Cofinanziato dall'Unione europea

FinTech Software Developer
Basi di dati SQL

Docente: Roi Davide Simone

Titolo argomento: Sviluppo su Postgresql – Costrutti di querying
1° parte

Roi Davide
Dispense

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Costrutti principali della QUERY

Roi Davide Dispense

RAPPRESENTAZIONE E INDENTAZIONE

```
6 SELECT          Proietta
      lista_elementi_selezione
1 FROM            Seleziona
      lista_riferimenti_tabella/vista
2 [ WHERE         Condizione sulla selezione
      espressione_condizionale ]
3 [ GROUP BY      Raggruppa...
      lista_colonne ]
4 [ HAVING        ...i raggruppamenti Aventi
      espressione_condizionale ]
5 [ ORDER BY      Ordinati per
      lista_colonne ]
```

Le clausole obbligatorie in una query sono SELECT e FROM. La clausola SELECT, chiamata clausola di **proiezione** e determina le colonne che devono essere visualizzate nel risultato finale.
Mentre la clausola FROM è la clausola di **selezione** che determina da quale tabella/vista (o tabelle/viste) estrarre i dati.
La clausola WHERE è la clausola di **condizione** e viene utilizzata per filtrare le righe che verranno analizzate, mentre l'opzione ORDER BY viene utilizzata per ordinare il risultato finale. Nel caso in cui non venga specificata la clausola WHERE, tutte le righe verranno analizzate. Se non viene specificato alcun ordinamento, le righe verranno restituite senza un ordine specifico, in genere seguendo l'ordine della tabella.

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Costrutti principali della QUERY

Roi Davide Dispense

```
SELECT
        lista_colonne_selezione
```
<- **PROIEZIONE** (5 restituisce il risultato)

```
FROM
        lista_riferimenti_tabelle/viste
```
<- **SELEZIONE** (1 legge)

```
[ WHERE
        espressioni_condizionali ]
```
<- **CONDIZIONE sulla selezione** (2 filtra)

```
[ GROUP BY
        lista_colonne_raggruppamento ]
```
<- **RAGGRUPPAMENTO** (3 raggruppa)

```
[ HAVING
        espressioni_condizione_raggruppamento ]
```
<- **CONDIZIONE sul raggruppamento**
(4 filtra il raggruppamento)

```
[ ORDER BY
        lista_colonne ]
```
<- **ORDINAMENTO** (6 ordina il risultato)

```sql
select
    alias_tabella1.nome_campo1 as alias_campo1,
    alias_tabella2.nome_campo2 as alias_campo2,
    alias_select3.alias_campo10 as alias_campo3,
    case
        when alias_tabella1.nome_campo1 = 'ABC' then 1
        else 0
    end as alias_case4
from
    nome_tabella1 as alias_tabella1
inner join
    nome_tabella2 as alias_tabella2 on
        alias_tabella1.nome_campo1 = alias_tabella2.nome_campo1
left join
    (
    select
        nome_campo10 as alias_campo10,
        nome_campo20 as alias_campo20,
        nome_campo30 as alias_campo30
    from
        nome_tabella4 as alias_tabella4
    inner join
        nome_tabella5 as alias_tabella5 on
            alias_tabella4.nome_campo1 = alias_tabella5.nome_campo10
    ) as alias_select3 on
        alias_tabella2.nome_campo1 = alias_select3.nome_campo10
where
    condizione1
    and condizione2
    and condizione3
    and
    (
        condizione4
        or condizione5
    )
group by
    alias_tabella1.nome_campo1,
    alias_tabella2.nome_campo2,
    alias_select3.alias_campo10
having
    condizione1
    and condizione2
    and condizione3
    and
    (
        condizione4
        or condizione5
    )
order by
    alias_tabella1.nome_campo1,
    alias_tabella2.nome_campo2,
    alias_select3.alias_campo10
```

Indentazione SQL

Utilizzo di ALIAS per tabelle, viste e colonne

Roi Davide Dispense

In SQL è possibile associare un alias (nome abbreviativo) a:

- i nomi degli oggetti selezionati nel costrutto FROM (tabelle, viste, select annidate)
- i nomi delle colonne proiettate nel costrutto SELECT

Questa pratica è **fortemente raccomandata** quando si realizzano interrogazioni articolate in quanto permette di semplificare la comprensione e rendere il lavoro finale molto più pulito e preciso.

La sintassi per l'associazione di alias è la seguente:
nome_originale AS nome_alias
La parola chiave AS associa l'alias al nome originale

N.B. anche se la parola chiave **AS** è facoltativa si raccomanda di utilizzarla sempre.

```
--seleziona: nome, cognome, indirizzo dalla tabella dei clienti
filtrando i clienti con professione 'impiegato'
SELECT
  ft_cliente.nome,
  ft_cliente.cognome
FROM
  ft_cliente
 WHERE
   ft_cliente.professione ='impiegato';

--select riscritta con l'utilizzo degli alias
SELECT
   cl.nome AS name,
   cl.cognome AS  surname
FROM
  ft_cliente AS  cl
 WHERE
   cl.professione ='impiegato';
```

Carattere speciale * (asterisco)

Roi Davide
Dispense

Il carattere speciale * (asterisco) è una sitassi abbreviata per indicare in modo rapido TUTTE le colonne di una tabella/vista (senza doverle specificare manualmente una per una). Attenzione però che tale carattere NON permette il controllo puntuale di quali e quante colonne restituirà, ne sul loro ordine, in quanto il risultato verrà calcolato ogni volta in modo dinamico al momento dell'esecuzione della query. (se la struttura di una tabella varia varierà anche il risultato di questo comando). Per tale motivo **si sconsiglia** il suo utilizzo diretto all'interno di codice procedurale, fatta eccezione per le funzioni di aggregazione: sum, max, min, avg, count

NB la macchina sceglierà l'ordine di visualizzazione

```
--seleziona il cognome dalla tabella dei clienti,
--filtrando solo i clienti nati dopo il 01-01-2020
SELECT
  ft_cliente .cognome
FROM
  ft_cliente
 WHERE
  ft_cliente.data_nascita >='01-01-2000';

--seleziona TUTTE LE COLONNE dalla tabella dei clienti
SELECT
  ft_cliente.*
FROM
  ft_cliente;
```

```
--seleziona: nome, cognome, indirizzo dalla tabella dei clienti
filtrando i clienti con professione 'impiegato'
SELECT
  ft_cliente.nome,
  ft_cliente.cognome,
  ft_cliente.indirizzo
FROM
  ft_cliente
 WHERE
   ft_cliente.professione ='impiegato';
```

Filtrare e aggregare i dati

Roi Davide Dispense

Procediamo sul sito: https://www.sqltutorial.org/ per vedere le tecniche ed i comandi utilizzati per:

FILTRARE I DATI (Section 4: Filtering Data)

- DISTINCT
- LIMIT OFFSET
- WHERE
- OPERATORI (AND, OR, BEETWEEN, IN, LIKE, IS NULL, NOT)

FUNZIONI DI AGGREGAZIONE/RAGGRUPPAMENTO DATI (Section 7: Aggregate Functions)

- COUNT()
- SUM()
- MAX()
- MIN()
- AVG()
- < string_agg(nome_colonna , carattere_separatore) <– valida su postgresql e sqlserver >
- < group_concat(nome_colonna SEPARATOR carattere_separatore) <– valida su mysql >

GROUP BY e HAVING

Il costrutto **GROUP BY** viene utilizzato per raggruppare insieme le righe di una tabella in base ai valori presenti in una o più colonne. Quando si utilizza la clausola GROUP BY, il risultato della query viene diviso in **gruppi di righe** che condividono lo stesso valore nelle colonne specificate. In seguito, è possibile applicare una funzione di aggregazione (COUNT, SUM, MIN, MAX, AVG) a ogni gruppo di righe, in modo da ottenere un valore unico per ogni gruppo. In questo modo, è possibile eseguire calcoli su gruppi specifici di righe della tabella e ottenere informazioni aggregate su quei gruppi.

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
Monitor-CX12	1	148.00
Monitor-CX12	13	129.00
Maose-M23	12	19.90
Maose-M23	3	19.90

GROUP BY prodotto

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
	1	148.00
	13	129.00
Maose-M23	12	19.90
	3	19.90

GROUP BY e HAVING

Ad esempio, supponiamo di avere una tabella «vendite" con le seguenti colonne: " prodotto", " quantita", «prezzo_unitario". Se vogliamo sapere le quantità vendute per ogni prodotto, possiamo usare la clausola GROUP BY insieme alla funzione di aggregazione SUM in questo modo:

```
SELECT
    v.prodotto,
    SUM(v.quantita)  as quantita
FROM
    vendite as v
GRUOP BY
    v.prodotto;
```

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
Monitor-CX12	1	148.00
Monitor-CX12	13	129.00
Maose-M23	12	19.90
Maose-M23	3	19.90

GROUP BY prodotto

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
	1	148.00
	13	129.00
Maose-M23	12	19.90
	3	19.90

GROUP BY e HAVING

Roi Davide
Dispense

Supponiamo ora di voler sapere oltre alla quantità totale anche il prezzo massimo unitario.
Sarà sufficiente appllicare la funzione di raggruppamento MAX alla colonna prezzo_unitario

```
SELECT
    v.prodotto,
    SUM(v.quantita) AS quantita_totale,
    MAX(v.prezzo_unitario) AS prezzo_massimo
FROM
    vendite as v
GRUOP BY
    v.prodotto;
```

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
Monitor-CX12	1	148.00
Monitor-CX12	13	129.00
Maose-M23	12	19.90
Maose-M23	3	19.90

GROUP BY prodotto

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
	1	148.00
	13	129.00
Maose-M23	12	19.90
	3	19.90

Intervento realizzato da ITS Tecnologie dell'Informazione e della Comunicazione

GROUP BY e HAVING

(esempio di errore di raggruppamento)

Roi Davide Dispense

```
SELECT
    v.prodotto,  COLONNA RAGGRUPPATA
    v.quantita  COLONNA NON RAGGRUPPATA
FROM
    vendite as v
GROUP BY
    v.prodotto;
```

prodotto	quanita	prezzo_unitario
Tastiera-R1	3	49.90
Monitor-CX12	2	148.00
	1	148.00
	13	129.00
Maose-M23	12	19.90
	3	19.90

ERRORE!!! Quando uso un raggruppamento non posso chiedere dettagli multi-record per raggruppamento.

Per questo motivo nei costrutti GROUP BY e HAVING si utilizzano le «funzioni di aggregazione» che garantiscono la restituzione di un valore unico per gruppo:
COUNT, MIN, MAX, AVG, SUM

GROUP BY e HAVING

Roi Davide Dispense

Il comando GROUP BY agisce sul risultato prodotto dai precedenti 2 costrutti:
(Ricordiamo l'ordine di ESECUZIONE dei costrutti)

1. FROM (selezione)
2. [WHERE (condizione si selezione)]
3. [GROUP BY (raggruppamento)]
4. [HAVING (condizione su raggruppamento)]
5. SELECT (proiezione)
6. [ORDER (ordinamento)]

Il costrutto **HAVING** filtra i gruppi di righe generati dal costrutto GROUP BY in base a condizioni specifiche sulle funzioni di aggregazione (COUNT, SUM, MIN, MAX, AVG).

```
-- filtra i gruppi di prodotti che hanno più di 1 record
SELECT
    v.prodotto
FROM
    vendite as v
GRUOP BY
    v.prodotto
HAVING
    COUNT(*)>1;

-- filtra i gruppi di prodotti che hanno una quantità > 3
SELECT
    v.prodotto,
    sum(v.quantita) as quantita
FROM
    vendite as v
GRUOP BY
    v.prodotto
HAVING
     SUM(v.quantita) > 3
```

Fonti:

- Documentazione ufficiale di Postgresql
  https://www.postgresql.org/docs/

- SQL online tutorial.org
  https://www.sqltutorial.org/

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Fine della Presentazione