

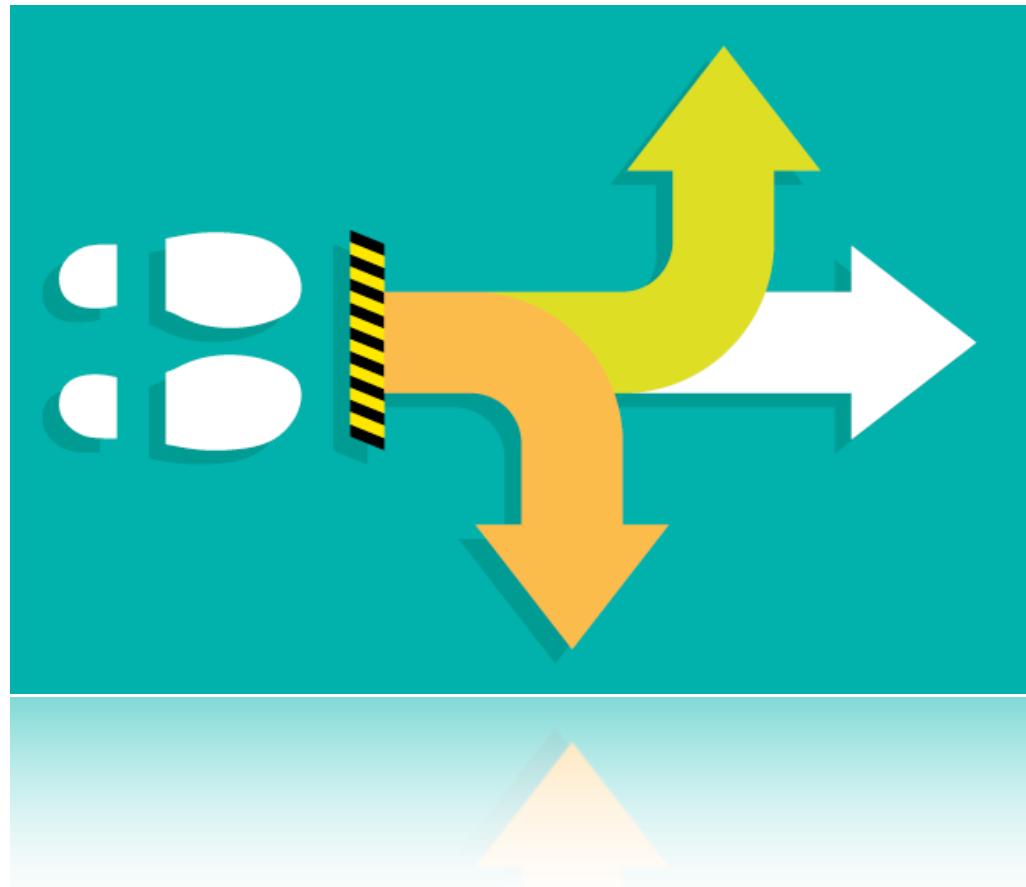
Basi di dati

Corso di laurea in Ingegneria Informatica
Scuola di Ingegneria – Università di Pisa

Oracle MySQL
A.A. 2017-2018

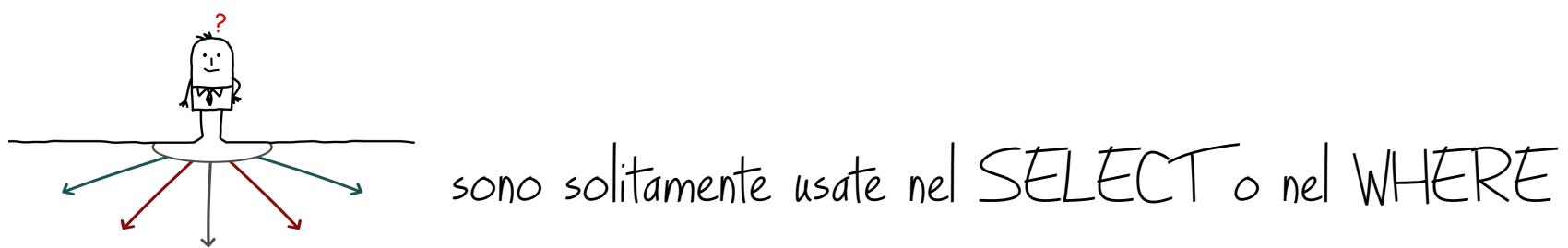
7
Ing. Francesco Pistolesi
Postdoctoral Researcher
Data Science and Engineering Lab
Dipartimento di Ingegneria dell'Informazione
francesco.pistolesi@iet.unipi.it

Funzioni per il controllo del flusso



Funzioni per il controllo del flusso

Sono funzioni che permettono di **modificare il valore di un attributo**



sono solitamente usate nel *SELECT* o nel *WHERE*

Funzione IF()

Per ciascun otorinolaringoiatra, indicarne la matricola e il numero di visite mutuate e non mutuate effettuate nell'anno in corso.

```
1  SELECT
2      M.Matricola
3      ,
4      SUM(
5          IF(V.Mutuata IS TRUE, 1, 0)
6      )
7      AS Mutuate
8      ,
9      SUM(
10         IF(V.Mutuata IS FALSE, 1, 0)
11     )
12     AS Non_Mutuate
13    FROM
14        Visita V
15        INNER JOIN
16            Medico M ON V.Medico = M.Matricola
17    WHERE
18        M.Specializzazione = 'Otorinolaringoiatria'
19        AND YEAR(V.`Data`) = YEAR(CURRENT_DATE)
20    GROUP BY
21        M.Matricola;
```

Matricola	Mutuate	Non_Mutuate
001	3	1
002	4	0
004	4	1

Funzione CASE()

Per ciascun otorinolaringoziatra, indicarne la matricola e un attributo contenente L se il numero di visite dell'anno corrente è inferiore a 5, oppure M se è superiore (o uguale) a 5 ma inferiore (o uguale) a 10, oppure H se è superiore a 10.

```
1  SELECT
2      M.Matricola
3
4      ,
5      CASE
6          WHEN COUNT(*) < 5 THEN 'L'
7          WHEN COUNT(*) BETWEEN 5 AND 10 THEN 'M'
8          WHEN COUNT(*) > 10 THEN 'H'
9      END
10     FROM
11         Visita V
12         INNER JOIN
13             Medico M ON V.Medico = M.Matricola
14     WHERE
15         M.Specializzazione = 'Otorinolaringoziatria'
16         AND YEAR(V.`Data`) = YEAR(CURRENT_DATE)
17     GROUP BY
18         M.Matricola;
```

Matricola	CASE
001	L
002	L
004	M

Funzione IFNULL()

Per ciascun otorinolaringoziatra, nessuno escluso, indicarne la matricola, il numero di visite effettuate nell'anno in corso e un attributo pari alla matricola se il medico ha effettuato almeno una visita nell'anno in corso, altrimenti pari a -1.

```
1  SELECT
2      M.Matricola,
3      COUNT(*),
4      IFNULL(D.Matricola, -1)
5  FROM
6  Medico M
7  NATURAL LEFT OUTER JOIN
8  ▼ (
9      SELECT M.Matricola
10     FROM
11        Visita V
12        INNER JOIN
13          Medico M ON V.Medico = M.Matricola
14    WHERE
15        M.Specializzazione = 'Otorinolaringoziatria'
16        AND YEAR(V.`Data`) = YEAR(CURRENT_DATE)
17  ▲ ) AS D
18  GROUP BY
19      M.Matricola;
```

Matricola	COUNT(*)	IFNULL(D.Matricola)
001	4	001
002	4	002
003	1	-1
004	5	004
005	1	-1
006	1	-1
007	1	-1
008	1	-1
009	1	-1
010	1	-1
011	1	-1
012	1	-1
013	1	-1
014	1	-1
015	1	-1
016	1	-1
017	1	-1
018	1	-1
019	1	-1
020	1	-1

Valori statici nella proiezione

derivati

Nella proiezione si possono aggiungere attributi contenenti un
valore costante record per record

Valori statici nella proiezione: esempio

Per ogni paziente **nessuno escluso**, indicarne il nome, il cognome e il numero di volte che è stato visitato dal dottor Amaranti.

...e se un paziente non è stato mai
visitato dal dottor Amaranti???



Valori statici nella proiezione: esempio

Per ogni paziente **nessuno escluso**, indicarne il nome, il cognome e il numero di volte che è stato visitato dal dottor Amaranti.

```
CREATE VIEW VisiteAmaranti AS  
    SELECT V.Paziente, COUNT(*) AS QuanteVisite  
        FROM Medico M INNER JOIN Visita V  
            ON M.Matricola = V.Medico  
        WHERE M.Cognome = 'Amaranti'  
        GROUP BY V.Paziente;
```

Si raggruppa sul codice fiscale perché è chiave di Paziente

Pazienti visitati da Amaranti

```
SELECT P.Nome, P.Cognome, VA.QuanteVisite  
    FROM Paziente P INNER JOIN VisiteAmaranti VA  
        ON P.CodFiscale = VA.Paziente  
UNION
```

Pazienti NON visitati da Amaranti

```
SELECT P.Nome, P.Cognome, 0  
    FROM Paziente P LEFT OUTER JOIN VisiteAmaranti VA  
        ON P.CodFiscale = VA.Paziente  
    WHERE VA.Paziente IS NULL;
```

Denota i pazienti non visitati da Amaranti, che il join sinistro conserva assegnando valori NULL a destra

Soluzione con IF()

Per ogni paziente **nessuno escluso**, indicarne il nome, il cognome e il numero di volte che è stato visitato dal dottor Amaranti.

```
1  SELECT
2      P.Nome,
3      P.Cognome,
4      IF(COUNT(*) = 1 AND D.Paziente IS NULL, 0, COUNT(*))
5  FROM
6      Paziente P
7      LEFT OUTER JOIN
8      (
9          SELECT V.Paziente
10         FROM
11             Visita V
12             INNER JOIN
13                 Medico M ON V.Medico = M.Matricola
14         WHERE
15             M.Cognome = 'Amaranti'
16     ) AS D
17     ON P.CodFiscale = D.Paziente
18 GROUP BY
19     P.CodFiscale,
20     P.Nome,
21     P.Cognome;
```

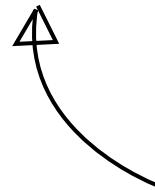
Nome	Cognome	IF(COUNT(*) = 1
Tommaso	Gatti	2
Maria	Cani	3
Maddalena	Bove	2
Romeo	Tigri	1
Roberto	Cavalli	2
Matilde	Fagiani	0
Epidonio	Gufi	0
Ultima	Gallina	3
Lorella	Nutrie	2
Osvaldo	Facoceri	2
Elettra	Faraona	2
Marcello	Orata	2
Maria Rita	Cinghiali	2
Elena	Leprotti	0
Egisto	Ricci	4
Norina	Pappagalli	2
Luigi	Cervi	2
Daniele	Galletti	0

Materialized view (a.k.a. snapshot)



Materialized view, che cosa sono?

un join corporo, un insieme di dati aggregati...

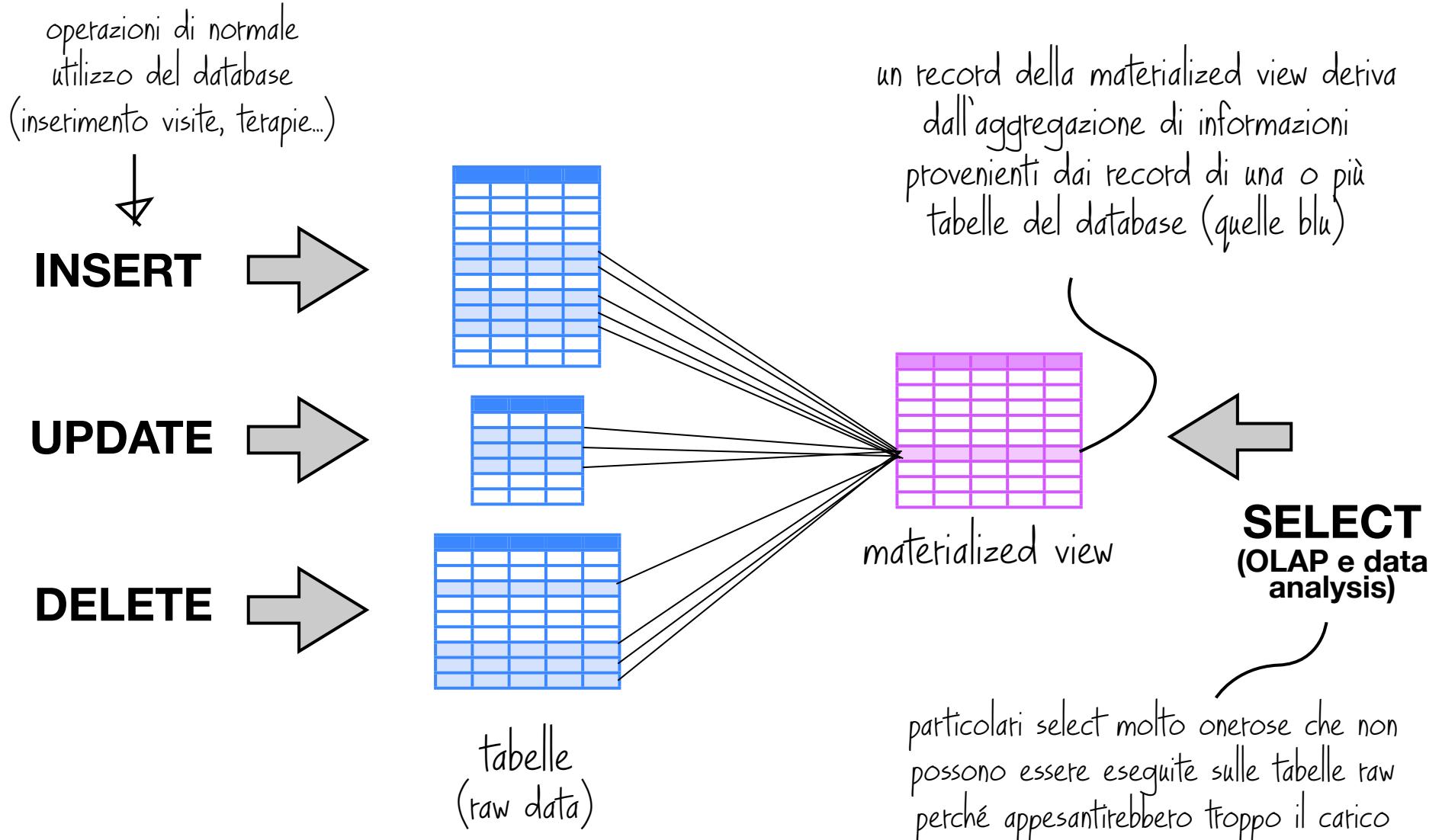


Una materialized view contiene il **risultato (pre-calcolato)** di una query



a differenza delle view, una materialized view non
è ricalcolata ogni volta che viene usata

Facciamo chiarezza



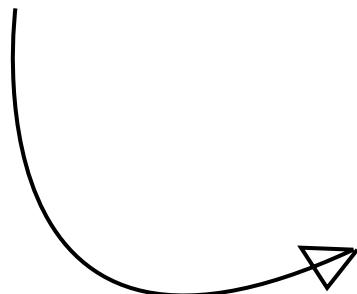
Ma allora???



Qual è la differenza fra
temporary table
e materialized view???

Temporary table vs. materialized view

Una materialized view è a tutti gli effetti una **tabella persistente**



al logout non viene distrutta come
avviene per le temporary table

Utilizzi

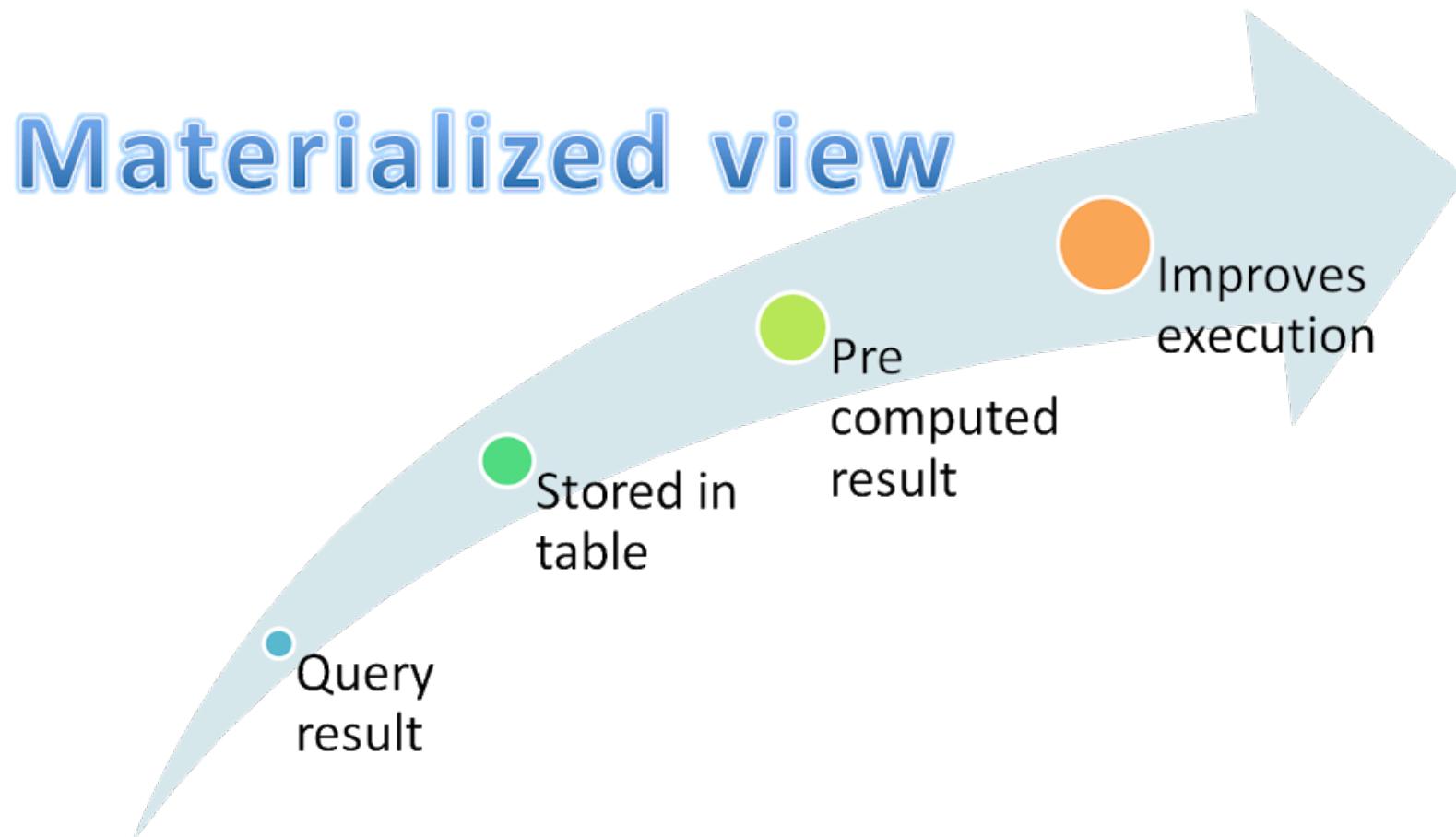
eventualmente anche non aggiornatissima

Si usano se c'è bisogno di una risposta veloce, e la query che restituisce il risultato della materialized view è **complessa e lenta nel restituirla**



da diversi minuti a ore

In sintesi



Da oggi le materialized view “ti regalano”...

- risultati di query complesse **già pronti**
- il risultato **senza ricalcolarlo**
- **indici ad hoc** per velocizzare l'accesso

a patto di accettare...

- possibilità di **informazioni non aggiornate**
- maggiore **occupazione di memoria**
- maggiore **carico sul server**



Refresh



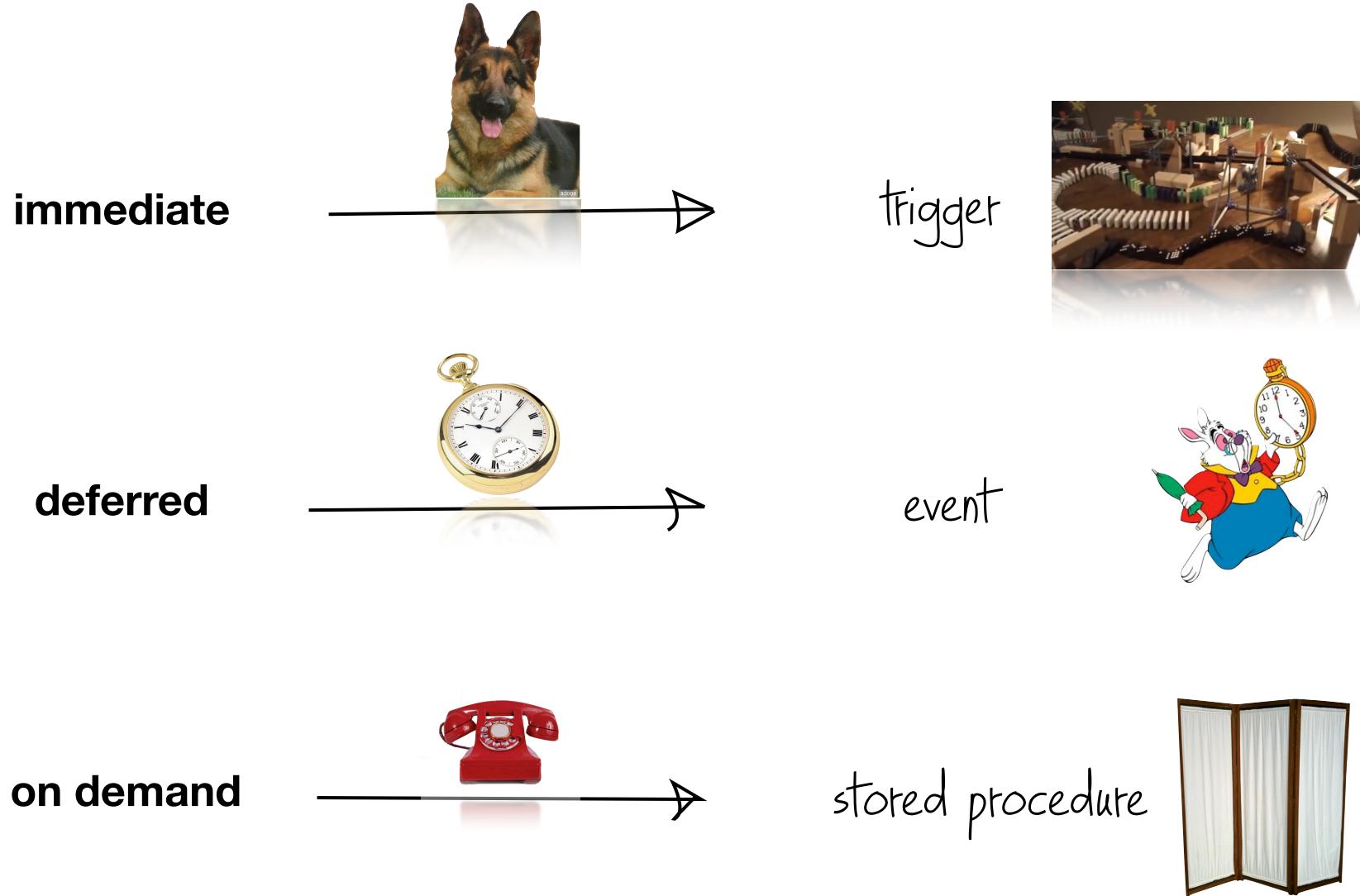
Refresh

Una materialized view può essere aggiornata secondo due modalità:

full refresh oppure **incremental refresh**



Politiche di refresh



Esempio

Creare una materialized view MV_RESOCONTO 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, l'immediate refresh e il deferred refresh.

Creazione

```
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;
```

in MySQL una materialized view è
fisicamente una tabella

Inizializzazione (build) 1 di 3

```
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;
```

Inizializzazione (build) 2 di 3

```
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
41             GROUP BY
42                 M2.Matricola
43     )
44 LIMIT 1;
```

Inizializzazione (build) 3 di 3

```
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;
```

On demand refresh

Rinnova 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



On demand refresh (1 di 2)

```
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
13        ROLLBACK;
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
```

On demand refresh (2 di 2)

```
21    -- full refresh
22    INSERT INTO MV_RESOCOMTO
23    SELECT
24        M.Specializzazione,
25        COUNT(*) AS Visite,
26        COUNT(DISTINCT V.Paziente) AS Pazienti,
27        IM.Incasso AS IncassoMeseCorrente,
28        MPP.MedicoMaxPazienti AS MedicoPiuPazienti
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 MPP
37    GROUP BY
38        M.Specializzazione;
39
40 ▲END $$
```

41

```
42 DELIMITER;
```

Perché si fa il flushing?

tipicamente

È più efficiente gettare il contenuto della materialized view
e **ricalcolarlo from scratch**

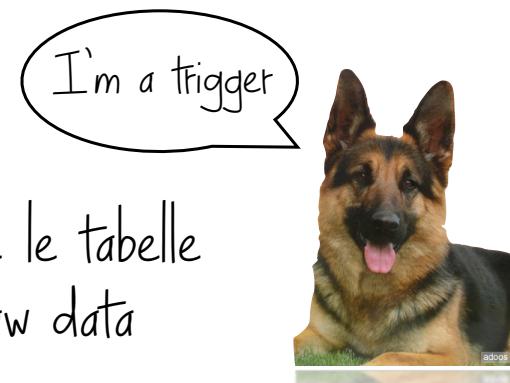


per sfruttare ciò che già è presente nella materialized view si dovrebbero esprimere condizioni articolate sulle tabelle su cui essa è basata, come condizioni temporali, condizioni sui valori degli attributi dei record ecc. Queste condizioni renderebbero più lento l'aggiornamento. Ecco perché il flushing conviene

Immediate refresh

Aggiorna immediatamente la materialized view a seguito della modifica delle tabelle (raw data) su cui si basa la materialized view

una schiera di cani lupo (trigger) monitorano costantemente le tabelle e mantengono la materialized view sincronizzata con i raw data



Immediate refresh (1 di 4)

```
1 DROP TRIGGER IF EXISTS resoconto_insert;
2
3 DELIMITER $$

4
5 CREATE TRIGGER resoconto_insert
6 AFTER INSERT ON Visita
7 FOR EACH ROW
8 BEGIN
9     -- variabili per recuperare i dati della materialized view
10    DECLARE tot_visite_prec INTEGER DEFAULT 0;
11    DECLARE tot_pazienti_prec INTEGER DEFAULT 0;
12    DECLARE tot_incasso_prec INTEGER DEFAULT 0;
13    DECLARE medico_più_pazienti_prec CHAR(100) DEFAULT '';
14
15    -- variabili per scrivere le condizioni
16    DECLARE specializzazione CHAR(100) DEFAULT '';
17    DECLARE pazienti_questo_medico INTEGER DEFAULT 0;
18    DECLARE pazienti_medico_top INTEGER DEFAULT 0;
19    DECLARE già_visitato INTEGER DEFAULT 0;
```

Immediate refresh (2 di 4)

```
21      -- recupero della specializzazione
22      SET specializzazione =
23      (
24          SELECT M.Specializzazione
25          FROM Medico M
26          WHERE M.Matricola = NEW.Medico
27      );
28
29      -- recupero dei dati attuali nella materialized view
30      SELECT MV.Visite,
31             MV.Pazienti,
32             MV.IncassoMese,
33             MV.MedicoPiuPazienti
34             INTO tot_visite_prec, tot_pazienti_prec,
35                  tot_incasso_prec, medico_più_pazienti_prec
36      FROM MV_RESOCONT0 MV
37      WHERE MV.Specializzazione = specializzazione;
38
```

Immediate refresh (3 di 4)

```
39    -- pazienti visitati dal medico del quale si sta inserendo la nuova visita
40    SET pazienti_questo_medico =
41    (
42        SELECT COUNT(DISTINCT V.Paziente)
43        FROM Visita V
44        WHERE V.Medico = NEW.Medico
45    );
46
47    -- pazienti visitati dal medico avente la specializzazione di questo medico,
48    -- il quale ha visitato più pazienti
49    -- NOTA: se è stata inserita una visita relativa al medico già in testa alla
50    -- classifica, questo resta in testa alla classifica. Impostiamo quindi a -1
51    -- il valore aggiornato delle visite
52    SET pazienti_medico_top =
53    (
54        SELECT IF(
55            NEW.medico <> medico_più_pazienti_prec,
56            COUNT(DISTINCT V.Paziente),
57            -1
58        )
59        FROM Visita V
60        WHERE V.Medico = medico_più_pazienti_prec
61    );
```

Immediate refresh (4 di 4)

```
62
63    -- controllo se il medico ha già visitato il paziente precedentemente
64    SET gia_visitato =
65    (
66        SELECT COUNT(*)
67        FROM Visita V
68        WHERE V.Medico = NEW.Medico
69            AND V.Paziente = NEW.Paziente
70            AND V.Data < NEW.Data
71    );
72
73    -- sincronizzazione della materialized view
74    REPLACE INTO RESOCONTO
75    VALUES
76    (
77        NEW.Specializzazione,
78        tot_visite_prec + 1,
79        IF(gia_visitato = 0, tot_pazienti_prec + 1, tot_pazienti_prec),
80        IF(pazienti_questo_medico > pazienti_medico_top, NEW.Medico, medico_più_pazienti_prec)
81    );
82
83 END $$
```

84

```
85 DELIMITER ;
```

Deferred refresh

Rinnova il contenuto della materialized view **rispettando una periodicità**

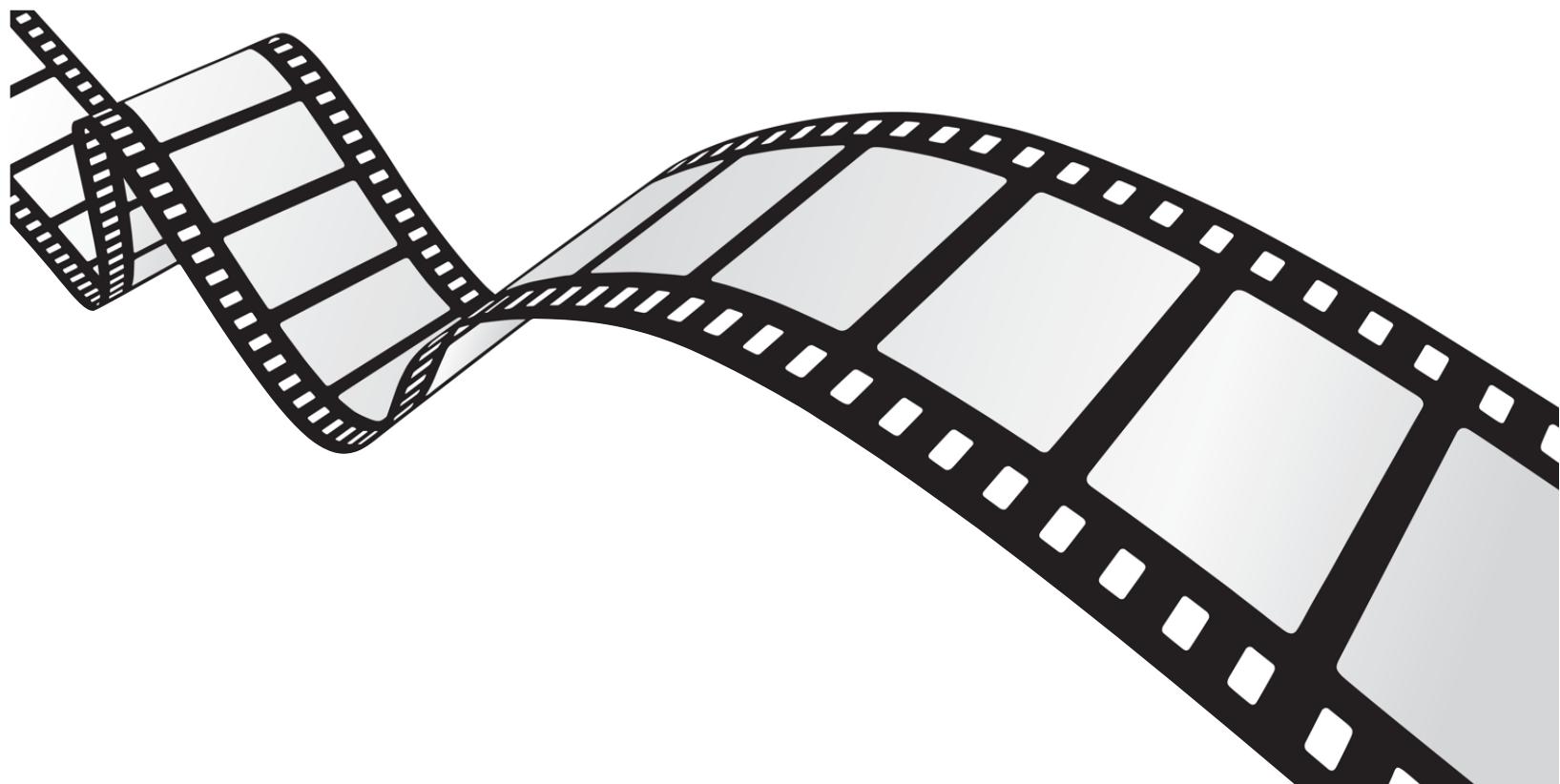
un event va in esecuzione solitamente in momenti in cui il carico applicativo è basso, aggiornando il contenuto della materialized view



Deferred refresh

```
1 DELIMITER $$  
2 CREATE EVENT Refresh_MV_Resoconto  
3 ON SCHEDULE EVERY 1 WEEK  
4 DO  
5 BEGIN  
6  
7     SET @esito = 0;  
8  
9     CALL refresh_MV_Resoconto(@esito);  
10  
11    IF @esito = 1  
12        SIGNAL SQLSTATE '45000'  
13        SET MESSAGE_TEXT = 'Errore refresh.' ;  
14    END IF;  
15  
16    END  
17 $$  
18 DELIMITER ;
```

Incremental refresh



Incremental refresh

trade-off aggiornamento/prestazioni

Funzionalità che permette di avere i dati di una materialized view aggiornati
solamente fino a un certo istante di tempo

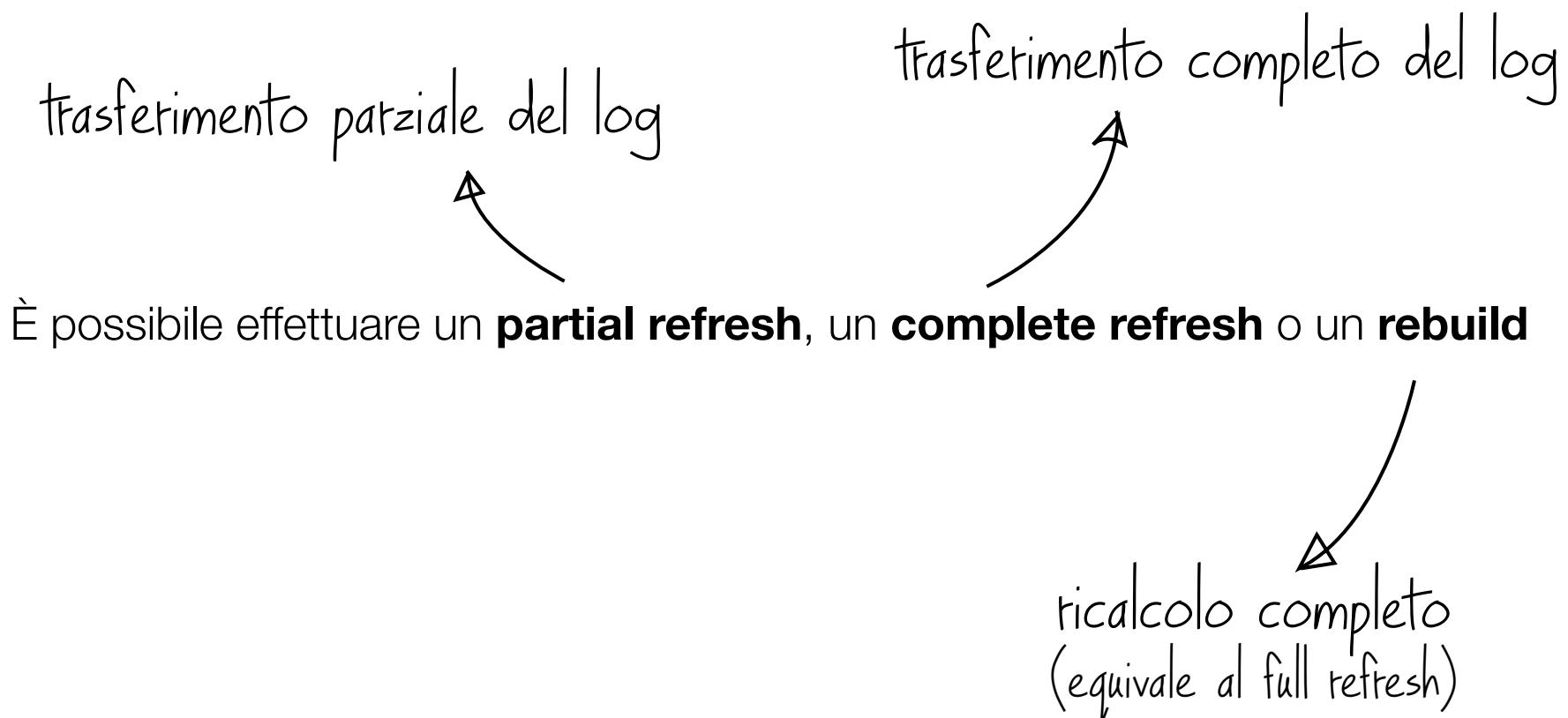


fra un aggiornamento e il successivo,
le modifiche alle tabelle raw sono
mantenute in un log

Partial vs. complete refresh



Modalità di incremental refresh



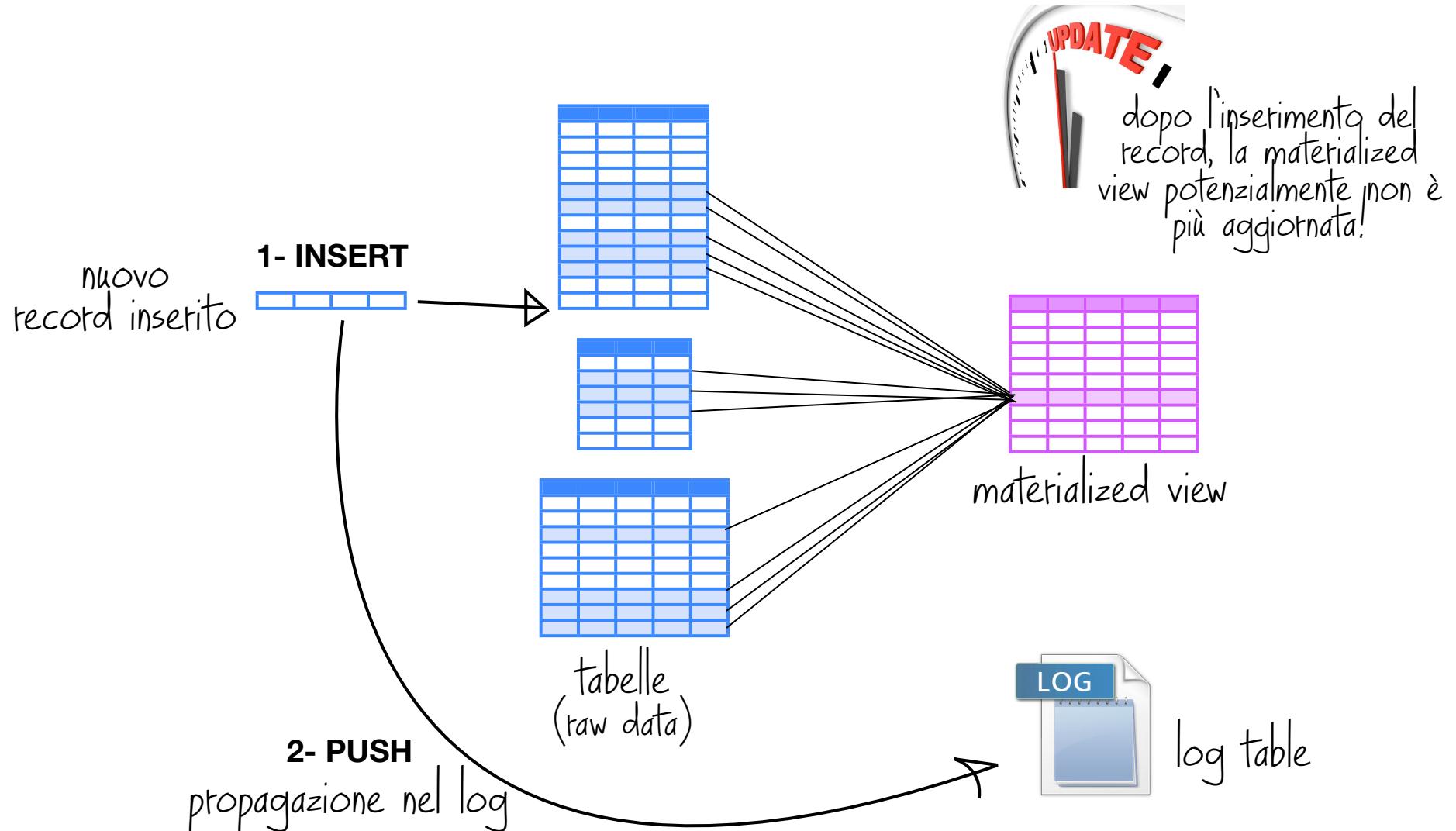
Perché l'incremental refresh è migliore?

Tramite l'incremental refresh è possibile aggiornare la materialized view sfruttando solo i **dati che essa contiene** e la **log table**

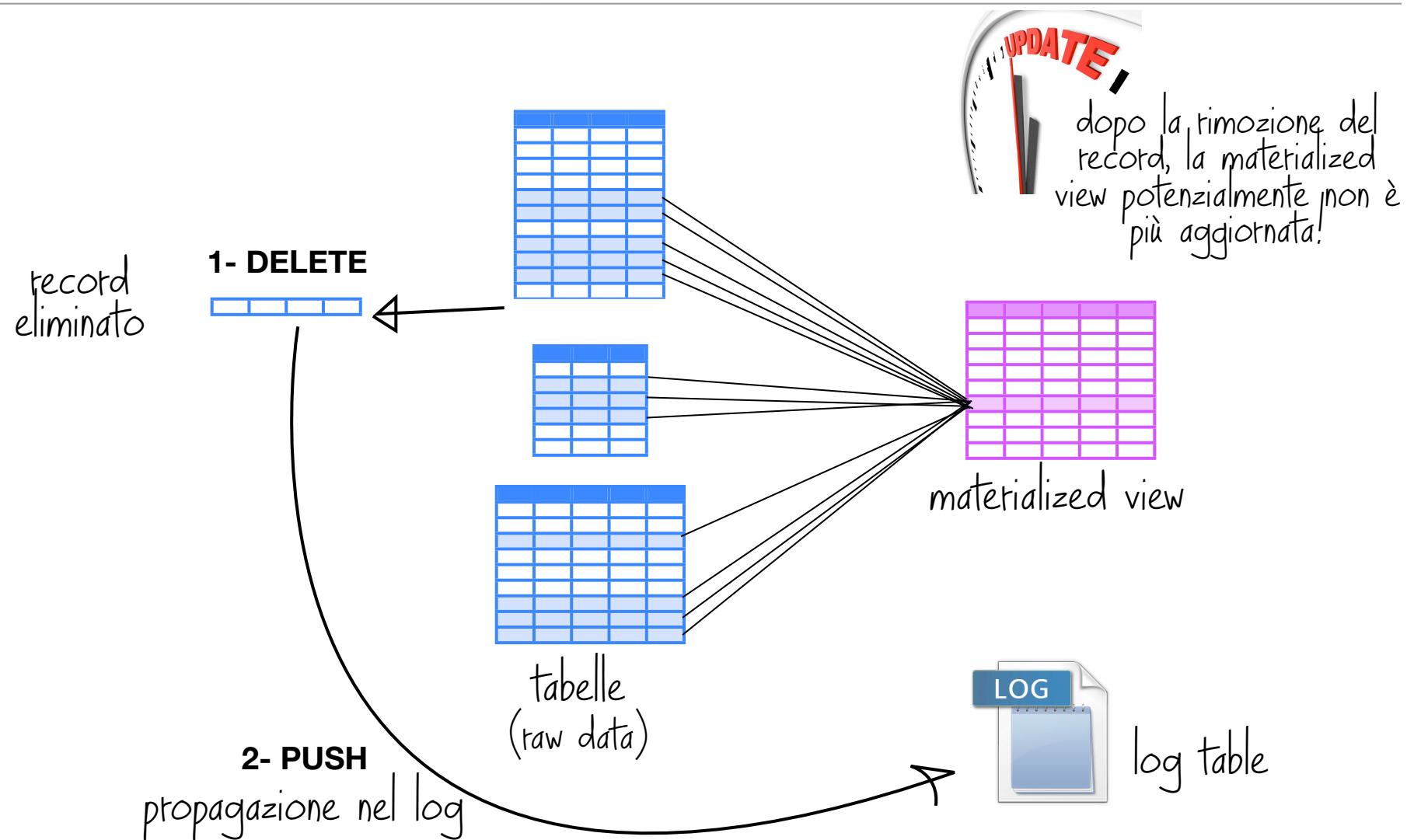
è molto più performante perché la log table ha molti meno record rispetto alle tabelle raw (quelle blu)



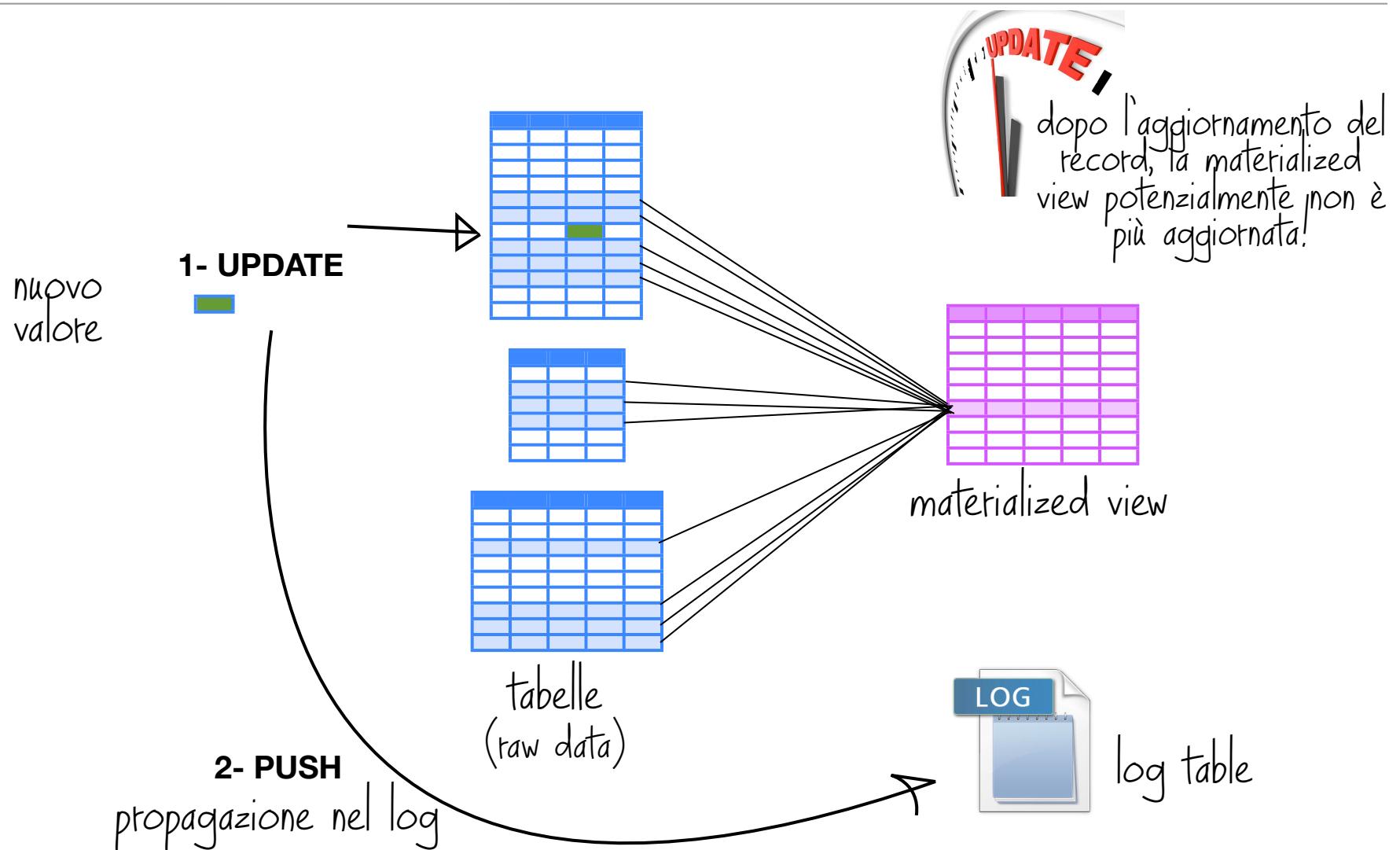
Come funziona: inserimento



Come funziona: cancellazione



Come funziona: aggiornamento

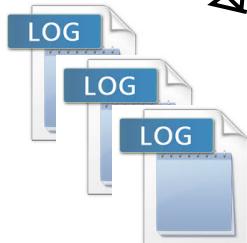


Cosa salvare nella log table?



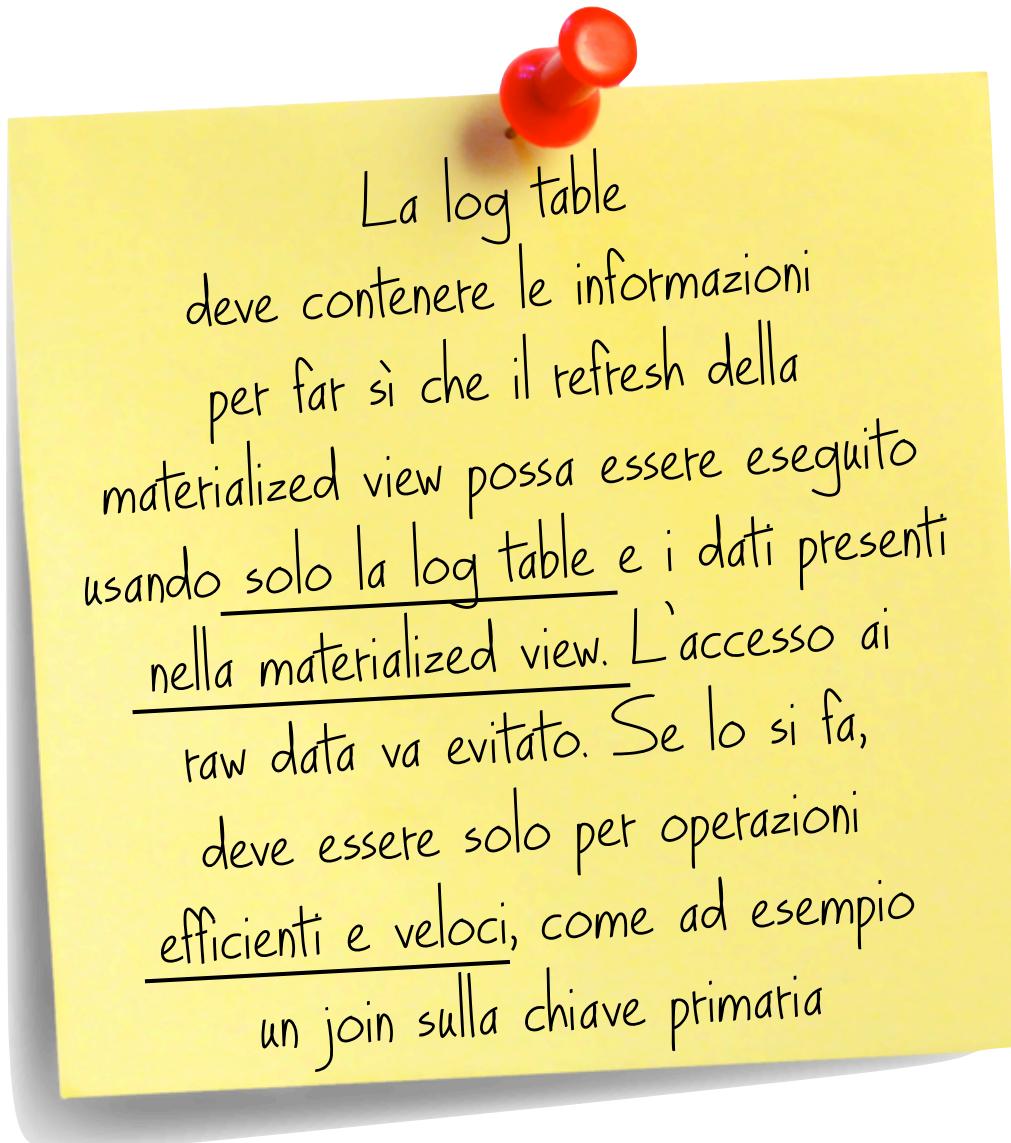
significa che quando si decide di effettuare il refresh, i dati nella log table devono essere sufficienti per effettuarlo (può bastare il banale push del record nella log table, oppure il push del record corredata da informazioni aggiuntive che vanno salvate nella log table per permettere poi di aggiornare la materialized view. In generale, è meglio evitare l'accesso ai raw data nel trigger di push.

La log table deve **tenere traccia** di tutte le modifiche delle tabelle del database sulle quali si basano i suoi dati

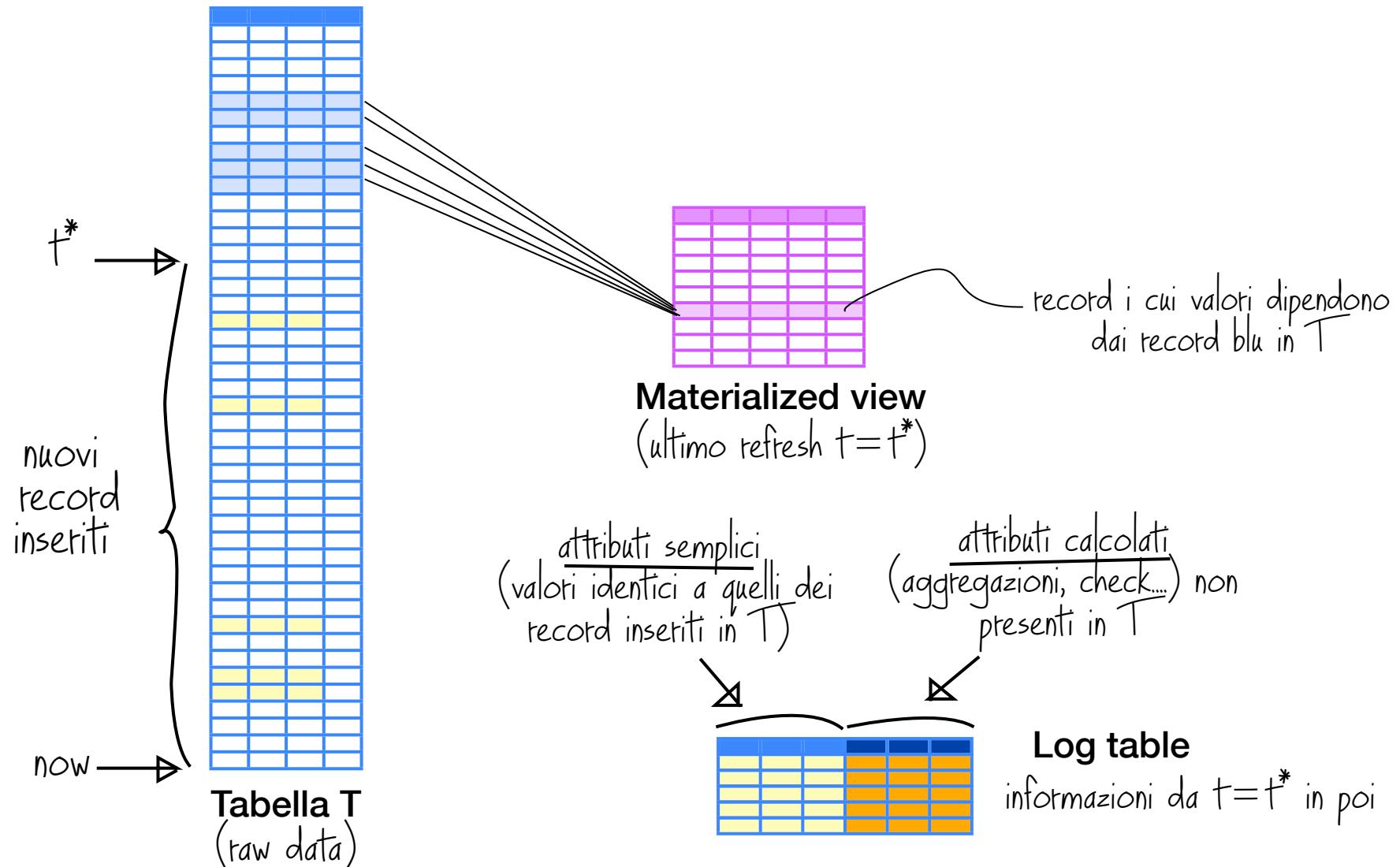


se la materialized view si basa su più tabelle, potrebbero essere utili più log table

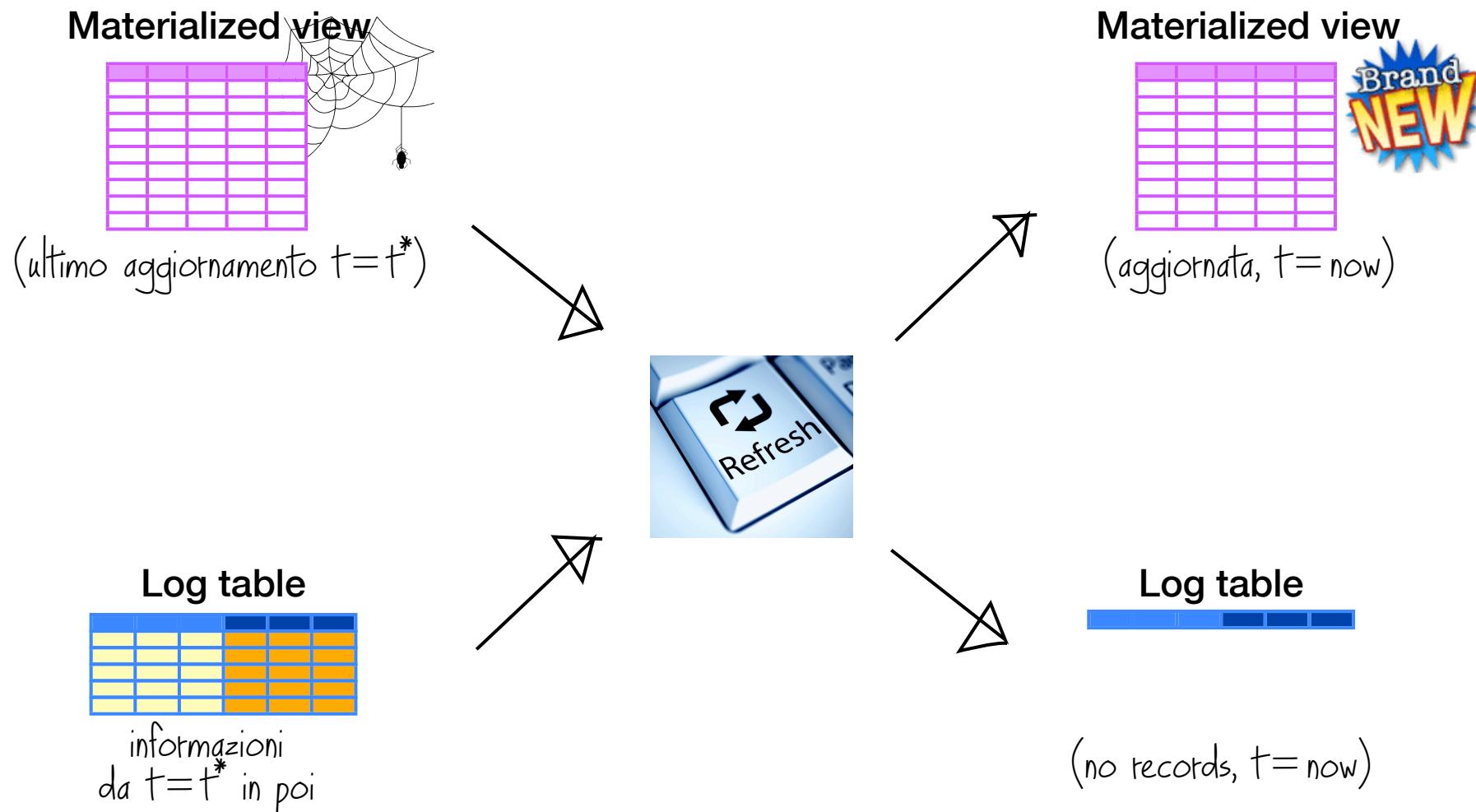
Ricorda



Fra un refresh e l'altro (consideriamo solo le insert)



Let's refresh! [complete]



Stesso esempio, ma incremental refresh

Creare una materialized view MV_RESOCONTO 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.

Esempio di contenuto

MV_RESOCONTO

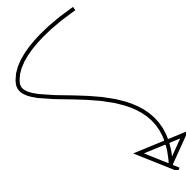
Specializzazione	Visite	NuoviPaz	IncassoMese	MedicoPiuPazienti
Cardiologia	138	125	16500	Rossi
Ortopedia	214	192	26750	Verdi
Neurologia	81	75	15200	Celesti
Otorinolaringoiatria	147	145	14500	Neri
Nefrologia	65	60	11700	Gialli



La materialized view contiene informazioni relative alle specializzazioni per le quali c'è almeno una visita nel mese in corso

Cosa deve contenere la log table?

Occorre analizzare ciascun attributo della materialized view e capire
quali informazioni ne influenzano il valore



ogni attributo della materialized view dipende dai valori presenti in
una tabella del database (potenzialmente anche più di una)

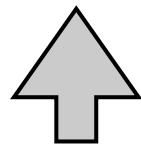
Attributo Visite

VISITA

Medico	Paziente	Data	Mutuata
012	kkw3	2016-05-30	1
03	slq6	2016-05-30	0
012	ppy9	2016-06-01	0
015	slq6	2016-06-03	1
011	hjt6	2016-06-03	0
06	ttw1	2016-06-03	0

MV_RESOCONTO

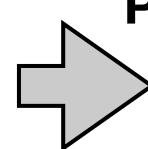
Specializzazione	Visite
Cardiologia	138
Ortopedia	214
Neurologia	81
Otorinolaringoiatria	147
Nefrologia	65



INSERT

nuova visita

016	slq6	2016-06-03	0
-----	------	------------	---



PUSH

016



La matricola del medico è sufficiente a determinare la sua specializzazione e quindi a capire quale record della materialized view dovrà essere incrementato al refresh

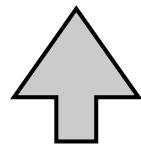
Attributo NuoviPaz (metodo 1)

VISITA

Medico	Paziente	Data	Mutuata
012	kkw3	2016-05-30	1
03	slq6	2016-05-30	0
012	ppy9	2016-06-01	0
015	slq6	2016-06-03	1
011	hjt6	2016-06-03	0
06	ttw1	2016-06-03	0

MV_RESOCONTO

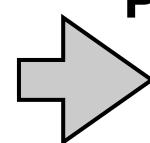
Specializzazione	NuoviPaz
Cardiologia	125
Ortopedia	192
Neurologia	75
Otorinolaringoiatria	145
Nefrologia	60



INSERT

nuova visita

016	slq6	2016-06-03	0
-----	------	------------	---



PUSH

1



Occorre verificate se il medico ha già visitato il paziente in precedenza. Si può fare il push di un booleano che vale 1 se il paziente non è stato visitato precedentemente dal medico, 0 altrimenti.

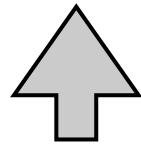
Attributo NuoviPaz (metodo 2)

VISITA

Medico	Paziente	Data	Mutuata
012	kkw3	2016-05-30	1
03	slq6	2016-05-30	0
012	ppy9	2016-06-01	0
015	slq6	2016-06-03	1
011	hjt6	2016-06-03	0
06	ttw1	2016-06-03	0

MV_RESOCONTO

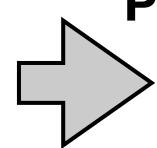
Specializzazione	NuoviPaz
Cardiologia	125
Ortopedia	192
Neurologia	75
Otorinolaringoiatria	145
Nefrologia	60



INSERT

nuova visita

016	slq6	2016-06-03	0
-----	------	------------	---



PUSH

2016-06-03



LOG

Invece di fare il controllo al momento del push e inserire un booleano nella log table, si può fare il push della sola data della visita appena inserita, e controllare poi in fase di refresh se 016 aveva già visitato slq6 prima di questa data. Così facendo, il push è più semplice e non accede ai raw data. La scelta fra metodo 1 e metodo 2 va fatta analizzando il carico del dbms.

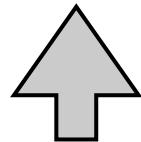
Attributo IncassoMese

VISITA

Medico	Paziente	Data	Mutuata
012	kkw3	2016-05-30	1
03	slq6	2016-05-30	0
012	ppy9	2016-06-01	0
015	slq6	2016-06-03	1
011	hjt6	2016-06-03	0
06	ttw1	2016-06-03	0

MV_RESOCONTO

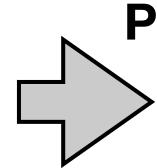
Specializzazione	IncassoMese
Cardiologia	16500
Ortopedia	26750
Neurologia	15200
Otorinolaringoiatria	14500
Nefrologia	11700



INSERT

nuova visita

016	slq6	2016-06-03	0
-----	------	------------	---



PUSH



Si è già fatto il push della matricola del medico che è sufficiente a ottenere la sua parcella in fase di refresh. Niente da aggiungere.

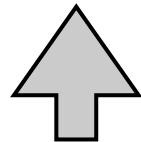
Attributo MedicoPiuPazienti

VISITA

Medico	Paziente	Data	Mutuata
012	kkw3	2016-05-30	1
03	slq6	2016-05-30	0
012	ppy9	2016-06-01	0
015	slq6	2016-06-03	1
011	hjt6	2016-06-03	0
06	ttw1	2016-06-03	0

MV_RESOCONTO

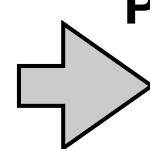
Specializzazione	MedicoPiuPazienti
Cardiologia	Rossi
Ortopedia	Verdi
Neurologia	Celesti
Otorinolaringoiatria	Neri
Nefrologia	Gialli



INSERT

nuova visita

016	slq6	2016-06-03	0
-----	------	------------	---



PUSH

slq6



Si è già fatto il push della matricola del medico, occorre però fare anche il push del codice fiscale del paziente per ottenere il numero di pazienti visitati da ogni medico, in fase di refresh.

Log table

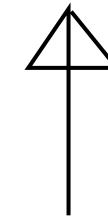
Occorre sempre il timestamp di inserimento, per poter effettuare il refresh incrementale con una determinata "recentness".
Ovviamente il timestamp funge anche chiave.



Istante	Paziente	Medico	NuovoPaz
---------	----------	--------	----------



Attributi semplici



Attributo calcolato
(dal trigger di push)

Stato della log table dopo la INSERT

LOG TABLE CON METODO 1

Istante	Paziente	Medico	NuovoPaz
2016-06-03 15:10:05	slq6	16	1

LOG TABLE CON METODO 2

Istante	Paziente	Medico	Data
2016-06-03 15:10:05	slq6	16	2016-06-03

NOTA: nella seconda versione della log table, ci accorgiamo che l'attributo Data di fatto è inutile salvarlo al momento del push. Il timestamp di inserimento (Istante) permette infatti di ricavare la data della visita usandolo come argomento della funzione date()

Quale log table scegliere?



La migliore log table è quella che rende il trigger di push più veloce e scattante possibile, **demandando al refresh** il carico computazionale.

Nel trigger di push occorre fare operazioni semplici. Il massimo della semplicità è la sola propagazione del record nella log table. Se il database ha una frequenza di insert dell'ordine del milione di inserimenti al minuto, il trigger di push deve essere velocissimo, perché va in esecuzione dopo ogni insert! In questi casi ci si limita alla propagazione. Se invece il database ha un minore volume di dati inseriti al minuto, le piccole operazioni fatte nel trigger di push (come quella della slide 54) sono accettabili, perché la mole di dati raw che manipolano non è sproporzionata, e non introduce quindi latenze intollerabili a seguito di ogni nuovo record inserito nelle tabelle raw. Queste operazioni rendono poi più veloce il refresh.

Log table

```
1 ▼CREATE TABLE RESOCONTO_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;
```

NOTA: qui si è scelto il metodo 1 per realizzare la log table

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 ;
```

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/Partial incremental refresh (1 di 2)

```
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 RESOCONT0_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
```

Complete/Partial incremental refresh (2 di 2)

```
41      UNION ALL
42      SELECT Specializzazione,
43              Visite,
44              Pazienti,
45              Incasso
46      FROM MV_RESOCENTO
47      ) AS D
48      NATURAL JOIN
49      MediciPiuPazienti MPP
50      GROUP BY D.Specializzazione;
51
52      IF metodo = 'full refresh' THEN
53          TRUNCATE TABLE RESOCENTO_LOG;
54      ELSE
55          DELETE FROM RESOCENTO_LOG
56          WHERE Istante <= istante_soglia;
57      END IF;
58
59      END;
60  ELSE
61      SET esito = 1;
62  END IF;
63 END $$ DELIMITER ;
```

Derived table D e processazione

Specializzazione	Visite	NuoviPaz	IncassoMese
Ortopedia	18	7	1980
Otorinolaringoiatria	16	16	2400
Nefrologia	23	12	3220
Cardiologia	138	125	16500
Ortopedia	214	192	26750
Neurologia	81	75	15200
Otorinolaringoiatria	147	145	14500
Nefrologia	65	60	11700

Nuovi dati
(dalla log table)

Vecchi dati
(materialized view)



D unisce i nuovi dati ottenuti dalla log table con i vecchi dati della materialized view. La query wrapper (in grigio nelle slide precedenti) affianca a ogni record il rispettivo medico con più pazienti, dopodiché fonde i dati specializzazione per specializzazione.

Ma non si doveva usare solo la log table?!

Di fatto è utilizzata solo la log table. Il join con la tabella Medico serve solo per **recuperare le informazioni del medico, utilizzandone la primary key**.



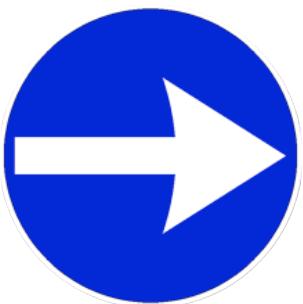
... dalla slide 64

```
SELECT M.Specializzazione,  
       COUNT(*) AS Visite,  
       SUM(NuovoPaz) AS Pazienti,  
       SUM(M.Parcella) AS Incasso  
  FROM RESOCONT0_LOG RL  
    || INNER JOIN  
      Medico M ON RL.Medico = M.Matricola  
 WHERE Istante <= IF(methodo='full refresh',  
                      CURRENT_TIMESTAMP,  
                      istante_soglia)  
 GROUP BY M.Specializzazione
```

Concludendo



Durante il refresh di una materialized view con politica incremental,
non si può usare la tabella su cui è basata la materialized view,
ma solo i dati già presenti nella materialized view, la log table e tabelle raw che
permettono di recuperare efficientemente dettagli che occorrono per sfruttare i
record della log table. Se la materialized view è basata sull'evoluzione dei dati di
più tabelle, queste **non possono essere usate durante il refresh**. Esisterà
una log table che tiene traccia delle modifiche di ciascuna, oppure una sola log
table più articolata.



Il push deve essere semplice e veloce. Attenzione a non rendere il push
troppo articolato per semplificare il refresh. Così facendo si affossano le
prestazioni del database perché il/i trigger di push va/vanno in esecuzione ogni
volta che avviene una modifica (inserimento, modifica o cancellazione) nella/e
tabella/e su cui la materialized view è basata. Push e refresh gestiscono una
coperta corta: se si semplifica al massimo il push, si rende pesantissimo il
refresh, e viceversa. È sempre meglio **optare per un refresh molto più
pesante del push**, perché il refresh viene opportunamente schedulato in
momenti in cui il carico del database è ridotto (per esempio la notte).