = PM.ParcellaMax
21   GROUP BY M.Specializzazione, M.Matricola
22   HAVING COUNT(*) = 1
23 )
24 SELECT V.Paziente AS Paz
25   FROM Visita V
26     INNER JOIN
27       MediciCostosi MC ON V.Medico = MC.Matricola
28 WHERE NOT EXISTS
29   (
30       SELECT *
31         FROM Visita V2
32           INNER JOIN
33             Medico M ON V2.Medico = M.Matricola
34           WHERE V2.Paziente = V.Paziente
35             AND V2.Medico <> V.Medico
36   )
37 GROUP BY V.Paziente
38 HAVING COUNT(DISTINCT MC.Specializzazione) =
39   (
40       SELECT COUNT(DISTINCT M3.Specializzazione)
41         FROM Medico M3
42           INNER JOIN
43             Visita V3 ON M3.Matricola = V3.Medico
44           WHERE V3.Paziente = Paz
45   );
46
47 /*
48 - OSS: La parola "SEMPRE" oppure "TUTTI" si può sostituire con "SOLO" oppure
49   "SOLAMENTE", facendo così quando posso fare questa trasformazione, senza
50   che il testo perda di significato ottengo: --> UNA DIFFERENZA
51 - Nella 1° CTE da RIGA 7 a RIGA 13 ottengo la parcella massima
52   specializzazione per specializzazione
53 - Nella 2° CTE da RIGA 14 a RIGA 23 prendo i medici con parcella massima
54   dalla 1° CTE da RIGA 17 a RIGA 19 e impongo che ce ne sia soltanto uno
55   RIGA 22 per ogni specializzazione RIGA 21, poichè così me lo impone il
56   testo dell'esercizio
57 - Facendo così considero, per ogni specializzazione, solamente il medico che
58   ha la parcella più alta
59 - Nella prima parte della Query finale da RIGA 24 a RIGA 27 prendo le visite
60   di un determinato paziente con i medici più costosi RIGA 27 laddove non
61   esiste RIGA 28 (SUBQUERY da RIGA 29 a RIGA 36) un'altra visita di quel
62   paziente per una determinata specializzazione da RIGA 31 a RIGA 34 con
63   un medico diverso RIGA 35
64 - OSS: La SUBQUERY deve essere verificata per tutte le specializzazioni per
65   le quali esiste almeno una visita di quel paziente --> CORRELATED!
66 - Raggruppo per paziente RIGA 37 e faccio una CORRELATED SUBQUERY nella
67   HAVING CLAUSE da RIGA 39 a RIGA 45, dove vengono contate tutte le
68   specializzazioni RIGA 40 per le quali è verificata la subquery,
69   cioè alle quali il paziente preso in considerazione si è rivolto RIGA 44
70 */

```

```

1  /* Scrivere una stored procedure per l'inserimento di una nuova terapia. Nel caso
2   in cui il paziente oggetto della terapia non abbia assunto in precedenza lo
3   stesso principio attivo, la terapia non deve essere inserita e deve essere
4   restituito un messaggio di errore del tipo: "Il paziente potrebbe essere
5   allergico al principio attivo X". Sostituire X con il nome del principio attivo
6   oggetto della terapia. La stored procedure non contenere istruzioni di tipo
7   CREATE. */
8
9  DROP PROCEDURE IF EXISTS inserisci_terapia;
10 DELIMITER $$ 
11 CREATE PROCEDURE inserisci_terapia(
12             IN paziente CHAR(100),
13             IN patologia CHAR(100),
14             IN dataEsordio DATE,
15             IN farmaco CHAR(100),
16             IN posologia INTEGER
17         )
18 BEGIN
19     DECLARE sicuro INTEGER DEFAULT 0;
20     DECLARE finito INTEGER DEFAULT 0;
21     DECLARE principioAttivoCur CHAR(100) DEFAULT '';
22     DECLARE principioAttivoPrescritto CHAR(100);
23     DECLARE terapiePassate CURSOR FOR
24         SELECT DISTINCT F.PrincipioAttivo
25             FROM Terapia T
26                 INNER JOIN
27                     Farmaco F ON T.Farmaco = F.NomeCommerciale
28                 WHERE T.Paziente = paziente;
29     DECLARE CONTINUE HANDLER FOR
30         NOT FOUND SET finito = 1;
31     SET principioAttivoPrescritto =
32     (
33         SELECT PrincipioAttivo
34             FROM Farmaco
35             WHERE NomeCommerciale = farmaco
36     );
37     OPEN terapiePassate;
38     prelievo: LOOP
39         FETCH terapiePassate INTO principioAttivoCur;
40         IF finito = 1 THEN
41             LEAVE prelievo;
42         END IF;
43         IF principioAttivoCur = principioAttivoPrescritto THEN      /* 13 */
44             BEGIN
45                 SET sicuro = 1;
46                 LEAVE prelievo;
47             END;
48         END IF;
49     END LOOP prelievo;
50     CLOSE terapiePassate;
51     IF sicuro = 1 THEN
52         INSERT INTO Terapia
53             VALUES (
54                 paziente, patologia, dataEsordio, farmaco,
55                 CURRENT_DATE, NULL,
56                 posologia
57             );
58     ELSE
59         SELECT CONCAT(
60             'Il paziente potrebbe essere allergico al principio attivo',
61             principioAttivoPrescritto
62         );
63     END IF;
64 END; $$ 
65 DELIMITER ;
66
67
68
69
70
71
72
73

```

```

74 /*
75 - FUNZIONAMENTO:
76 Si crea la procedura che, prima di inserire la nuova terapia con il farmaco
77 (dato in ingresso) con il principio attivo X, controlla la presenza del principio
78 attivo tra i farmaci di quel paziente (dato in ingresso), assunti in precedenza.
79 Il controllo viene fatto sulle terapie precedenti, tramite una query la quale,
80 recupera le terapie passate del paziente e le inserisce in un RESULT SET.
81 Utilizzo un cursore per scorrere questo RESULT SET che, parte puntando al primo
82 record e, scorrendo record per record confronta il principio attivo del farmaco
83 del paziente da inserire, con quello assunto in precedenza.
84 Non appena si trova il principio attivo, si rompe il ciclo e si procede
85 all'inserimento, altrimenti si mostra l'errore.
86 - RIGA 9: Va fatto inizialmente, altrimenti si incombe nell'errore che la procedura
87 è già esistente, e si mette il 'punto e virgola' perchè prima del cambio di
88 delimitatore la compilazione avviene ogni volta che si trova il
89 'punto e virgola'
90 - RIGA 10: Da qui in avanti dico al compilatore di non compilare più nulla, finchè
91 non si ritrova $$"
92 - Da RIGA 12 a RIGA 16: Passo alla procedura i parametri relativi alla terapia del
93 del paziente e non glieli passo tutti gli attributi, come ad esempio
94 DataInizioTerapia non gliela passo, perchè viene presa all'esecuzione, cioè
95 dalla CURRENT_DATE
96 - RIGA 18: Inizio del body della procedura
97 - RIGA 19: Variabile che mi serve per la verifica di inserimento o non inserimento
98 - RIGA 20: Variabile posta a 0 inizialmente, che poi verrà gestita dall'handler,
99 il quale la metterà ad 1; questa variabile mi rappresenta il 'fine corsa' del
100 RESULT SET, che tirerà fuori dopo il cursore
101 - RIGA 21: Variabile che servirà per fare il FETCH sui record presi dal cursore
102 - RIGA 22: Variabile che conterrà il principio attivo del nuovo farmaco
103 (dato in ingresso)
104 - Da RIGA 23 a RIGA 28: Chiamo un cursore che va nella Terapia, fa il JOIN con
105 il Farmaco per avere il principio attivo del farmaco delle terapie passate,
106 di un determinato paziente, laddove questo paziente è uguale al paziente preso
107 in ingresso, cioè quello della terapia da inserire RIGA 28
108 - RIGA 29 e RIGA 30: Metto l'handler di 'fine corsa' a 1 che, se non trovo nulla
109 da estrarre successivamente, vorrà dire che avrò finito il RESULT SET e, di
110 conseguenza andrò all'istruzione successiva con il CONTINUE
111 - Da RIGA 31 a 36: Imposto il principio attivo prescritto con il principio attivo
112 del farmaco preso in input RIGA 35
113 - RIGA 37 e RIGA 38: Apro il cursore facendo il ciclo
114 - RIGA 39: Faccio FETCH sui principi attivi che sono stati tirati fuori dal cursore
115 - Da RIGA 40 a RIGA 42: Controllo se ho finito di scorrere il RESULT SET e, se ho
116 finito lascio il ciclo andando a RIGA 51
117 - Da RIGA 43 a RIGA 48: Se il principio attivo della terapia passata, pescato dal
118 FETCH, è uguale al principio attivo che sto tentando di inserire, allora rompo
119 il ciclo, poichè ho trovato la terapia con lo stesso principio attivo, setto la
120 variabile 'sicuro' a 1 RIGA 45, la quale mi rappresenterà l'ok per il tentativo di
121 inserimento e vado a RIGA 51
122 - Da RIGA 51 a RIGA 57: Se sicuro=1, inserisco la nuova terapia con le variabili
123 prese in input
124 - Da RIGA 58 a RIGA 62: Altrimenti, messaggio di errore concatenando la stringa
125 di errore al principio attivo del farmaco prescritto (in input)
126 */

```

```

1  /* Scrivere una function che, ricevuto in ingresso il codice fiscale di
2   un paziente, restituisca il suo stato attuale di salute SS ottenuto
3   mediante l'espressione: SS=n*( (SUM i=1 to N of(wi*gi))^-1 )
4   dove 'n' è il numero di esordi attualmente in corso, 'gi' è la gravità
5   con cui la patologia è stata contratta nello esordio i-esimo, e 'wi' è
6   un coefficiente di penalizzazione pari a: 1 se l'esordio i-esimo non ha
7   terapie fallite; 1.5 se l'esordio i-esimo ha da 1 a 2 terapie fallite;
8   2.5 se l'esordio i-esimo ha più di 3 terapie fallite */
9
10 DROP FUNCTION IF EXISTS stato_salute;
11 DELIMITER $$ 
12 CREATE FUNCTION stato_salute(codiceFiscale CHAR(16))
13 RETURNS DOUBLE NOT DETERMINISTIC
14 BEGIN
15     DECLARE statoSalute DOUBLE DEFAULT 0;
16     DECLARE esordiInCorso INTEGER DEFAULT 0;
17     DECLARE gravita INTEGER DEFAULT 0;
18     DECLARE terapieFallite INTEGER DEFAULT 0;
19     DECLARE w DOUBLE DEFAULT 0;
20     DECLARE finito INTEGER DEFAULT 0;
21     DECLARE cursoleTerapie CURSOR FOR
22         SELECT D2.Paziente, D2.Gravita,
23             IFNULL(TerapieFallite, 0) AS TerapieFallite
24         FROM
25             SELECT E.Paziente, E.Patologia,
26                 E.DataEsordio, E.Gravita,
27                 COUNT(*) AS TerapieFallite
28         FROM Esordio E
29             NATURAL JOIN
30                 Terapia T
31             WHERE E.DataGuarigione IS NULL
32             AND
33                 T.DataFineTerapia IS NOT NULL
34             AND
35                 E.Paziente = codiceFiscale
36             GROUP BY E.Paziente, E.Patologia,
37                 E.DataEsordio, E.Gravita
38         ) AS D1
39         NATURAL RIGHT OUTER JOIN (
40             SELECT *
41             FROM Esordio E
42             WHERE E.DataGuarigione IS NULL
43             AND
44                 E.Paziente = codiceFiscale
45         ) AS D2;
46     DECLARE CONTINUE HANDLER FOR
47         NOT FOUND SET finito = 1;
48     OPEN cursoleTerapie;
49     scan: LOOP
50         FETCH cursoleTerapie INTO gravita, terapieFallite;
51         IF finito = 1 THEN
52             LEAVE scan;
53         END IF;
54         CASE
55             WHEN terapieFallite = 0 THEN
56                 SET w = 1;
57             WHEN terapieFallite BETWEEN 1 AND 2 THEN
58                 SET w = 1.5;
59             ELSE
60                 SET w = 2.5;
61             END CASE;
62             SET statoSalute = statoSalute + (w*gravita);
63             SET esordiInCorso = esordiInCorso + 1;
64     END LOOP scan;
65     CLOSE cursoleTerapie;
66     IF statoSalute <> 0 THEN
67         SET statoSalute = esordiInCorso*(IFNULL(1/statoSalute, 0));
68     ELSE
69         SET statoSalute = -1;
70     END IF;
71     RETURN statoSalute;
72 END $$ 
73 DELIMITER ;

```

```

74 /*
75 - Terapie fallite = Sono quelle concluse senza che l'esordio sia finito,
76 quindi DataGuarigione della tabella esordio è NULL
77 - Casi possibili per calcolare il coefficiente wi:
78   0 terapie fallite -> wi=1
79   1,2 terapie fallite -> wi=1,5
80   3+ terapie fallite -> wi=2,5
81 - CodiceFiscale paziente è in ingresso
82 - La stored function non ha variabili di output -> Restituisce sempre
83   un solo valore
84 - La stored function è di tipo NON DETERMINISTIC e può essere in 3 modi:
85   1) DICHIARATIVA
86   2) PROCEDURALE
87   3) 'MISTA', che è quella che sarà nella soluzione
88 - PROCEDIMENTO:
89   Dato un paziente P, si deve calcolare lo stato di salute SS, che dipende
90   dagli esordi in corso (tabella Esordio).
91   All'interno di questi esordi, devo vedere le terapie fallite e, alla fine
92   dovrò calcolarne la somma tramite un operatore di aggregazione (COUNT).
93   Il COUNT mi servirà per determinare il 'wi', cioè la gravità di 'i',
94   che è un attributo dell'esordio.
95 - Terapie fallite è data quando ho
96   (Tabella TERAPIA) DataFineTerapia=NOT NULL
97   AND
98   (Tabella ESORDIO) DataGuarigione=NULL
99 - RIGA 12: La funzione prende in ingresso CHAR(16), perchè 16 è la
100  lunghezza nel database del codice fiscale, nel dubbio mettiamo CHAR(100)
101 - RIGA 13: Metto DOUBLE, dato che devo fare le moltiplicazioni con i
102  valori 1.5, 2.5 ecc...
103  Il fatto che sia NON DETERMINISTIC invece, mi serve nel caso eseguissi
104  questa funzione più volte con lo stesso codice fiscale e, in tempi
105  diversi otterrei risultati diversi
106 - RIGA 15: Variabile che conterrà il risultato della function
107 - RIGA 16: Parametro 'n' della sommatoria nella formula SS, quindi è il
108  numero degli esordi in corso
109 - RIGA 17: Variabile che rappresenta la gravità e assume valori da 1 a 9
110 - RIGA 19: Variabile di tipo DOUBLE che rappresenta il coefficiente di
111  penalizzazione e assume i valori 1, 1.5 e 2.5
112 - RIGA 20: variabile di 'fine corsa', che verrà gestita poi da un cursore
113 - RIGA 23: IFNULL(...) -> Rappresenta se terapieFallite = NULL, restituisce 0,
114  altrimenti restituisci terapieFallite così com'è
115  Il caso di terapieFallite è dato se non avviene il raggruppamento nella
116  derived table D1 ottenuta da RIGA 25 a RIGA 38, quindi il NATURAL RIGHT
117  OUTER JOIN tra D1 e D2, da RIGA 40 a RIGA 45, non viene completato e, di
118  conseguenza non ci sono terapie fallite
119 - Da RIGA 25 a RIGA 27: Esordio per esordio, ne prendo la gravità che è il
120  'gi' della sommatoria nella formula SS, e conto anche le terapie ottenute
121  dalla avvenuta condizione nel WHERE da RIGA 31 a RIGA 35
122 - RIGA 29: Affianca ogni esordio ad una sua terapia, tramite gli attributi
123  chiave che hanno in comune entrambe le tabelle che è
124    -> Paziente, Patologia, DataEsordio
125 - Da RIGA 31 a RIGA 35: Tutta la condizione (con le due AND) rappresenta
126  la 'terapiaFallita' del paziente con il codice fiscale dato in
127  ingresso RIGA 35
128 - RIGA 36, RIGA 37: Raggruppo esordio per esordio, perchè devo andare a
129  vedere per ogni esordio quante terapie ci sono.
130  Il raggruppamento è fatto sugli attributi che formano la chiave della
131  tabella esordio + gravità che, non essendo chiave, è collegata con
132  la chiave per la presenza di una dipendenza funzionale, dalla chiave a
133  gravità
134 - Il risultato da RIGA 25 a RIGA 37, rappresenterebbe le terapie fallite
135  di un dato paziente preso in input, con il relativo conteggio, e il
136  tutto lo metto in una DERIVED TABLE D1 a RIGA 38
137 - RIGA 39: Faccio il JOIN NATURALE ESTERNO DX, con la tabella D1 appena
138  trovata e la tabella D2 da RIGA 40 a RIGA 45, che contiene tutti gli
139  esordi in corso del paziente con il codice fiscale dato in ingresso
140  RIGA 44 -> Questo join mi mantiene tutti record di D2
141 - RIGA 48: Apro il cursore che mi andrà a mettere i pesi delle gravità
142 - RIGA 51, RIGA 52: Se verificata questa condizione, sono arrivato in
143  fondo e quindi esco dal ciclo
144 - Da RIGA 54 a RIGA 61: Con l'istruzione CASE imposto il coefficiente
145  a seconda di quante terapie fallite ho, e lo faccio esordio per esordio
146 */

```

```

147 /*
148 - RIGA 62: Costruisco SS incrementalmente; grazie al LOOP aggiorno ogni
149 volta lo stato di salute comando a quello precedente il valore
150 appena ottenuto
151 - RIGA 63: Aggiorno l'esordio in corso, sommandoci a quello precedente
152 quello attuale
153 - Poiché l'espressione per il calcolo di SS è definita solamente nel
154 caso di esordi in corso, si sceglie di restituire il valore -1 se il
155 paziente, dato dal parametro della function, è sano RIGA 69
156 - RIGA 66: Se lo stato di salute è diverso da 0, il paziente non è sano
157 allora applico la formula per il calcolo di SS RIGA 67, dove
158 n=esordiInCorso che sarà moltiplicato per il valore della
159 sommatoria elevata a -1 -> Questa sommatoria è rappresentata dal valore
160 1/statoSalute (calcolata a RIGA 62)
161 IFNULL(1/statoSalute, 0) restituisce 0 se 1/statoSalute è NULL, cioè quando
162 non sono riuscito a trovare le terapie fallite, altrimenti faccio ritornare
163 lo stato di salute così com'è
164 - Un esempio di utilizzo della funzione può essere rappresentato così:
165   SELECT P.CodFiscale, stato_salute(P.CodFiscale)
166   FROM Paziente P;
167 */

```

```

1  /* Per un'analisi del tasso di incidenza dell'influenza, la clinica
2   desidera avere un report aggiornato dei casi che costantemente si
3   verificano. A tale scopo, implementare una materialized view
4   MV_INFLUENZA che contenga, per ciascuna città di provenienza dei
5   pazienti, nessuna esclusa, il nome della città, il totale di casi di
6   influenza, la loro durata media delle terapie in giorni, e il farmaco
7   assunto nel maggior numero di terapie, se esiste. Si richiede di
8   creare la materialized view e implementare il deferred full refresh
9   con cadenza mensile */
10
11  DROP TABLE IF EXISTS MV_INFLUENZA;
12  CREATE TABLE MV_INFLUENZA(
13      Città CHAR(100) NULL,
14      TotaleCasi INT(11) NULL,
15      DurataMedia DOUBLE NULL,
16      FarmacoPiuUsato CHAR(100) NULL,
17      PRIMARY KEY('Città')
18 );
19  DROP PROCEDURE IF EXISTS refresh_mv_influenza;
20  DELIMITER $$;
21  CREATE PROCEDURE refresh_mv_influenza()
22  BEGIN
23      DECLARE n_casi INTEGER DEFAULT 0;
24      DECLARE durata_media DOUBLE DEFAULT 0;
25      DECLARE farmaco_top CHAR(100);
26      DECLARE citta_cur CHAR(100) DEFAULT '';
27      DECLARE durata_cur INTEGER DEFAULT 0;
28      DECLARE farmaco_cur CHAR(100) DEFAULT '';
29      DECLARE n_terapie_cur INTEGER DEFAULT 0;
30      DECLARE somma_durate INTEGER DEFAULT 0;
31      DECLARE citta_previous CHAR(100) DEFAULT '';
32      DECLARE max_terapie INTEGER DEFAULT 0;
33      DECLARE n_farmaci INTEGER DEFAULT 0;
34      DECLARE finito INTEGER DEFAULT 0;
35      DECLARE CONTINUE HANDLER FOR
36          NOT FOUND SET finito=1;
37      DECLARE CURSOR usi_farmaco_per_citta FOR
38          SELECT D.Città,
39                  D.Farmaco,
40                  SUM(D.Durata),
41                  COUNT(*) AS n_terapie
42      FROM(
43          SELECT P.Città,
44                  DATEDIFF(
45                          IFNULL(T.DataFineTerapia, CURRENT_DATE),
46                          T.DataInizioTerapia
47                      )
48          T.Farmaco
49      FROM Esordio E
50          INNER JOIN
51              Paziente P ON E.Paziente = P.CodFiscale
52          NATURAL JOIN
53              Terapia T
54          WHERE E.Patologia = 'Influenza'
55      ) AS D
56          GROUP BY D.Città, D.Farmaco
57          ORDER BY D.Città
58      TRUNCATE MV_INFLUENZA;
59      OPEN usi_farmaco_per_citta
60      scan:LOOP
61          FETCH usi_farmaco_per_citta INTO citta_cur,
62                                      farmaco_cur,
63                                      durata_cur,
64                                      n_terapie_cur
65          IF finito = 1 THEN
66              LEAVE scan;
67          END IF;
68          IF citta_cur = citta_previous THEN
69              SET somma_durate = somma_durate + durata_cur;
70              SET n_farmaci = n_farmaci + 1;
71          IF farmaco_cur > farmaco_top THEN
72              SET max_terapie = n_terapie_cur;
73              SET farmaco_top = farmaco_cur;

```

```

74      ELSEIF n_terapie = max_terapie THEN
75          SET max_terapie = 0;
76          SET farmaco_top = '';
77      END IF;
78  ELSE
79      REPLACE INTO MV_INFLUENZA
80      VALUES(
81          citta_previous,
82          n_casi,
83          somma_durate/n_farmaci,
84          farmaco_top
85      );
86      SET somma_durate = durata_cur;
87      SET n_farmaci = 1;
88      SET max_terapie = n_terapie_cur;
89      SET farmaco_top = farmaco_cur;
90      SET n_casi =
91      (
92          SELECT COUNT(*)
93          FROM Esordio E
94          INNER JOIN
95              Paziente P ON E.Paziente = P.CodFiscale
96          WHERE E.Patologia = 'Influenza'
97          AND
98              P.Citta = citta_cur
99      );
100     END IF;
101 END LOOP
102 END $$
```

DROP EVENT IF EXISTS event_mv_influenza \$\$

```

103 CREATE EVENT event_mv_influenza
104 ON SCHEDULE EVERY 1 MONTH
105 DO
106     BEGIN
107         CALL refresh_mv_influenza();
108     END $$
```

/*

- RIGA 21: Creo la STORED PROCEDURE che verrà poi chiamata all'interno di un TRIGGER EVENT, per la cadenza mensile, per la proprietà del DEFERRED REFRESH
- RIGA 23: In questa variabile ci salverò il totale dei casi di influenza
- RIGA 24: In questa variabile ci salverò la durata media delle terapie in giorni
- RIGA 25: In questa variabile ci salverò il farmaco assunto nel maggior numero di terapie
- Essendo che userò un cursore a RIGA 37, dichiaro delle variabili per il FETCH del cursore che avranno come desinenza '_cur' e saranno dalla RIGA 26 alla RIGA 29
- RIGA 34: Variabile del cursore che nel caso verrà messa a 1 dalla 'campanella' dell'HANDLER, dichiarata dopo a RIGA 35 e RIGA 36
- Da RIGA 43 a RIGA 55: Derived nella quale ci sarà la città, il farmaco e la durata delle terapie relative alla patologia influenza
- RIGA 45: Considero tramite IFNULL se la data di fine terapia non c'è prendo la data attuale altrimenti prendo quella lì
- RIGA 40: Proietto la somma delle durate che mi servirà dopo (*) e andrà a finire nella variabile somma_durate
- RIGA 41: Proietto il numero delle terapie ottenute tramite la derived table
- RIGA 57: Ordino per 'citta' così le informazioni che vado ad ottenere dalla proiezione seguono l'ordine alfabetico sulle città
- La somma delle durate ottenuta qua (*) mi serve quando farò uno scorrimento procedurale, sulle città in ordine alfabetico. Per tutte le volte che troverò quel nome di città farò un conteggio che mi darà le terapie per quella città che chiamerò n_farmaci. La durata media delle terapie in giorni sarà data dal conto -> somma_durate/n_farmaci (RIGA 83)
- RIGA 58: Faccio il TRUNCATE della materialized view per il FLUSHING che farò
- RIGA 60: Faccio un ciclo per il refresh, sul result set (precedente) aperto a RIGA 59
- Da RIGA 61 a RIGA 64: In ogni record del cursore avrò i dati richiesti dal problema
- RIGA 65 e RIGA 66: Se l'handler suona finisco

```

147 /*
148 - Altrimenti posso iniziare la maniera procedurale
149 - RIGA 68: Se la città attuale è la stessa della precedente, cioè finchè mi
150   trovo sulla stessa città, alla somma della durata precedente aggiungo la somma
151   della durata attuale RIGA 69 così -> durata precedente(somma_durate)
152   + durata attuale (durata_cur), e al conteggio dei farmaci precedenti aggiungo
153   + 1 che riguarda la città attuale RIGA 70, e questo contatore dei farmaci mi
154   servirà poi per la media
155 - All'inizio dello scorrimento la città attuale non è uguale alla precedente perchè
156   l'ho inizializzata alla stringa vuota RIGA 31 e quindi vado a ricadere nel ramo
157   ELSEIF RIGA 74, andando poi ad invalidare il valore massimo mettendoci 0 RIGA 75,
158   e anche il farmaco top RIGA 76 mettendoci stringa vuota; faccio ciò perchè è
159   come se avessi cambiato città anche se parto da una città vuota.
160 Questo rinizializzazione è dovuta al fatto nel caso in cui mi trovi davanti ai
161 pari merito, dato che tutti i farmaci sono processati
162 - RIGA 71: Controllo se il farmaco attuale è maggiore del farmaco top precedente e
163   se vale allora aggiorno i valori RIGA 72 e RIGA 73 essendo in caso di superamento
164 - Se nello scorrere per città in ordine alfabeti, la città è cambiata allora
165   vado nel ramo ELSE RIGA 78
166 - Da RIGA 79 a RIGA 85: Scrivo nella materialized view i dati relativi alla città
167   precedente con la media calcolata -> somma_durate/n_farmaci e il farmaco_top
168   trovato al LOOP precedente; tutto questo inserisce un record nella materialized
169   view
170 - Da RIGA 86 a RIGA 89: Reinizializzo le variabili di controllo, di cui:
171   n_farmaci è posta a 1 perchè riguarda il primo di questa nuova città
172   max_terapie e farmaco_top sono rappresentati da questo valore corrente,
173   visto che non ce ne sono altri riguardo a questa nuova città
174 - Da RIGA 90 a RIGA 99: Recupero il numero di casi di Influenza con la città
175   cambiata a quella corrente RIGA 98, e questo numero recuperato andrà a finire
176   su n_casi, e poi ritorno nel loop come ho fatto per la città precedente
177 - Da RIGA 103 a RIGA 109: Gestisco un EVENT creato, che ogni mese richiama la
178   procedura, buttando via tutto e rimettendoci i dati nuovi di zecca, proprio
179   come è richiesto nel testo per un DEFERRED FULL REFRESH
180 */

```

```

1  /* Per un'analisi del tasso di incidenza dell'influenza, la clinica
2   desidera avere un report aggiornato dei casi che costantemente si
3   verificano. A tale scopo, implementare una materialized view
4   MV_INFLUENZA che contenga, per ciascuna città di provenienza dei
5   pazienti, nessuna esclusa, il nome della città, il totale di casi di
6   influenza, la loro durata media delle terapie in giorni, e il farmaco
7   assunto nel maggior numero di terapie, se esiste. Si richiede di
8   creare la materialized view e implementare l'incremental refresh */
9
10 DROP TABLE IF EXISTS MV2_INFLUENZA;
11 CREATE TABLE MV2_INFLUENZA(
12     Città CHAR(100) NULL,
13     TotaleCasi INT(11) NULL,
14     TotaleGiorni INT NULL,
15     NumeroTerapie INT NULL,
16     FarmacoPiuUsato CHAR(100) NULL,
17     PRIMARY KEY('Città')
18 );
19 CREATE TABLE LOG_TABLE(
20     Timest TIMESTAMP,
21     Città CHAR(100) NULL,
22     Farmaco CHAR(100) NULL,
23     Durata INT NULL,
24     PRIMARY KEY('Timest')
25 );
26 DROP TRIGGER IF EXISTS push_esordio_infl;
27 DELIMITER $$;
28 CREATE TRIGGER push_esordio_infl
29 AFTER INSERT ON Esordio
30 FOR EACH ROW
31 BEGIN
32     DECLARE citta_esordio CHAR(100) DEFAULT '';
33     IF NEW.Patologia='Influenza' THEN
34         SET citta_esordio=
35         (
36             SELECT P.Città
37             FROM Paziente P
38             WHERE P.CodFiscale = NEW.Paziente
39         );
40         INSERT INTO LOG_TABLE(Timest, Città)
41             VALUES(CURRENT_TIMESTAMP, citta_esordio);
42     END IF;
43 END $$;
44 DROP TRIGGER IF EXISTS push_start_therapy $$;
45 CREATE TRIGGER push_start_therapy
46 AFTER INSERT ON Terapia
47 FOR EACH ROW
48 BEGIN
49     DECLARE citta_paziente CHAR(100) DEFAULT '';
50     IF NEW.Patologia='Influenza' THEN
51         SET citta_paziente=
52         (
53             SELECT P.Città
54             FROM Paziente P
55             WHERE P.CodFiscale=NEW.Paziente
56         );
57         INSERT INTO LOG_TABLE(Timest, Città, Farmaco)
58             VALUES(CURRENT_TIMESTAMP, citta_paziente, NEW.Farmaco);
59     END IF;
60 END $$;
61 DROP TRIGGER IF EXISTS push_stop_therapy $$;
62 CREATE TRIGGER push_stop_therapy
63 AFTER UPDATE ON Terapia
64 FOR EACH ROW
65 BEGIN
66     DECLARE citta_paziente CHAR(100) DEFAULT '';
67     IF NEW.Patologia='Influenza'
68     AND (
69         NEW.DataFineTerapia IS NOT NULL
70         AND
71         OLD.DataFineTerapia IS NULL
72     )
73     THEN

```

```

74      SET citta_paziente =
75      (
76          SELECT P.Citta
77          FROM Paziente P
78          WHERE P.CodFiscale = NEW.Paziente
79      );
80      INSERT INTO LOG_TABLE
81      VALUES(
82          CURRENT_TIMESTAMP,
83          citta_paziente,
84          NEW.Farmaco,
85          DATEDIFF(NEW.DataFineTerapia, NEW.DataInizioTerapia)
86      );
87      END IF;
88  END $$
```

DROP PROCEDURE IF EXISTS incremental_refresh \$\$

```

90 CREATE PROCEDURE incremental_refresh(IN till DATE)
91 BEGIN
92     WITH changes AS
93     (
94         SELECT MV.Citta,
95             SUM(IF(LT.Farmaco IS NULL, 1, 0)) AS NuoviCasi
96             SUM(IF(LT.Farmaco IS NOT NULL
97                 AND
98                 LT.Durata IS NULL,
99                 1, 0)) AS NuoveTerapie
100            SUM(IF(LT.Farmaco IS NULL
101                AND
102                LT.Durata IS NULL,
103                LT.Durata, 0)) AS NuovaSommaDurate
104        FROM LOG_TABLE LT
105        WHERE DATE(LT.Timest) <= till
106        GROUP BY MV.Citta
107    )
108    REPLACE INTO MV2_INFLUENZA
109    SELECT CH.Citta,
110        MV.TotaleCasi + CH.NuoviCasi
111        MV.TotaleGiorni + CH.NuovaSommaDurate
112        MV.NumeroTerapie + CH.NuoveTerapie
113    FROM changes CH
114    NATURAL JOIN
115        MV2_INFLUENZA MV;
116    DELETE FROM LOG_TABLE
117    WHERE DATE(LT.Timest)<=till;
118 END $$
```

DELIMITER;

/*

- Considerando che MV_INFLUENZA sia stata già creata devo fare l'incrementale refresh
- Come primo passo per l'aggiornamento incrementale devo progettare la LOG TABLE
- Come struttura della LOG TABLE devo capire quali sono i RAW DATA coinvolti (in termini di tavole) e quali sono le operazioni DML che modificano tali RAW DATA
- Le operazioni DML sono:
 - (1) Inserimento in Esordio che modifica TotaleCasi
 - (2) Aggiornamento in Terapia che modifica DurataMedia
 - (3) Inserimento in Terapia che modifica FarmacoPiuUsato
- Per la LOG TABLE devo prendere le precedenti operazioni e vedere cosa è sufficiente tenere traccia
- Oltre a quelle operazioni DML mi servirà un TIMESTAMP che sarà anche chiave (poiché ricordo che se non c'è una data che è univoca ci deve essere sempre una chiave che è il TIMESTAMP)
- Il TIMESTAMP mi raffigura il momento in cui scatta il trigger di PUSH per fare le operazioni (1)(2)(3) e chiamerò 'Timest'
- Spiegazione di OPERAZIONE (1):
 - Disallinea la MV dai RAW DATA
 - Suppongo che gli esordi non si cancellino
 - Ci sono anche gli update su Esordio, quando si va a mettere la data di guarigione, ma non ci serve monitorare quando una persona è guarita per contare il totale dei casi

```

147 /*
148     • Per tenere traccia che un nuovo caso si sia verificato, basta che si
149     propaghi la città del paziente che chiamerò 'Citta'
150     • Con solo la parte di TIMESTAMP e Citta risco a fare l'aggiornamento per
151     questa OPERAZIONE (1) e, nella LOG TABLE basta che vi faccia un COUNT con
152     Citta per aggiornare TotaleCasi nella MV
153     --> Questa sarà il [PUSH 1]
154 - Spiegazione di OPERAZIONE (2):
155     • Monitoraggio dell'aggiornamento di DataFineTerapia che va a finire NOT NULL
156     --> Questo è così perchè sto considerando le terapie finite
157     OSS: Se avessi considerato la terapia in corso avrei dovuto monitorare sia
158     l'inizio che la fine
159     • Per tenere conto di cambiare la durata media, posso memorizzare la durata
160     in giorni della terapia, facendo la differenza nella MV tra la DataFineTerapia
161     e DataInizioTerapia --> Nuovo attributo Durata
162     • Dopo aver messo Durata nel caso mi servisse fare l'OPERAZIONE (1), cioè
163     l'inserimento di un nuovo Esordio --> In Durata metto NULL
164     • Nel caso mi servisse fare l'OPERAZIONE (3), cioè la durata di una terapia
165     appena iniziata --> In Durata anche qua metto NULL (visto che la terapia
166     ancora non è finita)
167     --> Questa sarà il [PUSH 2]
168 - Spiegazione di OPERAZIONE (3):
169     • Quando c'è da monitorare un inserimento di una nuova terapia che modifica il
170     farmaco più usato, devo aggiornare degli attributi per fare in modo di
171     diversificare il caso in cui quello che ho inserito sia un esordio oppure
172     sia una terapia --> ci metto quindi Farmaco come attributo della log table e
173     gli Esordi si distinguono dalle terapie perchè in questo attributo hanno NULL
174     nel Farmaco, cosicchè tutti i record che hanno Timest(con valori),
175     Citta(con valori) e Farmaco(=NULL) --> sono Esordi!
176     --> Questa sarà il [PUSH 3]
177 - LOG TABLE attributi: Timest|Citta|Farmaco|Durata
178 - Attraverso i Refresh si vanno a fondere le operazioni attuate nella LOG TABLE
179 con il vecchio stato della MV, facendo i tre trigger di PUSH, e la CREATE TABLE
180 per la LOG TABLE
181 - REFRESH di TotaleCasi (1) -> Contare i Record di ogni città della LOG TABLE
182 inseriti e si sommano a quelli relativi alla MV
183 - REFRESH di DurataMedia (2) -> Data una città considero i record associati ad
184 essa nella LOG TABLE, quelli dell'attributo Durata, e poi ne faccio AVG(Durata)
185 Mi accorgo però che nella MV la durata media non mi permette di fare il REFRESH
186 e quindi devo scindere l'attributo in 2 parti: TotaleGiorni e NumeroTerapie
187 - Facendo così ottengo la durata media e nel codice per ottenerla, mi basterà fare
188 --> "SELECT TotaleGiorni/NumeroTerapie AS 'DurataMedia'"
189 - Per fare quindi il REFRESH prendo dalla LOG TABLE:
190     • Il risultato ottenuto dall'OPERAZIONE (2) e lo metto in TotaleGiorni, visto che
191     la terapia è terminata; prendo quindi ciò che ha l'attributo Durata nella
192     LOG TABLE e lo sommo a TotaleGiorni (che è il nuovo attributo in MV)
193     • Il risultato ottenuto dall'OPERAZIONE (3) e lo metto in NumeroTerapie, che mi
194     darà quante terapie ci sono, perchè sono gli inizi delle terapie.
195     Conterò quante volte è presente questo valore (dalla LOG TABLE) e lo sommerò
196     a NumeroTerapie (che è il nuovo attributo in MV)
197 -----
198 SPIEGAZIONE CODICE
199 -----
200 - Creo la tabella della MV con DurataMedia spartita in TotaleGiorni e
201     NumeroTerapie da RIGA 11 a RIGA 18
202 - Da RIGA 12 a RIGA 16 i valori di default possono essere di tipo NULL, tranne
203     'Citta' RIGA 12, che è la chiave primaria dichiarata a RIGA 17
204 - Creo la LOG TABLE da RIGA 19 a RIGA 25
205 - TIMESTAMP di inserimento RIGA 20 ed è chiave primaria RIGA 24, così non assume
206     il valore NULL di default
207 - 'Citta' è quella dei pazienti RIGA 21, 'Farmaco' pure RIGA 22 e 'Durata' è
208     quella della terapia RIGA 23
209 - Da RIGA 26 a RIGA 43 uso un TRIGGER per l'inserimento del nuovo Esordio in
210     Influenza
211 - RIGA 32 mi dichiaro una variabile citta_esordio per recuperare la città dove
212     avviene l'esordio del caso di Influenza (che calcolerò dopo)
213 - RIGA 33 faccio scattare il TRIGGER solo se la patologia che viene inserita
214     è l'influenza (NEW.Patologia)
215 - Da RIGA 34 a RIGA 39 recupero la città del paziente 'influenzato', che viene
216     inserito (NEW.Paziente)
217 - RIGA 40 e RIGA 41 inserisco un record nel LOG del tipo :
218     [Timest, Citta, NULL, NULL] per il motivo successivo
219 */

```

```

220 /*
221 Timest e Citta sono le uniche due cose che si modificano e a RIGA 41 gli passo
222 il CURRENT_TIMESTAMP e citta_esordio (calcolata da RIGA 34 a RIGA 39)
223 Visto che a RIGA 40 non ho messo altri attributi, di default assumeranno i valori
224 che ci sono nella CREATE TABLE della LOG TABLE, cioè NULL sia per Farmaco RIGA 22,
225 sia per Durata RIGA 23
226 - Da RIGA 44 a RIGA 60 uso un TRIGGER per l'inserimento di un inizio di nuova
227 terapia in Influenza
228 - RIGA 50 anche qua faccio scattare il TRIGGER solo se la patologia è l'influenza
229 - Da RIGA 51 a RIGA 56 recupero la città del paziente 'influenzato'
230 - RIGA 57 e RIGA 58 inserisco un record nel LOG del tipo :
231 [Timest, Citta, Farmaco, NULL], con Durata = NULL, visto che non la sappiamo
232 - Da RIGA 61 a RIGA 88 uso un TRIGGER per l'inserimento di una fine terapia in
233 Influenza
234 - Da RIGA 67 a RIGA 72 faccio scattare il TRIGGER se la patologia è l'influenza e
235 contemporaneamente (la data di fine terapia che ho impostato non è più NULL
236 RIGA 69 ma lo era prima RIGA 71) --> Ciò vuol dire che questa è una terminazione
237 di una terapia
238 - Da RIGA 74 a RIGA 79 recupero la città del paziente 'influenzato'
239 - Da RIGA 80 a RIGA 86 inserisco un record nel LOG del tipo:
240 [Timest, Citta, Farmaco, Durata], la cui Durata è calcolata a RIGA 85
241 OSS: Le righe di codice in cui si recupera un paziente 'influenzato' potrebbero
242 essere messe in una function!
243 - Da RIGA 89 fino alla RIGA FINALE proseguo con l'INCREMENTAL REFRESH
244 - RIGA 90 'IN till DATE' è una data che mi indica fino a dove devo arrivare
245 - Da RIGA 92 a RIGA 107 uso una CTE (changes) che va a calcolare le informazioni
246 riepilogative all'interno della LOG TABLE, cioè le operazioni descritte
247 inizialmente (1)(2)(3)
248 - RIGA 105 la funzione DATE prende il TIMESTAMP e ne estrae la data
249 - RIGA 95 per andare a trovare i nuovi casi dalla LOG TABLE, devo controllare quelli
250 che hanno NULL dentro Farmaco e dentro Durata
251 Se il farmaco è NULL associo il record ad 1, 0 altrimenti
252 Facendo ciò, si sommano i rimanenti 1 che, sono tanti quanto sono i record della
253 città considerata, che hanno Farmaco=NULL (*)
254 Posso evitare di controllare poi, se la Durata=NULL, visto che è compresa nella
255 condizione precedente (*)
256 - Da RIGA 96 a RIGA 99 devo considerare per le nuove terapie quelle che non hanno
257 una Durata, perché non sono terminate, quindi hanno Durata=NULL nella LOG TABLE
258 Inoltre, hanno anche Farmaco NOT NULL
259 Se il predicato booleano vale, restituisco 1, 0 altrimenti, ed il tutto andrà
260 nella somma (SUM)
261 - Da RIGA 100 a RIGA 103 per le nuove durate faccio una somma verificando che Farmaco
262 sia NULL della LOG TABLE RIGA 100 e, contemporaneamente sia NULL anche Durata della
263 LOG TABLE RIGA 102
264 Restituisco quindi LT.Durata nel caso la condizione booleanata sia verificata,
265 0 altrimenti RIGA 103, e con la somma (SUM) ottengo la somma delle durate dei nuovi
266 casi di influenza
267 - Questo record così composto nella SELECT va a finire nella CTE
268 - Da RIGA 108 a RIGA 112 la REPLACE usa la CTE e aggiorna i dati della MV
269 controllando:
270     se il dato è presente lo aggiorna, altrimenti lo inserisce
271 - RIGA 113 pesco dalla CTE i dati necessari
272 - Nella MV metto il totale dei casi precedenti + quelli nuovi RIGA 110 e, faccio la
273 la stessa cosa per il totale dei giorni RIGA 11 e, pure per il numero delle terapie
274 RIGA 112 --> Facendo così si ottiene la combinazione tra la MV e la LOG TABLE
275 */

```

```

1  /* Classificare i farmaci a base di ketoprofene sulla base della durata
2   media delle terapie, per ciascuna delle loro indicazioni restituire
3   nome farmaco, patologia, durata media delle terapie e posizione in
4   classifica */
5
6  WITH durate_medicamenta AS
7  (
8      SELECT T.Farmaco, T.Patologia,
9          AVG(DATEDIFF(IFNULL(T.DataFineTerapia, CURRENT_DATE),
10                 T.DataInizioTerapia
11                 )
12             ) AS DurataMedia
13     FROM Terapia T
14     INNER JOIN
15         Farmaco F ON T.Farmaco=F.NomeCommerciale
16     WHERE F.PrincipioAttivo='Ketoprofene'
17     GROUP BY T.Farmaco, T.Patologia
18 )
19  SELECT DM.Farmaco, Dm.Patologia,
20     DM.DurataMedia
21     RANK() OVER(
22         PARTITION BY DM.Farmaco, DM.Patologia
23         ORDER BY DM.DurataMedia
24     )
25  FROM durate_medicamenta DM;
26
27 /*
28 - "Classificare" ci dovrebbe far pensare subito ad un RANK, e si fa quello
29   classico (il DENSE RANK non si considera)
30 - "... a base di ketoprofene" è una condizione che dovrà finire in un WHERE
31   (che è un vincolo sui record)
32 - Il criterio del RANK (della classifica) è la durata media delle terapie e,
33   va considerata la durata media più breve delle terapie, visto che un
34   farmaco è tanto più efficiente, quanto più è corta la sua durata (*)
35 - "... per ciascuna delle loro indicazioni" inanzitutto, considero la tabella
36   INDICAZIONI(Farmaco, Patologia, DoseGiornaliera, NumGiorni, AVita) e
37   valuto farmaco per farmaco con, ad ognugno di questi la patologia associata
38   --> Avrò quindi un'indicazione di ogni farmaco per ciascuna sua patologia
39   associata --> Della tabella INDICAZIONI, per ogni indicazione si prende la
40   coppia (Farmaco, Patologia), che è anche la chiave primaria della tabella
41 - "...restituire...la posizione in classifica" vuol dire restituire il RANK
42 - La durata media delle terapie non è un attributo classificabile
43   immediatamente ma, va calcolato in questa maniera:
44   1) Considero la 1° coppia (Farmaco, Patologia), dalla quale ne verrà fuori
45     un insieme di Terapie T1...Tn
46   2) Per ciascuna terapia di questo insieme ne ricavo la durata (in giorni)
47   3) Applico poi AVG su queste durate, ottendendo lo SCORE di ciascun farmaco
48     legato alla propria patologia
49   --> Ricapitolando il 1° passo sta nel calcolo delle durate e il 2° passo
50     sta nel calcolo della media delle durate
51 - Ripeto il procedimento precedente per il resto delle coppie
52   (Farmaco, Patologia) che rappresentano l'INDICAZIONE
53 - Tutti gli SCORE ottenuti li andrò a posizionare nella colonna a loro
54   interessata, effettuando la classifica secondo il criterio stipulato qua (*)
55 - Essendo che il testo non specifica se una terapia, sia o meno conclusa
56   --> Qua faremo riferimento anche alle terapie non concluse
57 */
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

```

```

74 /*
75 -----
76 SPIEGAZIONE CODICE
77 -----
78 - Da RIGA 6 a RIGA 18 utilizzo la CTE per il calcolo delle durate medie
79 - Da RIGA 13 a RIGA 15 ho preso le terapie e ci ho abbinato il farmaco con il JOIN
80 - RIGA 16 condizione di vincolo sui record (come da testo)
81 - RIGA 17 raggruppo così, visto che il testo mi impone di considerare INDICAZIONE
82     per INDICAZIONE
83 - RIGA 8 proietto gli attributi che ho raggruppato, i quali mi rappresentano i
84     record di ciascuna INDICAZIONE (visto che INDICAZIONI come chiave primaria ha
85     proprio la coppia (Farmaco, Patologia))
86 - Da RIGA 9 a RIGA 12 per ciascuna INDICAZIONE calcolo la media delle durate delle
87     terapie
88 - Per ciascuna terapia, calcolo la media della differenza in giorni (RIGA 9) fra:
89     1° RIGA 9 DataFineTerapia se non è NULL, altrimenti la CURRENT_DATE; questa data
90         è la più recente nella differenza --> Facendo così si considerano anche le
91         terapie attualmente in corso
92     2° RIGA 10 DataInizioTerapia, e questa data è la più remota nella differenza
93 - Dalla RIGA 25 prendo le durate medie derivate dalla CTE e proietto ciò che mi
94     richiedeva il testo cioè:
95     • L'indicazione ottenuta dalla coppia (Farmaco, Patologia) RIGA 19
96     • La durata media in giorni RIGA 20
97     • La classifica da RIGA 21 a RIGA 24
98 - La classifica avviene sulla sua partizione composta dalla coppia (Farmaco,
99     Patologia)
100    RIGA 22, visto che si richiedeva che il RANK dovesse essere fatto per ogni
101        INDICAZIONE; si ordina poi per DurataMedia, in maniera ascendente RIGA 23
102 - A parole potrei esprimere da RIGA 19 a RIGA 25, facendo un esempio solo su un'unica
103        coppia (Farmaco, Patologia), e poi vale anche per il restante:
104            "Quando considero (F1,P1) calcolo la durata media delle terapie, del Farmaco F1 per
105                la Patologia P1; faccio poi la classifica vedendo in che posizione si pongono, sullo
106                    stesso farmaco, sulla stessa patologia (dovute alla partizione) ordinati per la
107                        durata media"
108 */

```

```

1  /* Generare una tabella pivot che, per ogni medico (riga), indichi
2   quante visite ha fatto ciascun medico in ogni città */
3
4  SELECT GROUP_CONCAT(
5      CONCAT('COUNT(IF(Cittta= ''',T.Cittta, '''',1,NULL)
6          ) AS ''', T.Cittta, '''')
7      )
8      ) INTO @pivot_query
9  FROM(SELECT DISTINCT Cittta
10     FROM Paziente
11     ) AS T;
12  SET @pivot_query=
13  CONCAT('SELECT Medico, ',
14          @pivot_query,
15          'FROM Visita
16          INNER JOIN
17          Paziente ON Paziente = CodFiscale
18          GROUP BY Medico;');
19  PREPARE statement FROM @pivot_query;
20  EXECUTE statement;
21  DEALLOCATE PREPARE statement;
22
23 /*
24 - Nella tabella "...per ogni medico" diventerà la RIGA e "... indichi
25 quante visite ha fatto ciascun medico in ogni città " diventerà
26 la COLONNA
27 - Nella colonna si dovrà avere ogni città del paziente e, per ogni
28 città ci sarà da fare un conteggio su quanti pazienti sono stati
29 visitati
30 - Esempio PSEUDO TABELLA PIVOT (la faccio corta per RIGHE e COLONNE)
31 |MEDICO |PISA |ROMA |NAPOLI |
32 -----
33 |001    |2    |1    |1    |
34 -----
35 |003    |3    |1    |0    |
36 -----
37 |005    |7    |4    |10   |
38 - Il PIVOTING è di tipo DINAMICO e la dinamicità la devo avere nel
39 prendere le città che diventeranno le COLONNE
40 -----
41 SPIEGAZIONE CODICE
42 -----
43 - Dalla RIGA 4 alla RIGA 11 è una query che costruisce la dimensione della tabella,
44 cioè effettua la rotazione, che è quell'insieme di record che poi rappresenterà
45 le città con i relativi valori inseriti all'interno
46 - Considero dalla RIGA 9 alla RIGA 10 le città di provenienza dei pazienti, città
47 per città e, questa sarà una DERIVED TABLE che chiamerò T come scritto in RIGA 11
48 - Alla RIGA 4 uso la funzione GROUP_CONCAT che mi permette di mettere, separati da
49 virgola i record, che sono risultato di una query --> Alla RIGA 4 gli arrivano le
50 città, che è roba selezionata dalla DERIVED TABLE dal FROM da RIGA 9 a RIGA 11
51 - RIGA 5 e RIGA 6, concateno e faccio il COUNT(IF... che mi permette di generare un
52 elenco delle città, che è dinamico e che dipende dalle città che seleziono alla
53 RIGA 9 e RIGA 10 --> In questo RIGA 5 e RIGA 6 mi inserisce tante stringhe con i
54 nomi delle città, concatenate
55 - Le stringhe inserite RIGA 5 e RIGA 6 sono fatte da COUNT(IF(Cittta= alla città che
56 arriva dalla RIGA 11 e, se le città sono corrisposte metto 1 altrimenti metto NULL
57 - Chiamo quindi alla RIGA 6 l'attributo con la ridenominazione T.Cittta in modo
58 dinamico
59 - Inserisco il tutto nella variabile pivot_query RIGA 8
60 - Dalla RIGA 12 alla RIGA 18 costruisco la query di PIVOTING che, finirà nella
61 variabile pivot_query RIGA 12, che sarà settata nuovamente (visto che aveva già
62 dei valori inseriti alla RIGA 8)
63 - RIGA 13, RIGA 14 la concatenazione mi servirà farla, per avere una tabella con
64 gli attributi disposti in questa maniera: MEDICO|Cittta1|Cittta2| ... |CitttaN,
65 le cui città, disposte in maniera concatenata a loro volta, sono state pescate
66 dalla pivot_query ottenuta a RIGA 8
67 - Dalla RIGA 15 alla RIGA 18 faccio il JOIN tra Visita e Paziente e, poi raggruppo
68 su Medico, proiettando infine per Medico RIGA 13 e per tutti i COUNT(IF... sulle
69 città RIGA 14
70 - Dalla RIGA 19 alla RIGA 21 preparo tutto il PIVOTING, lo eseguo e poi lo dealloco
71 OSS: La parte dinamica dentro le due CONCAT (RIGA 5-RIGA 6 e dalla RIGA 13 alla
72 RIGA 18) --> È rappresentata dalle scritte con colore nero (diverso dal grigio) !
73 */

```