

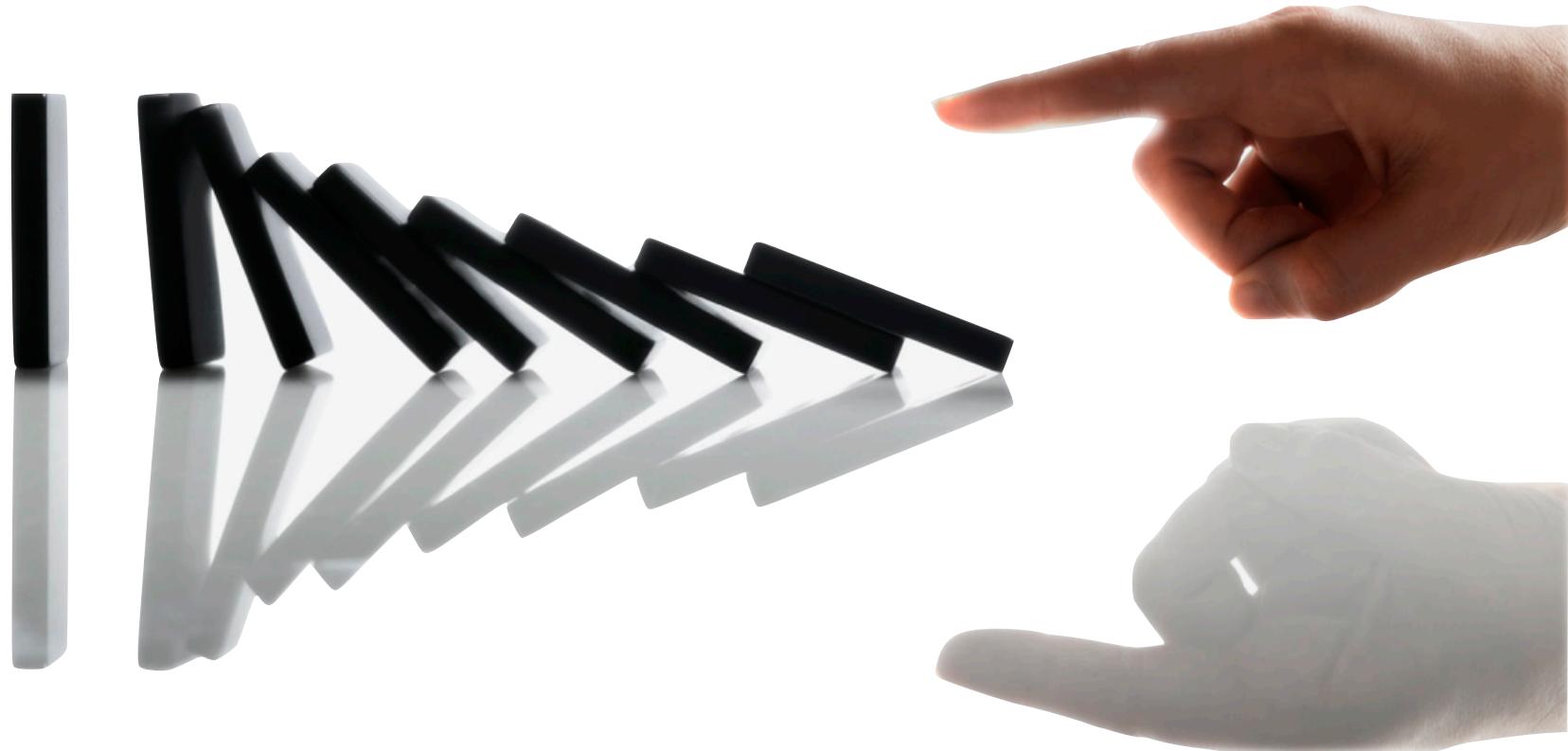
Basi di dati cod. 861II [9 CFU]

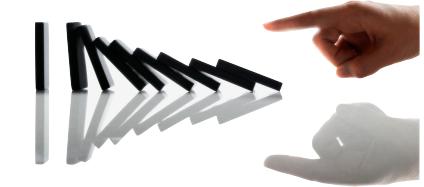
Corso di Laurea in Ingegneria Informatica

Oracle MySQL
A.A. 2022-2023

Francesco Pistolesi
Dipartimento di Ingegneria dell'Informazione
Università di Pisa
francesco.pistolesi@unipi.it

Database attivi





Database attivi

cause scatenanti (istruzioni DML o istanti temporali)

Sono dotati di una parte **reattiva** che permette loro di reagire ai cambiamenti mediante comportamenti specifici

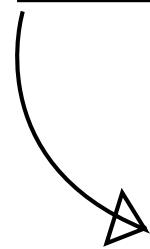
particolari operazioni sui dati

Trigger



Trigger

È un **insieme di istruzioni** eseguite al verificarsi
di una causa scatenante.



inserimento, aggiornamento, cancellazione

La causa è un comando DML



A cosa servono i trigger



detti vincoli di integrità generici
(o business rule)

Un trigger è capace di **gestire vincoli**, oppure è usato per **aggiornare ridondanze** presenti nel database



quando l'utente aggiorna il database, la modifica è propagata alle ridondanze coinvolte mediante un trigger

Sintassi MySQL per i trigger



```
DROP TRIGGER IF EXISTS nome_trigger;  
CREATE TRIGGER nome_trigger  
[BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON TabellaTarget  
FOR EACH ROW blocco_istruzioni
```



Crea il trigger nome_trigger.
Prima (dopo) che una o più row siano (siano state) inserite o modificate nella TabellaTarget, o cancellate da essa, per ciascuna, esegui il blocco_istruzioni.

Tipi di trigger: before vs. after

operazioni sugli attributi della row
che si sta inserendo/modificando, controllo di vincoli...



Con **BEFORE** si indica un'azione di **preprocessing**, mentre
AFTER denota un'azione **a posteriori**, o comunque "collaterale"



una conseguenza, come ad esempio
un'operazione su un'altra tabella

Business rule



Business rule (e vincoli di integrità generici)

Permettono di vietare determinati **scenari incoerenti** in cui potrebbero trovarsi i dati

 vincoli legati alla realtà aziendale in cui i dati si collocano
(vera accezione di business rule)

Esempi di business rule

I numero di visite per paziente
non può superare 5

Ogni mese, le visite non mutuate di un medico
non possono essere più di 10

Non si accettano medici con parcella
superiore a una certa parcella massima

I pazienti devono essere maggiorenni

Esempio di vincolo (business rule)

```
1  DROP TRIGGER IF EXISTS blocca_non_mutuate;
2  DELIMITER $$

3
4  CREATE TRIGGER blocca_non_mutuate
5    BEFORE INSERT ON Visita
6    FOR EACH ROW
7    BEGIN

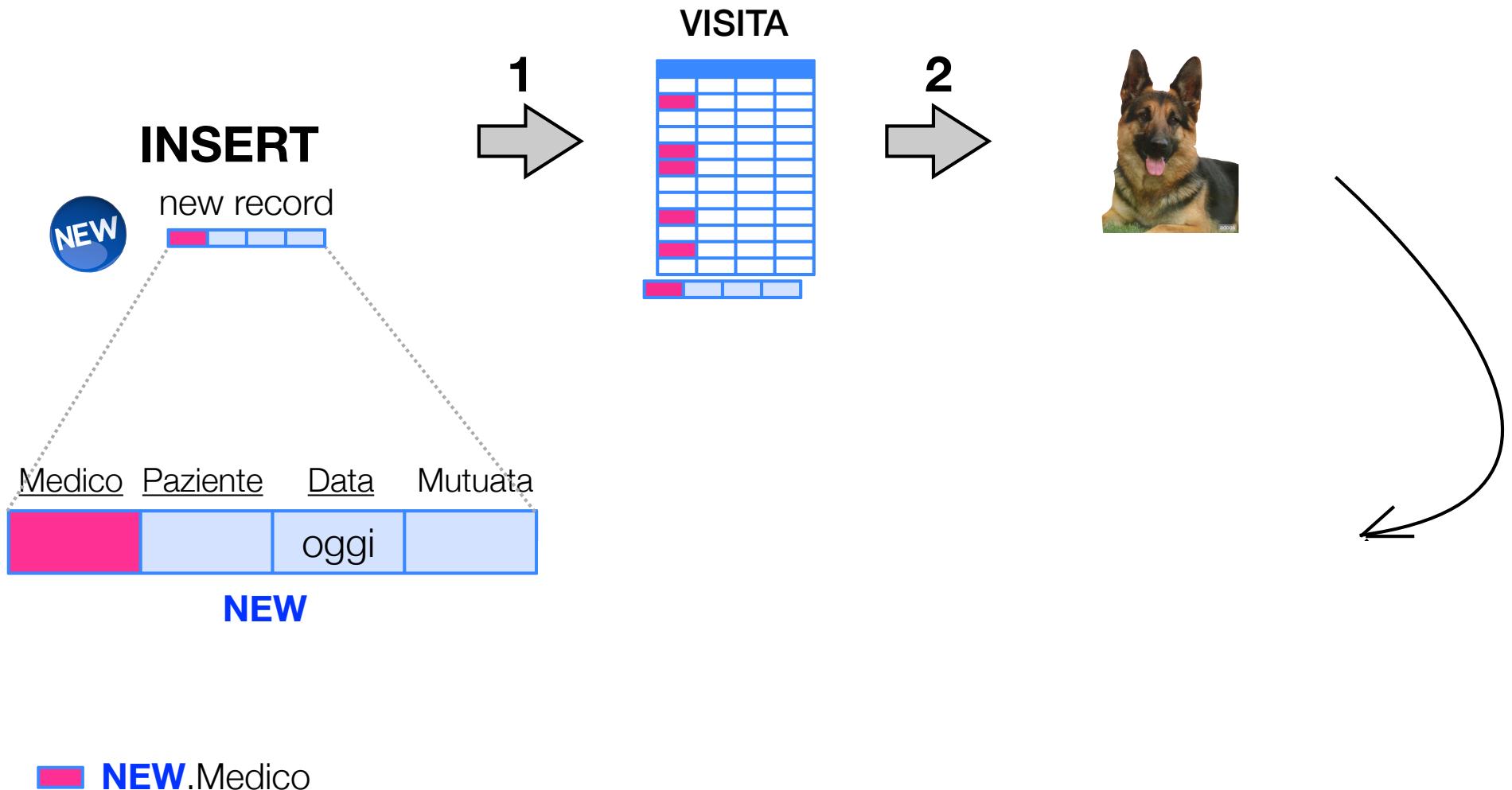
8      DECLARE non_mutuate_mese INTEGER DEFAULT 0;

9      SET non_mutuate_mese =
10         (
11             SELECT COUNT(*)
12               FROM Visita V
13              WHERE V.Medico = NEW.Medico
14                AND MONTH(V.Data) = MONTH(CURRENT_DATE)
15                AND YEAR(V.Data) = YEAR(CURRENT_DATE)
16                AND V.Mutuata = 1
17         );
18
19      IF non_mutuate_mese = 10 THEN
20          SIGNAL SQLSTATE '45000'
21          SET MESSAGE_TEXT = 'Visita non inseribile';
22      END IF;

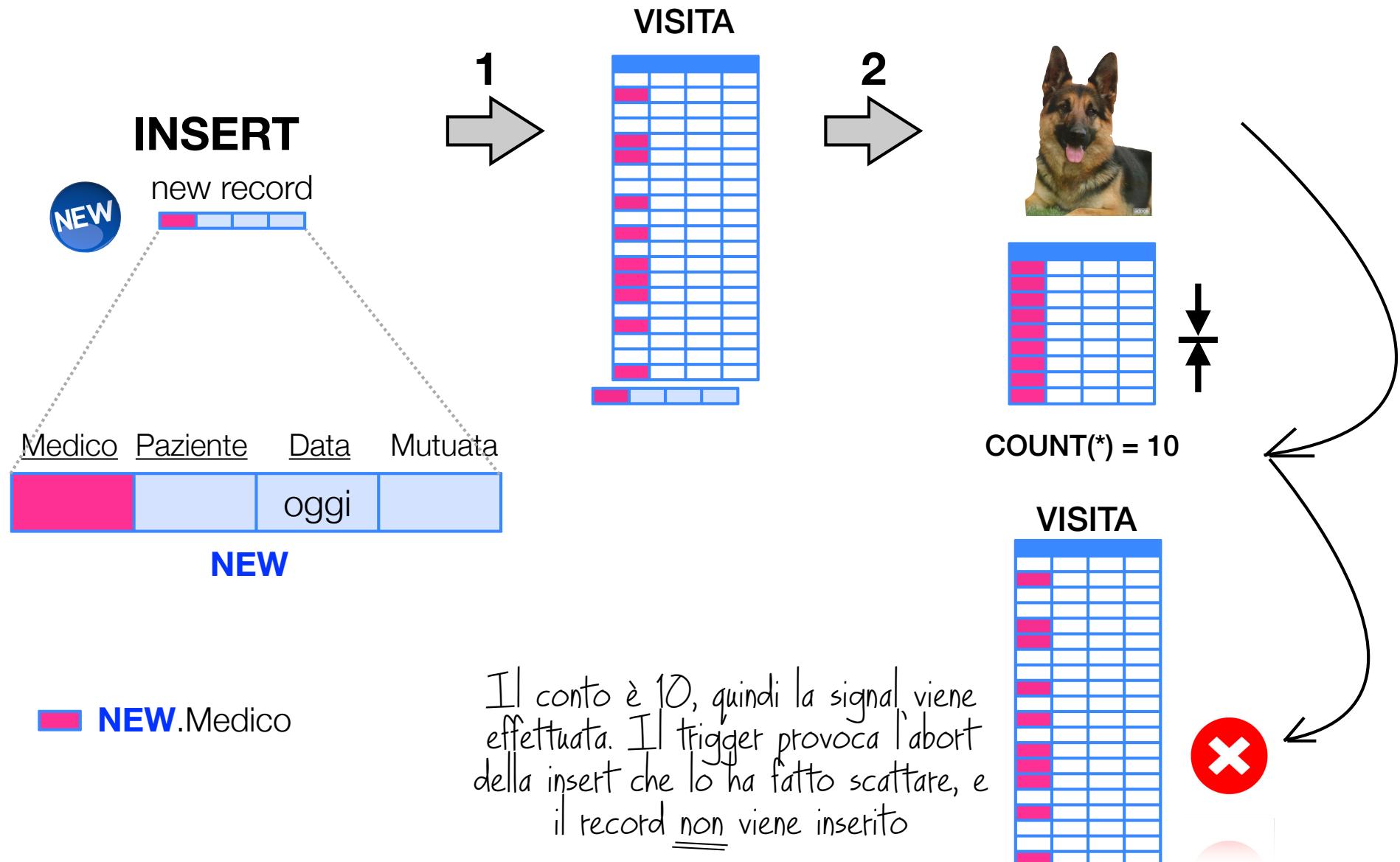
23
24
25  END $$
26  DELIMITER ;
```

1 cancella *blocca_non_mutuate* se esiste
2
3
4 crea il trigger *blocca_non_mutuate*
5 il quale, **prima** di ogni inserimento in Visita
6 per tutte le row (visite) che si inseriscono
7 faccia ciò che segue
8
9
10 metti in *non_mutuate_mese*
11 il conto
12 delle visite
13 del medico di cui si sta inserendo la visita
14 fatte in questo mese,
15 di questo anno,
16 mutuate
17
18 se le visite sono già 10, allora
19 solleva un errore che impedirà l'inserimento
20 stampando 'Visita non inseribile'
21
22
23
24
25
26

Trigger di tipo before: successo



Trigger di tipo before: blocco della insert



Esempio di business rule

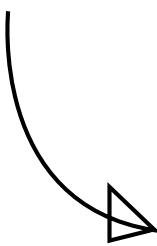
Ogni mese, le visite non mutuate di un medico non devono superare quelle mutuate.

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

Gestione di una ridondanza

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

lo si può trovare nella tabella Visita.



Se un paziente viene visitato
la data dell'ultima visita cambia!



Gestione di ridondanze



Sono attributi il cui valore è **ricavabile** da altri attributi di tabelle del database

ridondante significa "in più", cioè se ne può anche fare a meno, non si perde informazione

Tabella Paziente con ridondanza

<u>CodFiscale</u>	Cognome	Nome	Ultima Visita
BVEMDL68	Bove	Maddalena	2013-04-15
LPREDR75	Lepre	Edoardo	2012-11-07
CPRRNZ82	Capra	Renzo	2013-02-25
FRNLT A74	Faraona	Elettra	2013-03-30

Esempio: evento, condizione, azione

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



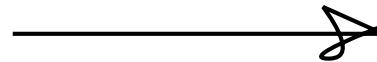
Evento



Insertimento di una nuova row
nella tabella Visita



Condizione



Non c'è



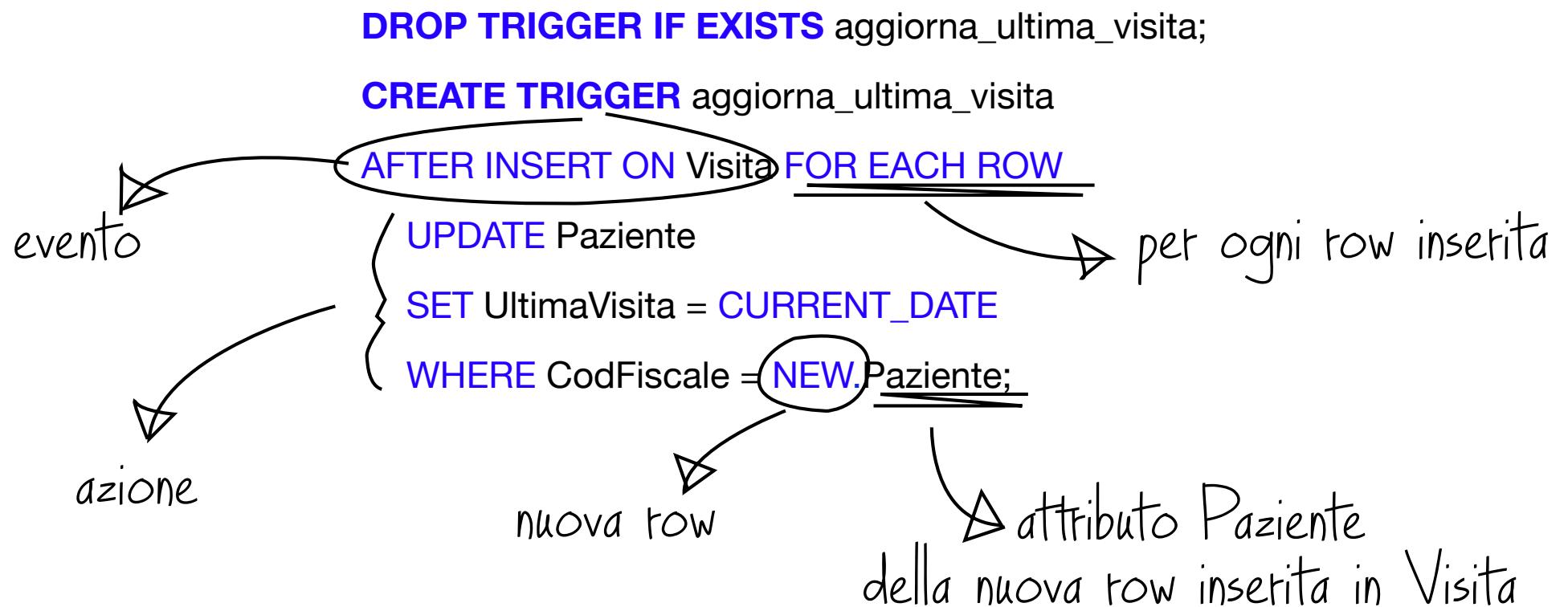
Azione



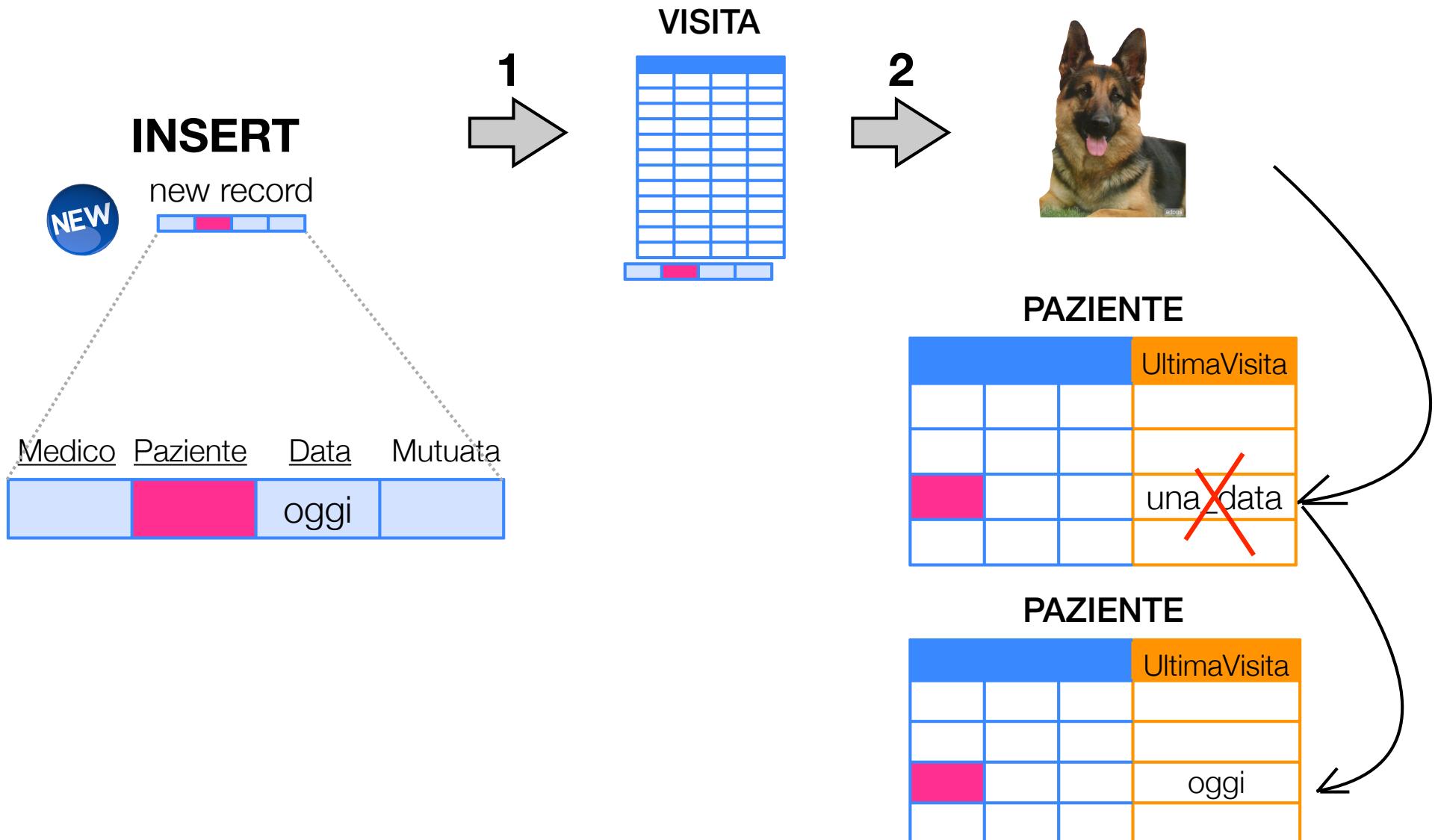
Aggiornare l'attributo UltimaVisita nella
tabella Paziente con la data corrente

Trigger in MySQL

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



Trigger di tipo after



Esempio

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

- ticket pari a euro 36.15 se reddito fra euro 0 ed euro 15,000
- ticket pari a euro 45.25 se reddito fra euro 15,000 ed euro 25,000
- ticket pari a 50.00 euro se reddito oltre 25,000 euro.

Se la visita non è mutuata, inserire NULL.

<u>Medico</u>	<u>Paziente</u>	<u>Data</u>	<u>Mutuata</u>	<u>Ticket</u>
AMRADV44	NTRLRL52	2013-05-14	TRUE	36.15
CLSCLL49	BVEMDL64	2013-05-18	FALSE	NULL
RSSMRO56	RCCEGS32	2013-05-18	FALSE	NULL
AMRADV44	MLLNTA68	2013-05-19	TRUE	45.25

Evento, condizione, azione

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 euro 0 ed euro 15,000
- ticket pari a euro 45.25 se reddito fra euro 15,000 ed euro 25,000
- ticket pari a 50.00 euro se reddito oltre 25,000 euro.

Se la visita non è mutuata, inserire NULL.

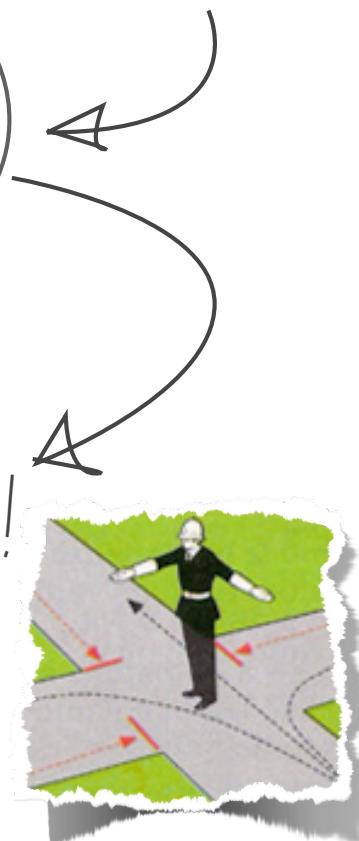


evento



azione

Occorre un controllo di flusso!



Dubbio amletico...

EVENTO → ogni volta che viene inserita una nuova visita



Evento

BEFORE INSERT ON Visita FOR EACH ROW

“prima di inserire una o più tuple in VISITA, per ognuna di esse...”

Calcolo del reddito annuo

	<u>Medico</u>	<u>Paziente</u>	<u>Data</u>	<u>Mutuata</u>	Ticket
NEW	018	NTRLRL64	2013-05-24	False	

```
SET @reddito_annuo = ( SELECT Reddito*12  
FROM Paziente  
WHERE CodFiscale = NEW.Paziente);
```



La variabile @reddito_annuo contiene il reddito annuo del paziente del quale si deve inserire una visita.

Azione

[...] se essa è mutuata, impostare l'attributo *Ticket* in base alle fasce di reddito annue

- ticket pari a euro 36.15 se reddito fra euro 0 ed euro 15,000
- ticket pari a euro 45.25 se reddito fra euro 15,000 ed euro 25,000
- ticket pari a 50.00 euro se reddito oltre 25,000 euro.

Se la visita non è mutuata, inserire NULL.

```
IF NEW.Mutuata IS TRUE THEN
    IF @reddito_annuo BETWEEN 0 AND 14999 THEN
        SET NEW.Ticket = 36.15;
    ELSEIF @reddito_annuo BETWEEN 15000 AND 25000 THEN
        SET NEW.Ticket = 45.25;
    ELSE
        SET NEW.Ticket = 50.00;
    END IF;
ELSE
    SET NEW.Ticket = NULL;
END IF;
```

Trigger

```
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 ;
```

Event



Event

Sono stored programs eseguiti in base a **condizioni dettate dal tempo**



possono anche essere ricorrenti



Esempio

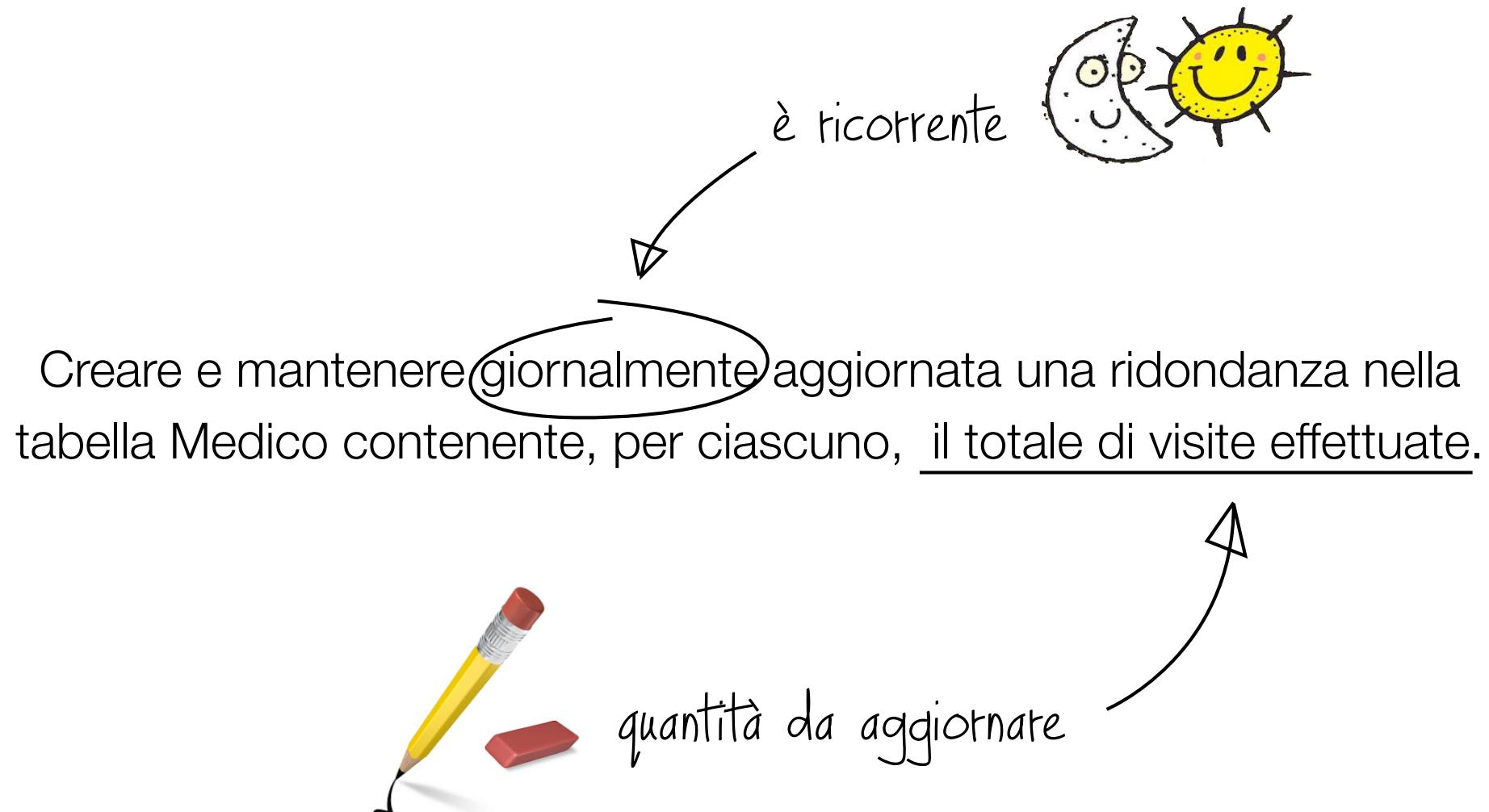


Tabella Medico con ridondanza

MEDICO

Matricola	Cognome	Nome	Specializzazione	Parcella	Citta	TotaleVisite
18339	Verdi	Paolo	Ortopedia	150	Pisa	135
35512	Rossi	Marta	Ortopedia	120	Pisa	62
16220	Gialli	Rita	Cardiologia	135	Roma	97
29858	Neri	Rino	Dermatologia	110	Siena	211

è ridondante perché si può
ottenere dai dati!!!

Events do it better!



Eseguito ogni giorno, a partire dalla creazione.

CREATE EVENT AggiornaTotaliVisite

ON SCHEDULE **EVERY 1 DAY**

DO

(aggiorna la tabella Medico
incrementando il numero totale di visite
ai soli medici che hanno visitato in data odierna
);

Azione

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

1. Trovate i medici che hanno visitato oggi
2. Incrementare TotaleVisite ai soli medici in #1

Soluzione corretta

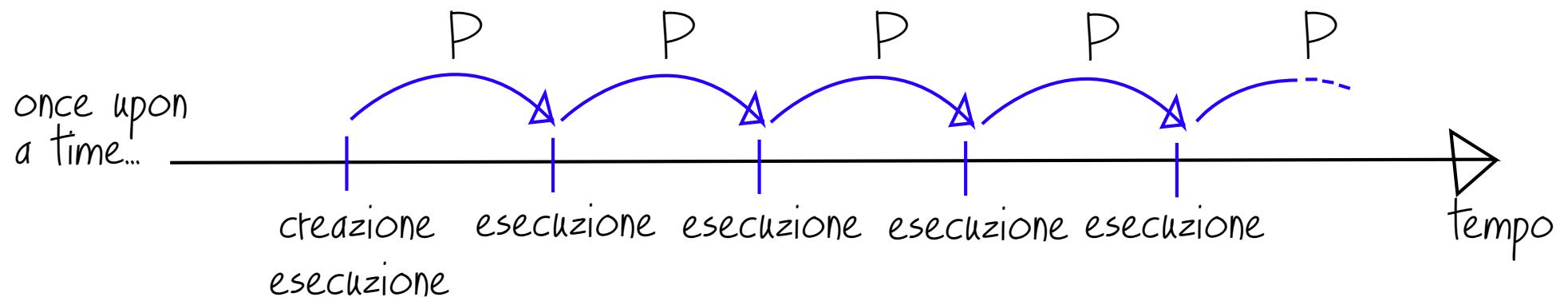
```
CREATE EVENT AggiornaTotaliVisite
  ON SCHEDULE EVERY 1 DAY          periodicità
  STARTS '2020-04-24 23:55:00'    prima esecuzione
DO
  UPDATE Medico
  SET TotaleVisite = TotaleVisite + (SELECT COUNT(*)
                                      FROM Visita V
                                      WHERE V.Medico = Matricola AND
                                            V.Data = CURRENT_DATE);
```

Sintassi MySQL

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



periodicità P con cui avviene l'esecuzione



Varianti

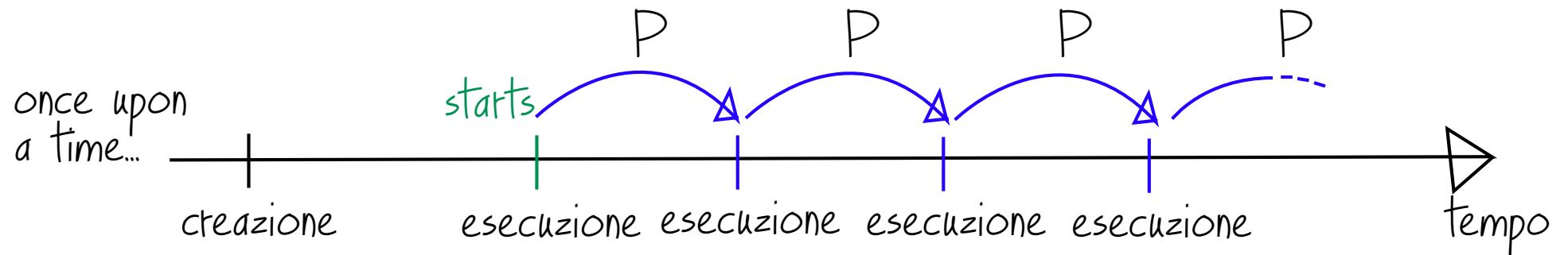
È possibile stabilire sia un istante di **inizio** che un istante di **fine**



se non si specificano, il recurring event
scatta immediatamente, e poi segue la periodicità

Varianti starts e starts-ends

ON SCHEDULE EVERY numero DAY | MONTH | YEAR | SECOND | MINUTE | HOUR
STARTS 'data_ora'



Event a singolo scatto

ON SCHEDULE AT 'data_ora' + INTERVAL numero DAY | MONTH | YEAR |
SECOND | MINUTE | HOUR

[ON COMPLETION PRESERVE]

il risultato di questa espressione è
l'istante di esecuzione

once upon
a time...

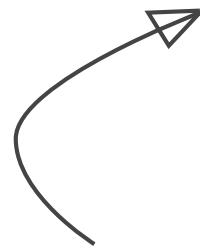
creazione

esecuzione

tempo

Attivazione dello scheduler

```
SET GLOBAL event_scheduler = ON;
```



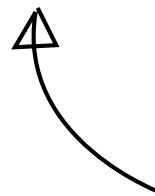
Settando questa variabile di sistema,
parte la schedulazione degli eventi

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

operazioni di normale
utilizzo del database
(inserimento visite, terapie...)

↓
INSERT →
UPDATE →
DELETE →

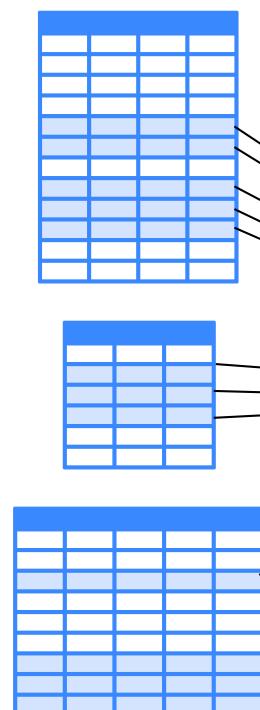
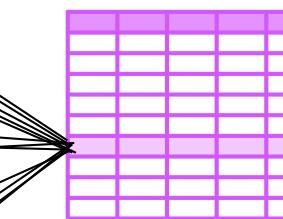


tabelle
(raw data)

un record della materialized view
deriva dall'aggregazione di informazioni
provenienti dai record di una o più
tabelle del database (quelle blu)



materialized view

←
SELECT
(OLAP e data
analysis)

particolari select molto onerose che non
possono essere eseguite sulle tabelle raw
perché appesantirebbero troppo il carico

Utilizzi

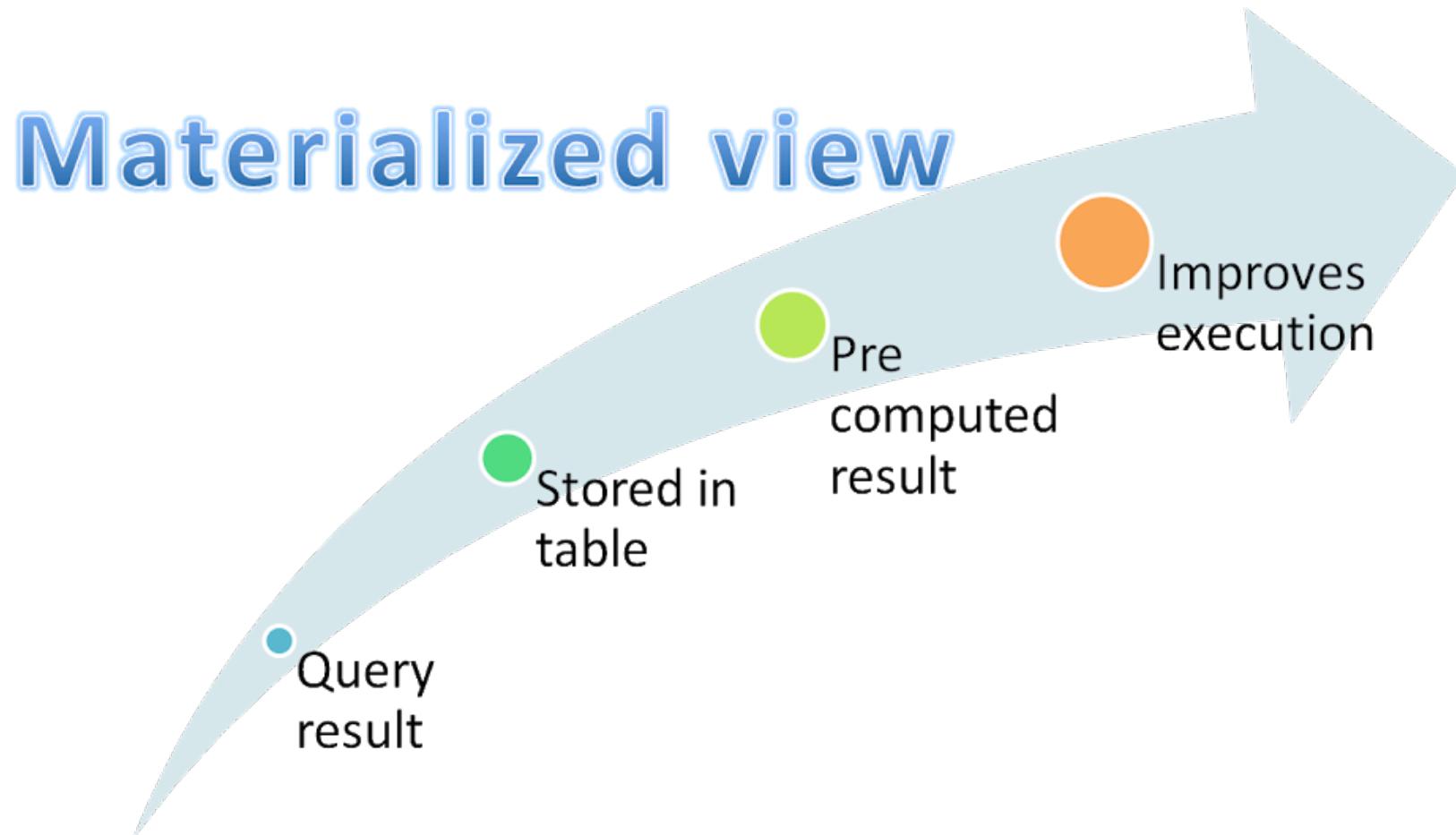
eventualmente anche non aggiornatissima

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”...

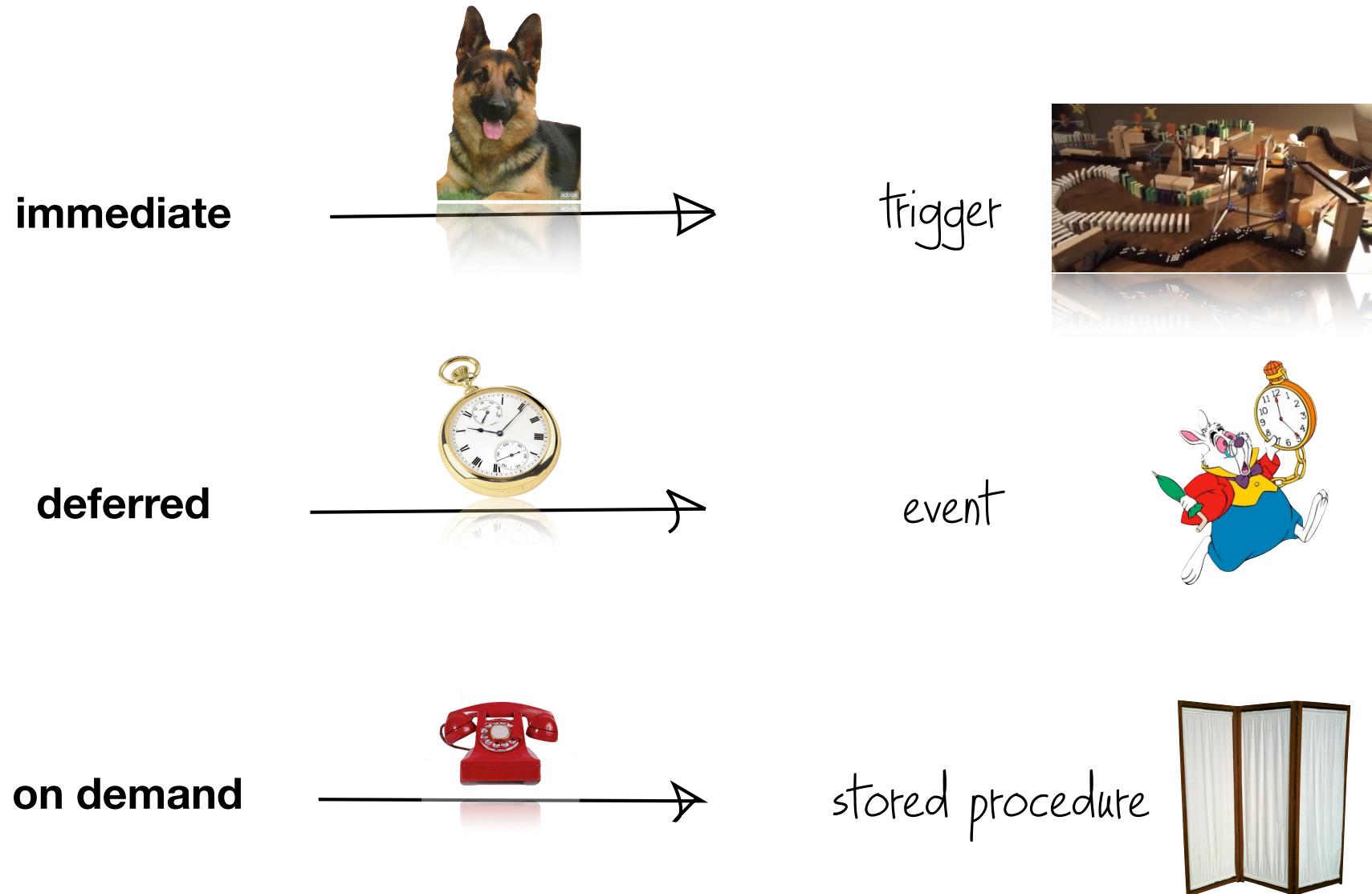
- risultati di query complesse **già pronti**
- il risultato **senza ricalcolarlo**
- **indici ad hoc** per velocizzare l'accesso

a patto di accettare...

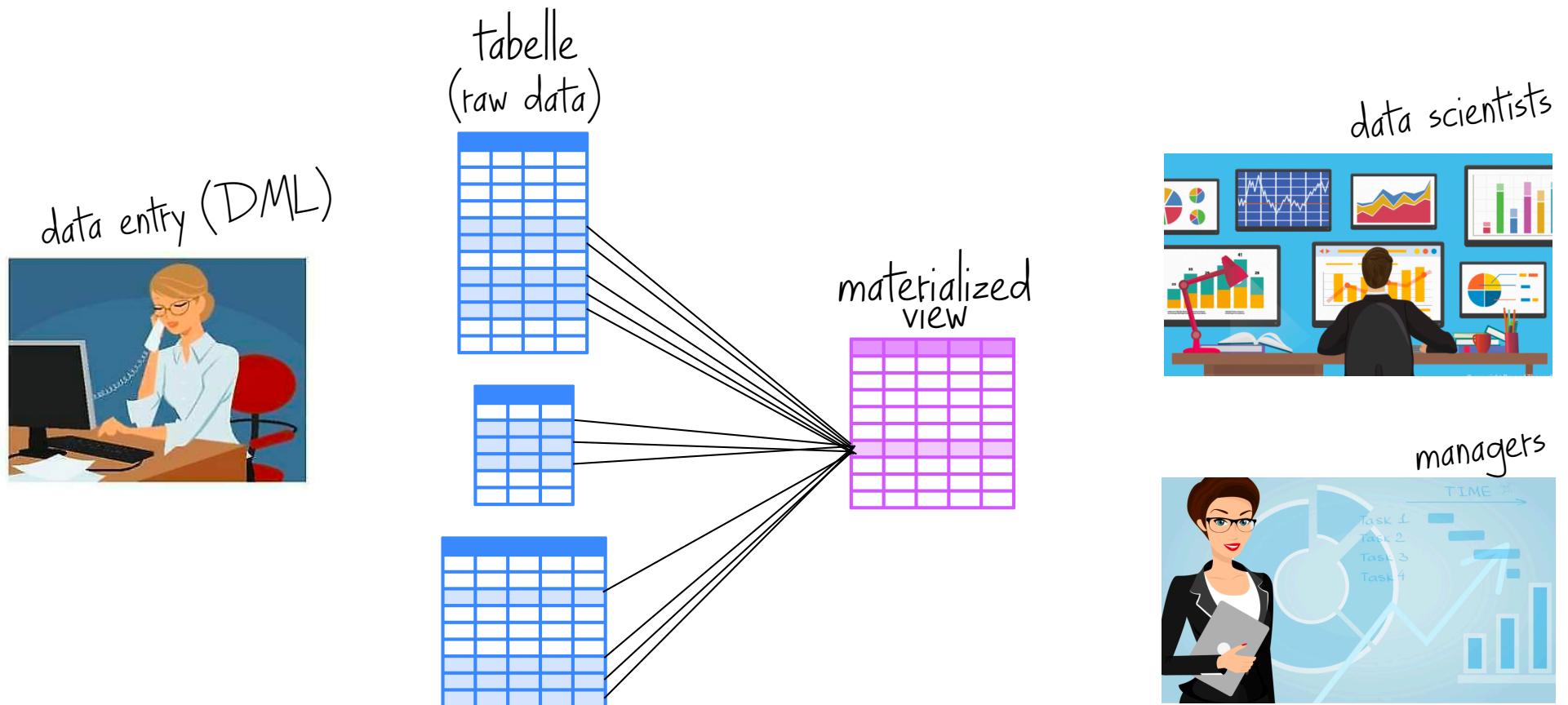
- possibilità di **informazioni non aggiornate**
- maggiore **occupazione di memoria**
- maggiore **carico sul server**



Politiche di refresh



Raw data vs. materialized view



Build

```
-- 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;
```

Esempio

PAZIENTE

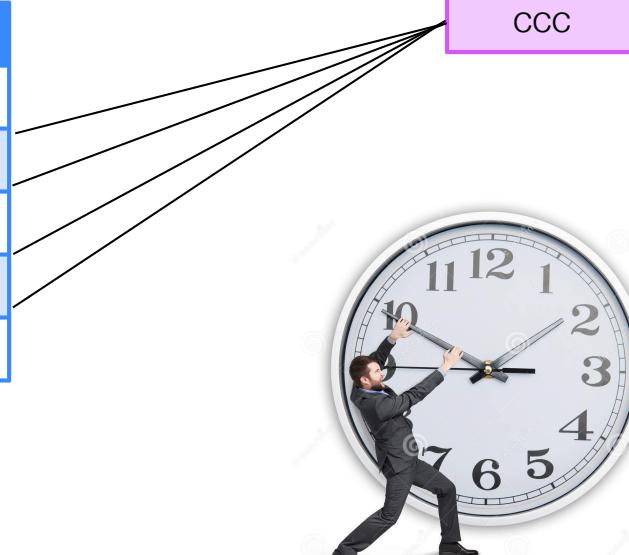
CodFiscale	Cognome	Nome	...
aaa	Gatto	Marco	
bbb	Cane	Giulia	
ccc	Topo	Aldo	

VISITA

Paziente	Medico	Data	Mutuata
aaa	1	23/4	0
ccc	3	23/4	1
aaa	3	29/4	0
ccc	2	2/5	1
bbb	4	2/5	0

MATERIALIZED VIEW

Paziente	NumVisite	UltimaVisita
aaa	2	29/4
bbb	1	2/5
ccc	2	2/5



Se tutto sta fermo,
la materialized view è
in sync con i raw data.

Ma però...

PAZIENTE

CodFiscale	Cognome	Nome	...
aaa	Gatto	Marco	
bbb	Cane	Giulia	
ccc	Topo	Aldo	

VISITA

Paziente	Medico	Data	Mutuata
aaa	1	23/4	0
ccc	3	23/4	1
aaa	3	29/4	0
ccc	2	2/5	1
bbb	4	2/5	0

Insert



ccc	1	8/5	0
-----	---	-----	---

MATERIALIZED VIEW

Paziente	NumVisite	UltimaVisita
aaa	2	29/4
bbb	1	2/5
ccc	2	2/5

E allora? Si butta via?

PAZIENTE

CodFiscale	Cognome	Nome	...
aaa	Gatto	Marco	
bbb	Cane	Giulia	
ccc	Topo	Aldo	

VISITA

Paziente	Medico	Data	Mutuata
aaa	1	23/4	0
ccc	3	23/4	1
aaa	3	29/4	0
ccc	2	2/5	1
bbb	4	2/5	0
ccc	1	8/5	0



MATERIALIZED VIEW

Paziente	NumVisite	UltimaVisita
aaa	2	29/4
bbb	1	2/5
ccc	2	2/5



Attenzione, attenzione,
materialized view non più
aggiornata!!!

Time goes on...

PAZIENTE

CodFiscale	Cognome	Nome	...
aaa	Gatto	Marco	
bbb	Cane	Giulia	
ccc	Topo	Aldo	
ddd	Rospo	Rita	
eee	Volpe	Ugo	
...

NEW

NEW

NEW

VISITA

Paziente	Medico	Data	Mutuata
aaa	1	23/4	0
ccc	3	23/4	1
aaa	3	29/4	0
ccc	2	2/5	1
bbb	4	2/5	0
...
...
...
...
...

NEW

NEW

NEW

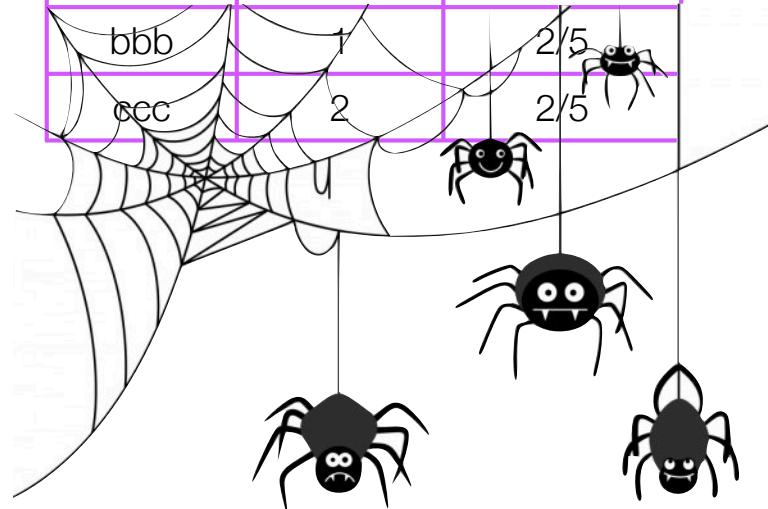
NEW

NEW

OUTDATED

MATERIALIZED VIEW

Paziente	NumVisite	UltimaVisita
aaa	2	29/4
bbb	1	2/5
ccc	2	2/5



Refresh



Refresh

Una materialized view può essere aggiornata secondo due modalità:

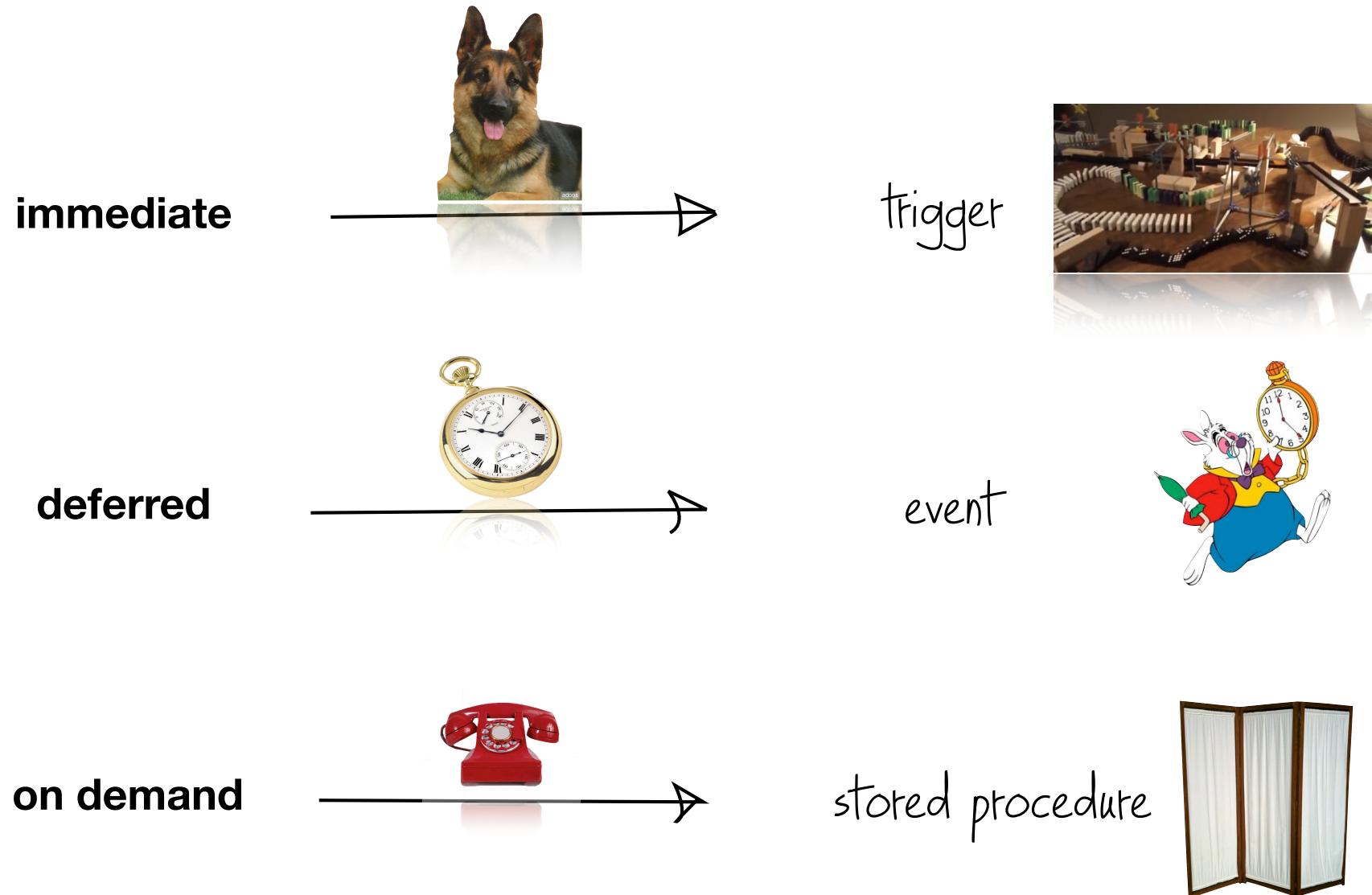
full refresh oppure **incremental refresh**



aggiorna la materialized view
"from scratch"

aggiornamento della parte
non più aggiornata
(totale o parziale)

Politiche di refresh



Immediate refresh (sync)



-- 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 $$
```

```
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 $$
```

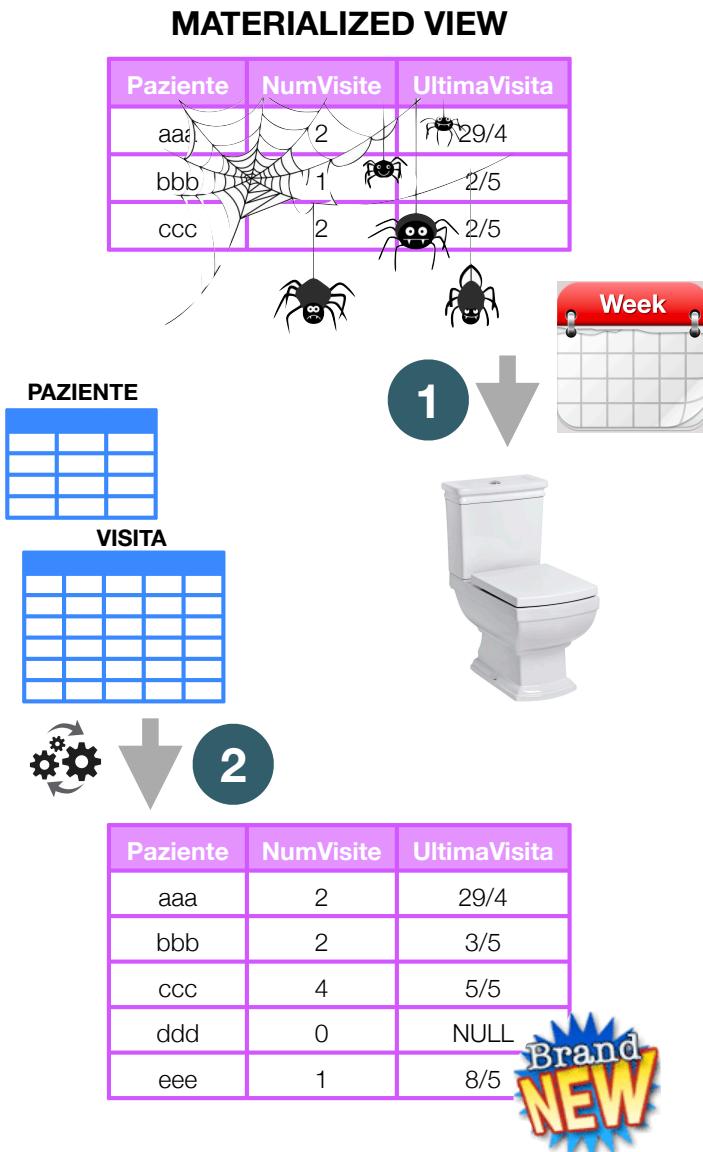
On demand refresh (full)



-- on demand refresh (full)

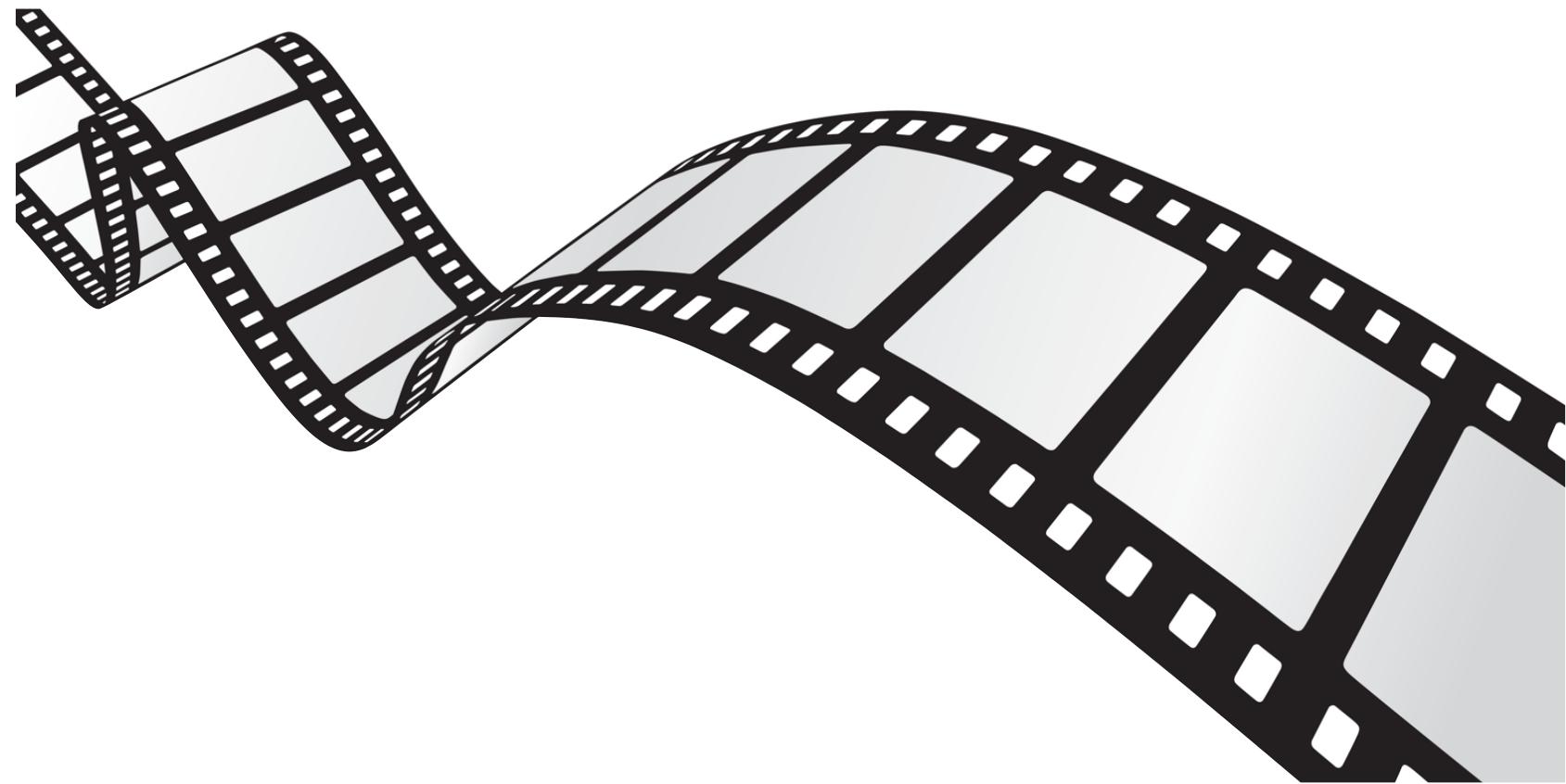
```
DROP PROCEDURE IF EXISTS on_demand_refresh_mv $$  
CREATE PROCEDURE on_demand_refresh_mv()  
BEGIN  
    1 TRUNCATE MATERIALIZED_VIEW;  
    2 INSERT INTO MATERIALIZED_VIEW  
        SELECT P.CodFiscale,  
               IF(V.Paziente IS NULL, 0, COUNT(*)),  
               IF(V.Paziente IS NULL,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

Deferred refresh (full)



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

Incremental refresh



Incremental refresh

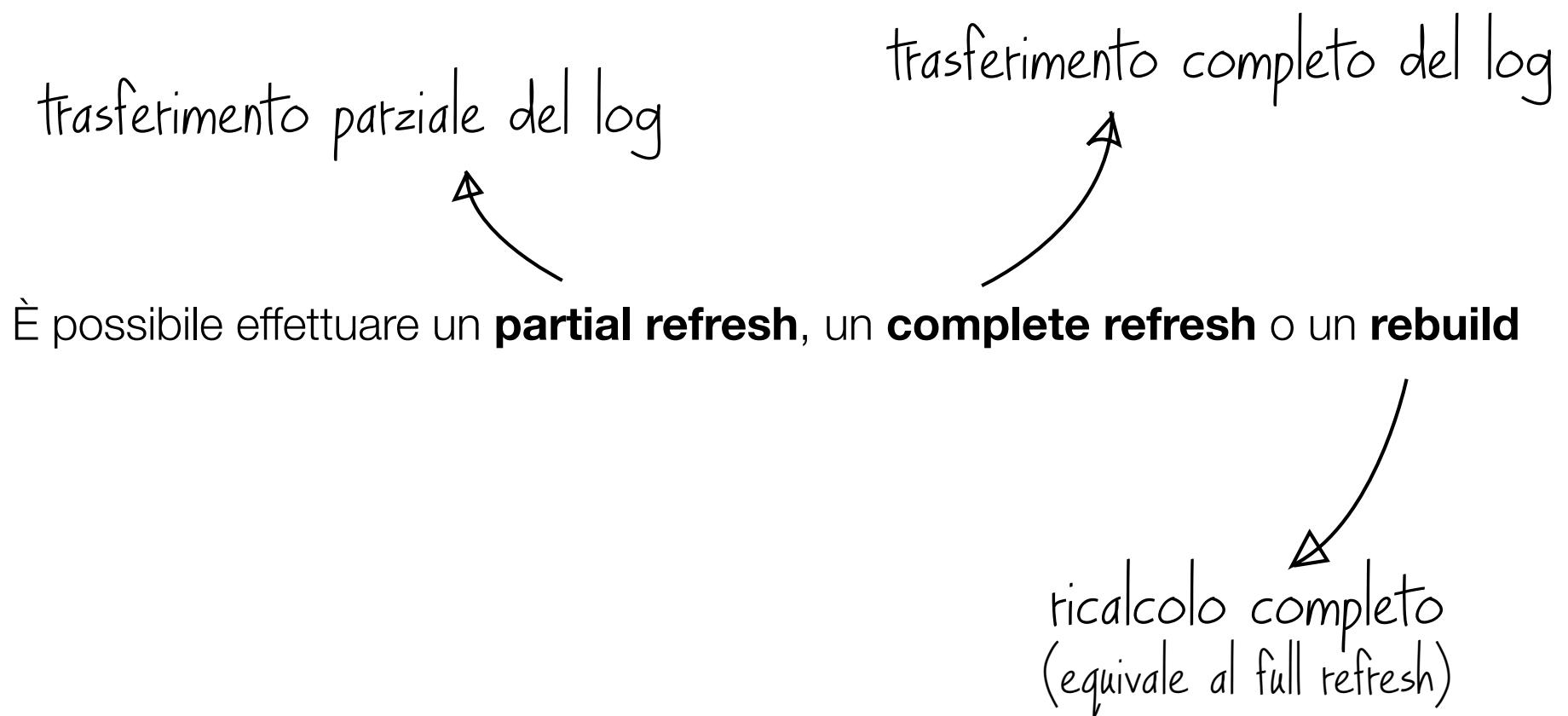
trade-off aggiornamento/prestazioni

Funzionalità che permette di avere i dati di una materialized view aggiornati
fino a un certo istante di tempo (del passato)

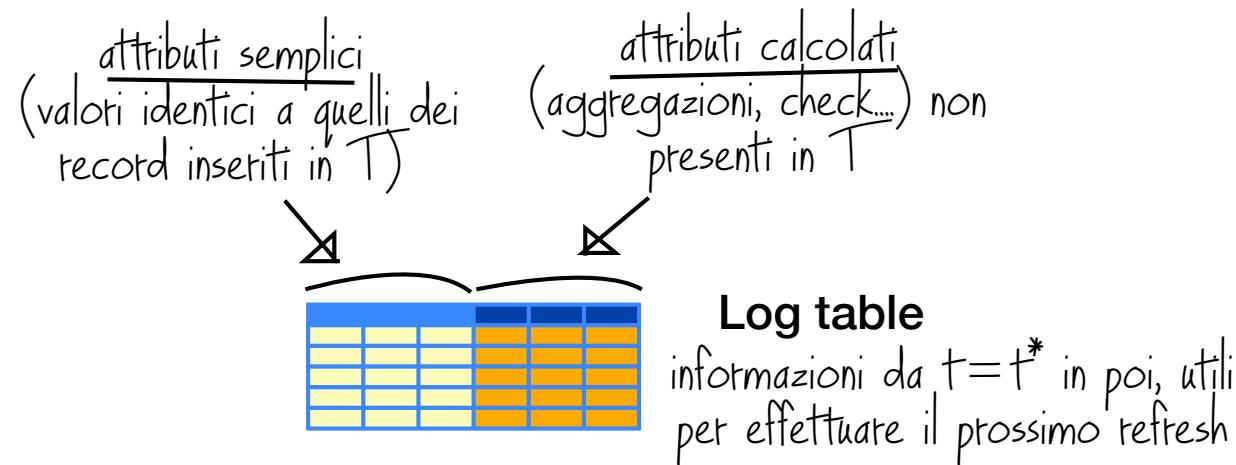


fra un refresh e il successivo,
le modifiche alle tabelle raw sono
mantenute in un log

Modalità di incremental refresh



Perché l'incremental refresh è migliore?

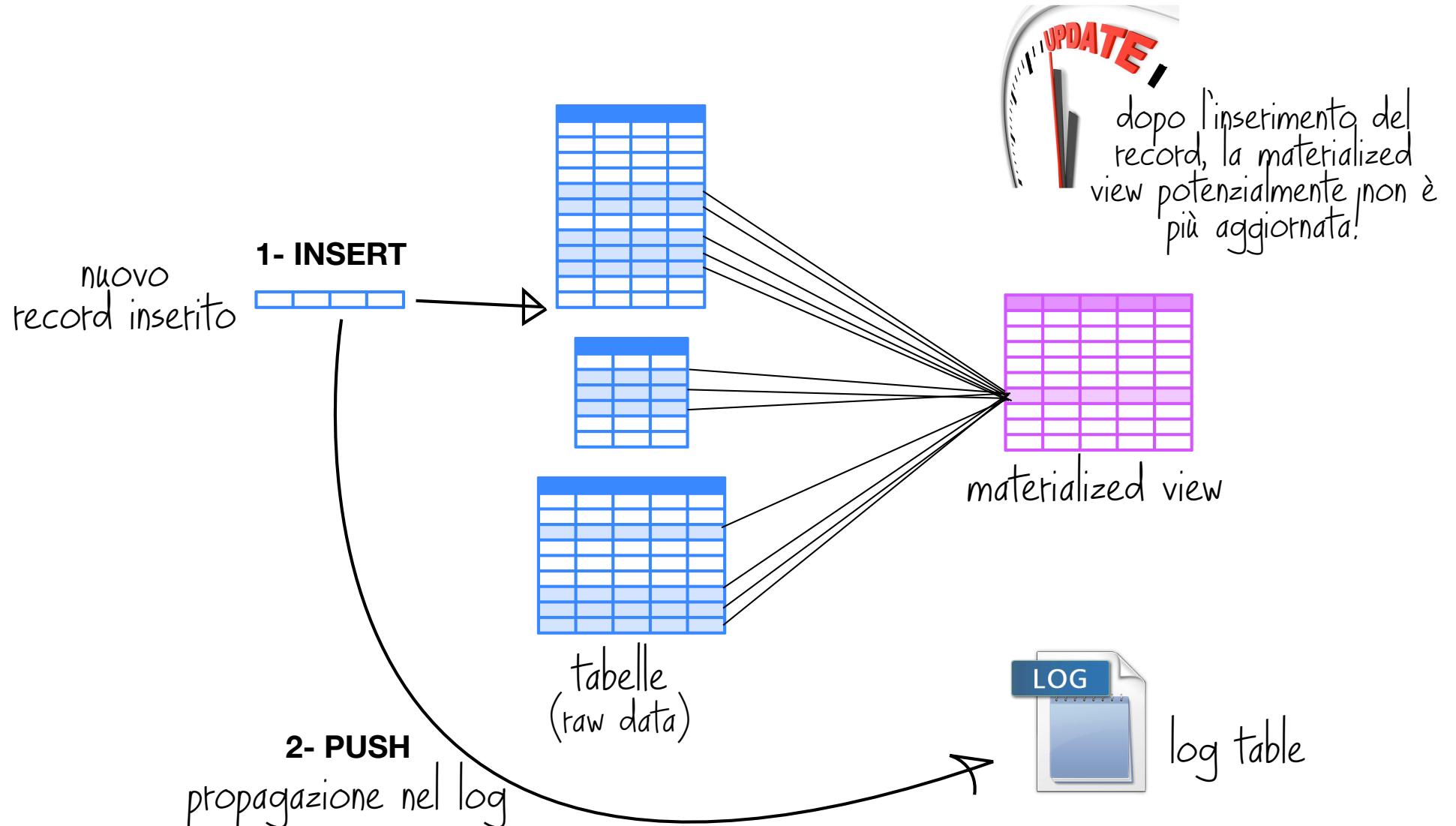


Tramite l'incremental refresh è possibile aggiornare la materialized view sfruttando solo i **dati che essa già contiene** e la **log table**

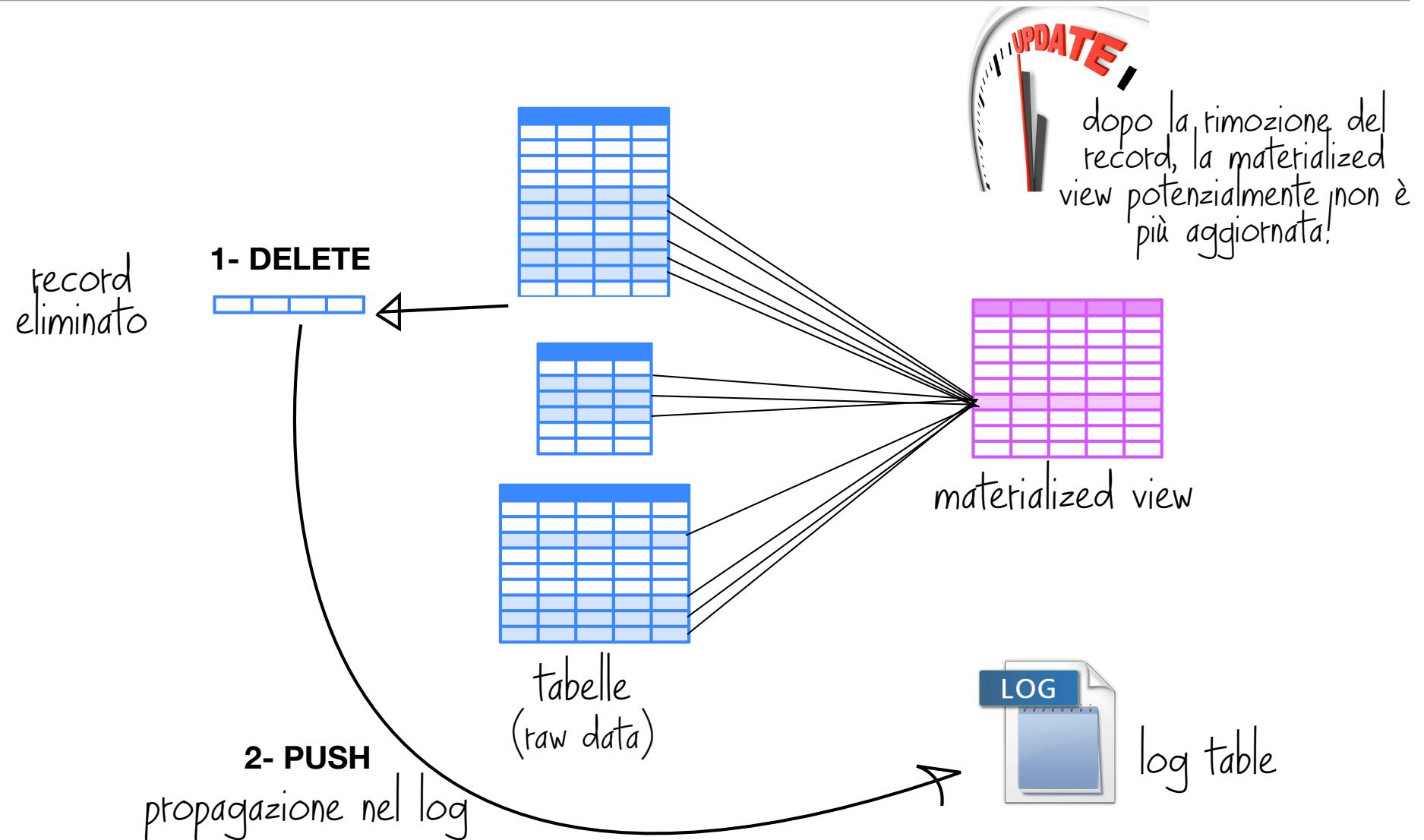
è molto più performante perché la log table ha molti meno record rispetto alle tabelle raw (quelle blu)



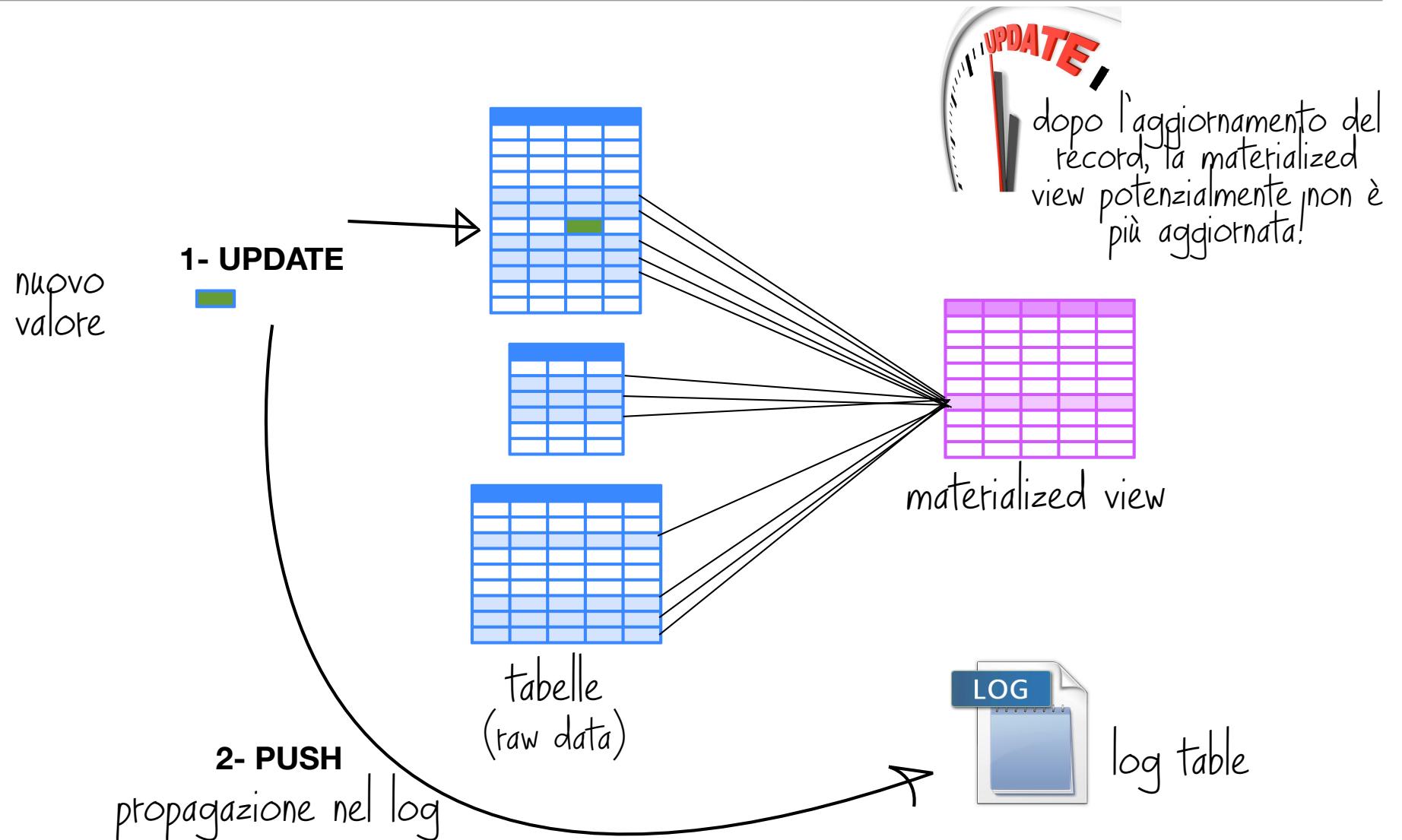
Come funziona: inserimento



Come funziona: cancellazione



Come funziona: aggiornamento

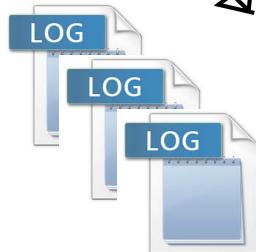


Cosa salvare nella log table?



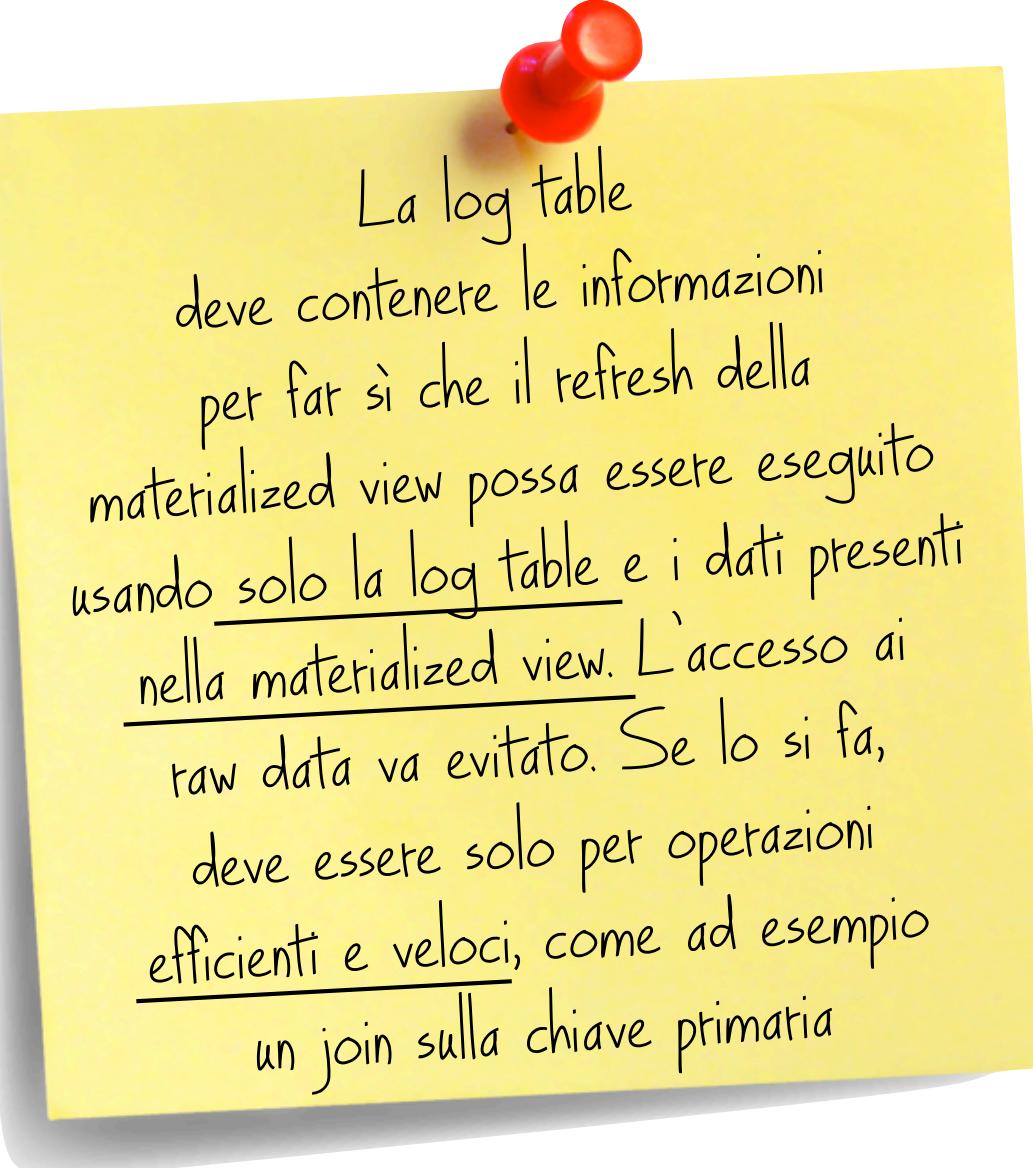
significa che quando si decide di effettuare il refresh, i dati nella log table devono essere sufficienti per effettuarlo

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

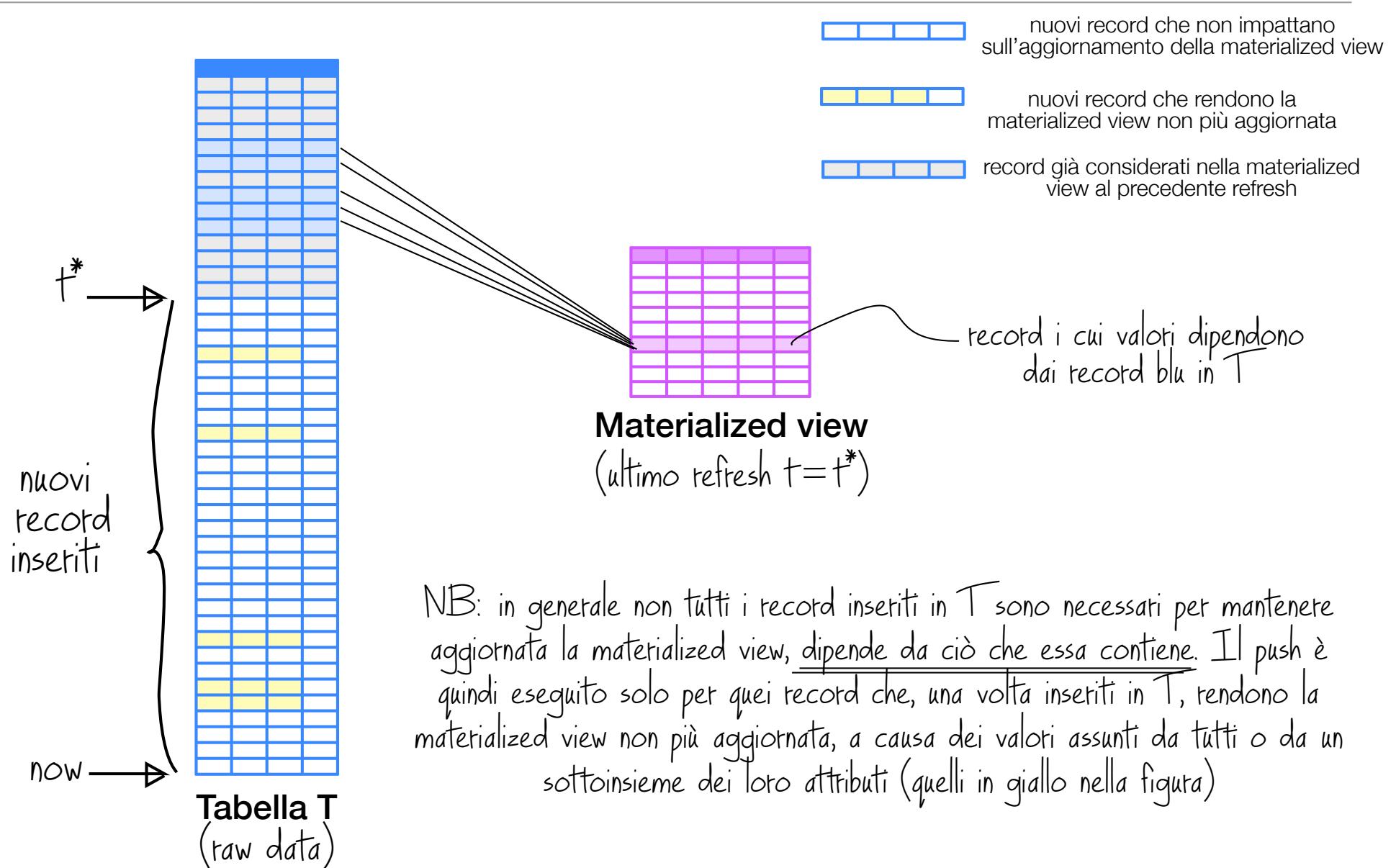


se la materialized view si basa su più tabelle, potrebbero essere utili più log table

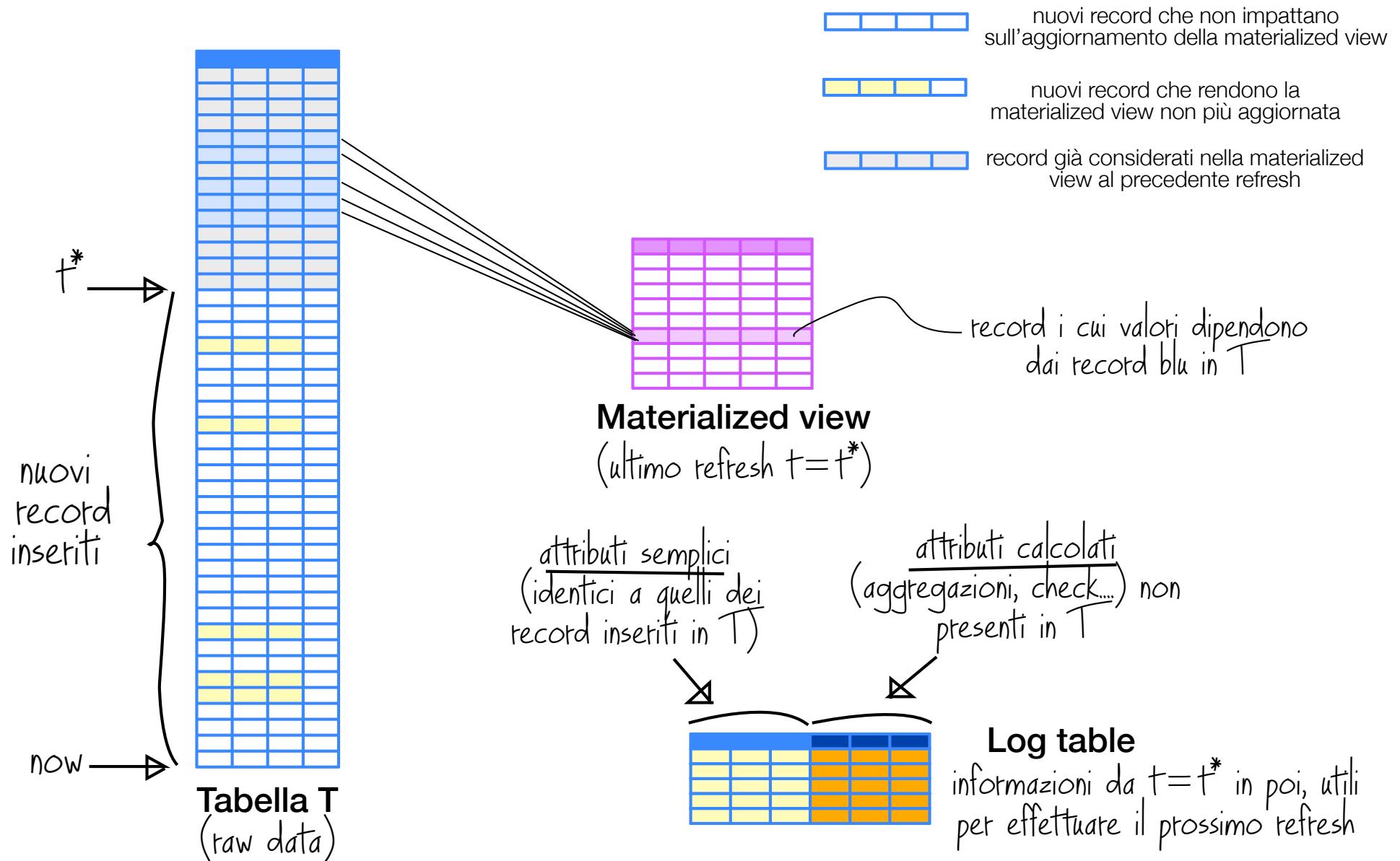
Ricorda



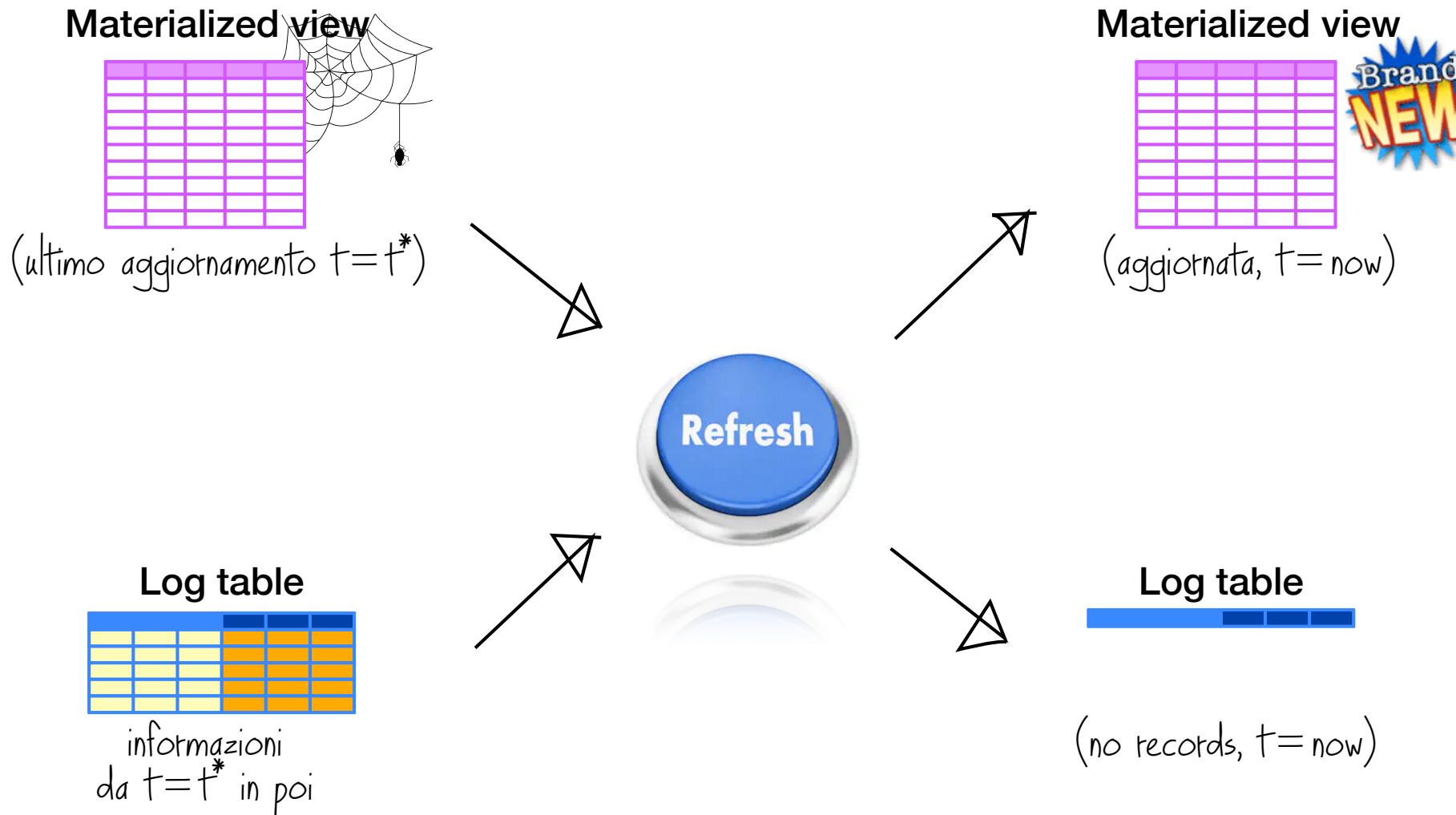
Fra un refresh e l'altro (consideriamo solo le insert)



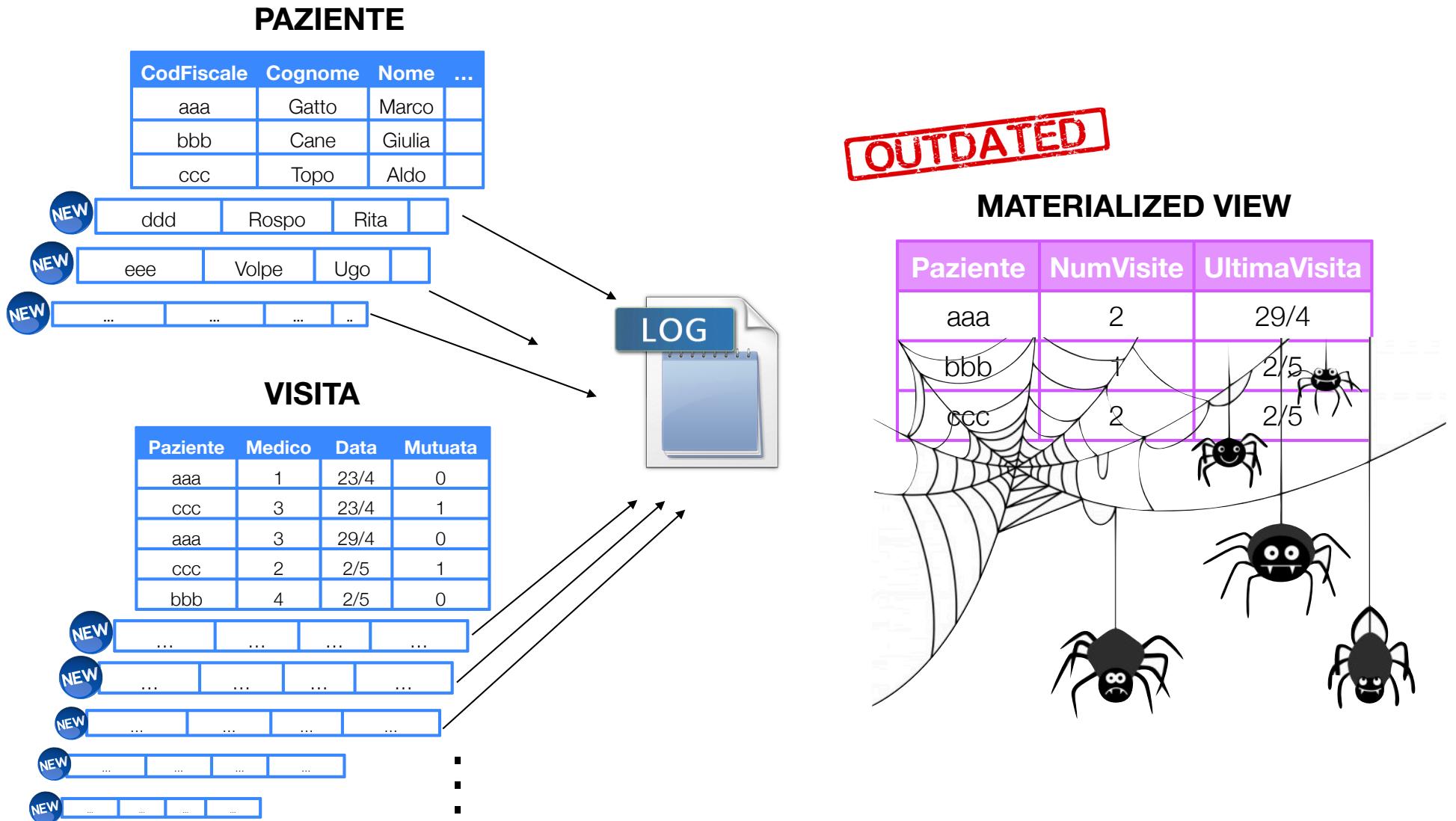
Fra un refresh e l'altro: cosa va nel log?



Let's refresh! [complete]



Stesso esempio di prima, ma incremental



Log table

MATERIALIZED VIEW

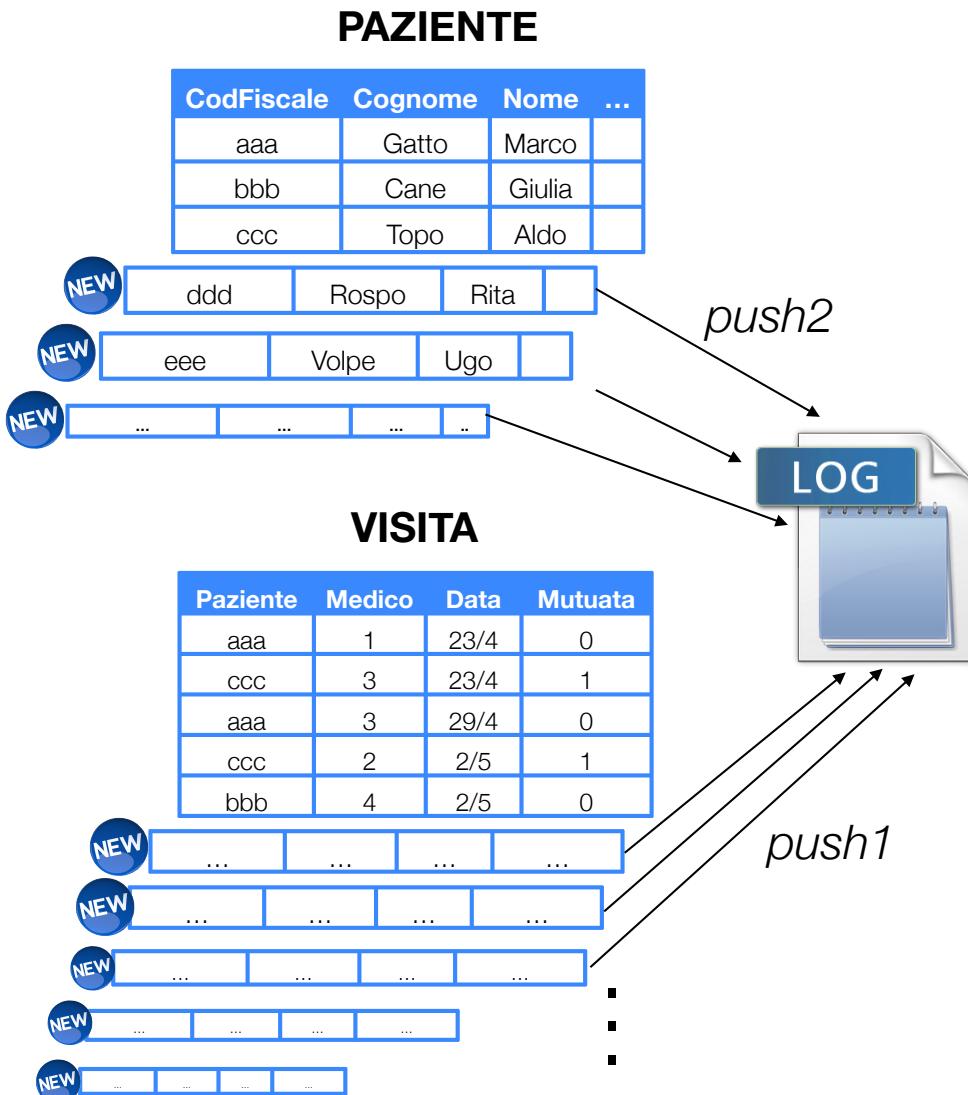
Paziente	NumVisite	UltimaVisita
aaa	2	29/4
bbb	1	2/5
ccc	2	2/5

LOG_TABLE

Paziente	DataVisita
aaa	3/5
bbb	4/5
ddd	4/5
aaa	8/5

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

Trigger di push



```
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 $$
```

```
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 $$
```

Partial refresh

```
DROP PROCEDURE IF EXISTS on_demand_refresh_mv $$  
CREATE PROCEDURE on_demand_refresh_mv(_up_to DATE)  
BEGIN  
  
    -- 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,  
            MAX(LT.DataVisita) AS NuovaUltima  
        FROM LOG_TABLE LT  
        WHERE LT.DataVisita <= _up_to OR LT.DataVisita IS NULL  
        GROUP BY LT.Paziente  
    )
```

LOG_TABLE

Paziente	DataVisita
aaa	3/5
bbb	4/5
ddd	4/5
aaa	6/5
eee	NULL



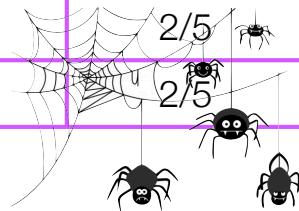
CTE aggregated_log

Paziente	NuoveVisite	NuovaUltima
aaa	2	6/5
bbb	1	4/5
ddd	1	4/5
eee	0	NULL

Cosa si vuole fare a questo punto

MATERIALIZED VIEW (MV)

Paziente	NumVisite	UltimaVisita
aaa	2	29/4
bbb	1	2/5
ccc	2	2/5



Paziente

CTE aggregated_log (AL)

Paziente	NuoveVisite	NuovaUltima
aaa	2	6/5
bbb	1	4/5
ddd	1	4/5
eee	0	NULL

Result set generato dal join esterno

MV.Paziente	MV.NumVisite	MV.UltimaVisita	AL.Paziente	AL.NuoveVisite	AL.NuovaUltima
aaa	2	29/4	aaa	2	6/5
bbb	1	2/5	bbb	1	4/5
NULL	NULL	NULL	ddd	1	4/5
NULL	NULL	NULL	eee	0	NULL

Partial refresh

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

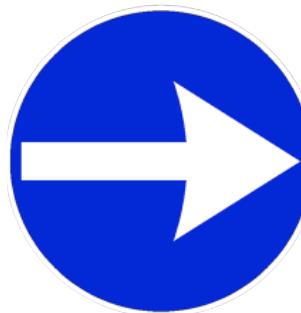
```
REPLACE INTO MATERIALIZED_VIEW
-- segue immediatamente la CTE aggregated_log
SELECT MV.Paziente,
       IF(MV.Paziente IS NULL,
          AL.NuoveVisite,
          MV.NumVisite + IF(AL.NuovaUltima IS NULL, 0, AL.NuoveVisite)
        ),
       IF(MV.Paziente IS NULL,
          IFNULL(AL.NuovaUltima, 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 OR (LT.Paziente IS NULL
                                     AND LT.Paziente IN (SELECT Paziente
                                                          FROM MATERIALIZED_VIEW));
END $$
```

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).