Relazione

Bruniera Alvise

Calabrigo Massimo

Università degli studi di Udine

November 28, 2021

Contents

1 Progettazione fisica

1.1 Scelta degli indici

Sono stati implementati degli indici per volocizzare l'esecuzione di alcune query per cui abbiamo ritenuto valesse la pena allocare lo spazio aggiuntivo necessario.

Abbiamo utilizzato indici secondari per la ricerca attraverso nome o cognome di pazienti o medici, e per la ricerca della terapia prolungata secondo le stime della frequenza di interrogazione delle rispettive tabelle nelle fase precedenti.

- cf_terapiaProlungata → per tutte le query che cercano dati su un paziente in generale (es. tutte le terapie del paziente x);
- terapia_appuntamento → per la query "cerca tutti gli appuntamenti di una determinata terapia x".
- codiceMedico_medicoAppuntamento → per la query "Tutti gli appuntamenti in cui il medico ha partecipato"

```
create index cf_terapiaProlungata on terapiaProlungata (cf)
;
create index cf_appuntamento on appuntamento (cf);
create index cf_programmato on programmato (cf);
create index cf_accettato on accettato (cf);
create index cf_seduta on seduta (cf);
create index terapia_appuntamento on appuntamento (
dataDiInizio,cf,tipoDiSpecializzazione);
create index codiceMedico_medicoAppuntamento on
medicoAppuntamento (codiceMedico);
create index codiceMedico_medicoSeduta on medicoSeduta (
codiceMedico);
create index nome_paziente on paziente (nome);
create index nome_medico on medico (nome);
create index cognome_paziente on paziente (cognome);
create index cognome_medico on medico (cognome);
```

Sono stati considerati anche degli indici secondari sulle tabelle programmato e accettato, ma abbiamo deciso di non metterli perchè nelle fasi precedenti di stima della quantità di queries, queste tabelle risultavano interrogate molto più raramente.

2 Alcuni Trigger e Query

2.1 Trigger

Nelle fasi precedenti (diagramma casi d'uso, frequenza e tipo operazioni) sono stati nominati diversi trigger, di seguito riportiamo l'implementazione di alcuni triggers esemplificativi.

Quando aggiungiamo un appuntamento accettato, la specializzazione richiesta dalla terapia deve essere tra le specializzazioni del medico che fa l'appuntamento

```
create or replace function controlla_specializzazione()
return trigger
language plpgsql as $$
begin
perform *
from specializzare
where codiceMedico = new.codiceMedico and new.
tipoDiSpecializzazione = tipoDiSpecializzazione;
if found then
    return new;
else
    raise exception 'Il medico inserito non ? specializzato
 per l appuntamento';
```

```
        return null;
    endif;
    end;
    $$;

    create trigger inserisci_accettato()
    before insert on medicoAppuntamento
    for each row
    execute procedure controlla_specializzazione();
```

Quando inseriamo una terapia prolungata, non ci devono essere, per quel paziente, terapie prolungate aperte con lo stesso tipo di specializzazione

```
    create or replace function
    controlla_specializzazione_terapieProlungate()
    return trigger
    language plpgsql as $$
    begin
    perform *
    from terapiaProlungata
    where new.tipoDiSpecializzazione = tipoDiSpecializzazione
    and tipoDiTerapia = 'aperta' and new.cf = cf and new.
    tipoDiTerapia = 'aperto';
    if found then
        raise exception 'Non puoi inserite 2 terapie aperte con
     la stessa specializzazione';
        return null;
    else
        return new;
    endif;
    end;
    $$;

    create trigger inserisci_terapiaProlungata()
    before insert on terapiaProlungata
    for each row
    execute procedure
    controlla_specializzazione_terapieProlungate();
```

Quando inseriamo un nuovo appuntamento nella tabella appuntamento, se è un appuntamento futuro lo inseriamo anche in nella tabella Programmato

```
    create or replace function
    check_programma_appuntamento_futuro()
    returns trigger
    language plpgsql as $$
    begin
        if ((new.data > CURRENT_DATE) or (new.data =
    CURRENT_DATE and new.ora > extract(hour from CURRENT_TIME))
```

```
    ) then
            insert into programmato (data, cf, dataDiInizio,
    tipoDiSpecializzazione, ora)
            values (new.data, new.cf, new.dataDiInizio, new.
    tipoDiSpecializzazione, new.ora);
        end if;
        return new;
    end;
    $$;

    create trigger programma_appuntamento_futuro after insert
    or update
    on appuntamento for each row
    execute procedure check_programma_appuntamento_futuro();
```

Verifichiamo che solo gli assistenti medici possano iscriversi ai corsi di aggiornamento

```
    create or replace function
    check_non_partecipa_amministrativo()
    returns trigger
    language plpgsql as $$
    begin
        perform * from membroPersonaleAusiliario
            where codicePersonale = new.codicePersonale and
    tipo = 'amministrativo';
        if found then
            raise exception 'Solo gli assistenti medici possono
     essere iscritti ai corsi di aggiornamento';
            return null;
        else
            return new;
        end if;
    end;
    $$;

    create trigger non_partecipa_amministrativo before insert
    or update
    on partecipa for each row
    execute procedure check_non_partecipa_amministrativo();
```

Tutti i trigger implementati sono nel file "Trigger.sql" in allegato

2.2 queries

Qui riportiamo alcune delle queries riportate nella tabella di "frequenza e tipo di operazioni" e altre queries più complicate.

```
    -- tutti i medici che hanno visitato il paziente ABCDEF
```

```
    select codiceMedico, nome, cognome
    from medico m
    where codiceMedico = any (select codiceMedico
        from medicoSeduta
        where cf = 'ABCDEF')
    or codiceMedico = any (select codiceMedico
        from medicoAppuntamento
        where cf = 'ABCDEF');
```

```
    -- per ogni paziente, i medici che lo hanno visitato pi
    spesso per problemi occasionali
    create view pazienteSedutaMedico as
    select cf, p1.nome nomePaziente, p1.cognome cognomePaziente
    , m1.codiceMedico codiceMedico, m1.nome nomeMedico, m1.
    cognome cognomeMedico, count(*) sedute
    from ((paziente p1 natural join seduta) natural join
    medicoSeduta) join medico m1 on medicoSeduta.codiceMedico =
     m1.codiceMedico
    group by p1.cf, m1.codiceMedico;

    select *
    from pazienteSedutaMedico s1
    where not exists (select *
        from pazienteSedutaMedico s2
        where s1.cf = s2.cf
        and s1.codiceMedico = s2.codiceMedico
        and s2.sedute > s1.sedute);
```

```
    -- medici che hanno seguito solo ed almeno una terapie che
    richiedevano la specializzazione ABCDEF
    select codiceMedico, nome, cognome
    from (medico m1 natural join medicoAppuntamento) natural
    join terapiaProlungata t1
    where t1.tipoDiSpecializzazione = 'ABCDEF'
    and not exists (select *
        from (medico m2 natural join medicoAppuntamento)
    natural join terapiaProlungata t2
        where t2.tipoDiSpecializzazione <> 'ABCDEF'
        and m1.codiceMedico = m2. codiceMedico);
```

Tutte le queries sono implementate nel file "Query.sql" in allegato