

SQL: Structured Query Language



SQL:Componenti Principali

- ❖ Data Manipulation Language (DML): interrogazioni, inserimenti, cancellazioni, modifiche
- ❖ Data Definition Language (DDL): creazione, cancellazione e modifica di tabelle e viste
- ❖ Trigger: azioni eseguite dal DBMS che soddisfano determinate condizioni
- ❖ SQL dinamico e embedded
- ❖ Esecuzione client-server
- ❖ Gestione di transazioni
- ❖ Sicurezza

Interrogazioni SQL di base

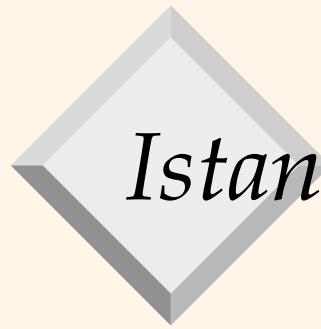
-SELECT (DISTINCT)	lista-attributi
-FROM	lista-relazioni
-WHERE	qualificazioni

- ❖ *Lista-relazioni.* Una lista di nomi di relazioni
- ❖ *Lista-attributi.* Una lista di attributi delle relazioni in *lista-relazioni*
- ❖ *Qualificazioni.* Confronti (*attr op cost* oppure *attr1 op attr2*, dove *op* è uno tra $<$, $>$, $=$, \leq , \geq , \neq) combinati usando AND, OR e NOT
- ❖ DISTINCT è una parola chiave opzionale che indica che la risposta non deve contenere duplicati (per impostazione predefinita i duplicati *non* sono eliminati)



Strategia di valutazione concettuale

- ❖ La semantica di una interrogazione SQL è definita in termini della seguente strategia di valutazione concettuale:
 - calcolare il prodotto cartesiano di *lista-relazioni*
 - scartare le tuple risultanti se non passano le *qualificazioni*
 - cancellare gli attributi che non sono in *lista-attributi*
 - se è specificato DISTINCT, eliminare le righe duplicate
- ❖ Questa strategia è probabilmente il modo meno efficiente di calcolare una interrogazione! Un ottimizzatore troverà strategie più efficienti per calcolare *le stesse risposte*



Istanze di esempio

- ❖ Useremo nei nostri esempi queste istanze delle relazioni Velisti e Prenota

-P1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

-V1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0


-V2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Esempio di valutazione concettuale

-SELECT V.vnome
-FROM Velisti V, Prenota P
-WHERE V.vid = P.vid AND P.bid = 103

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96



Variabili di range

- ❖ Variabili usate nella lista-relazioni per denotare specifiche istanze di relazioni
- ❖ Se ne ha veramente bisogno solo se la stessa relazione appare due volte nella clausola FROM
- ❖ È buono stile, comunque, usare sempre le variabili di range

Variabili di range

- ❖ L'interrogazione precedente può anche essere scritta come:

```
-SELECT      V.vnome
-FROM        Velisti V, Prenota P
-WHERE       V.vid = P.vid AND bid = 103
             -OPPURE
-SELECT      vnome
-FROM        Velisti, Prenota
-WHERE       Velisti.vid = Prenota.vid
             AND bid = 103
```



Trovare i velisti che hanno prenotato almeno una barca

```
-SELECT      V.vid  
-FROM        Velisti V, Prenota P  
-WHERE       V.vid = P.vid
```

- ❖ Farebbe differenza aggiungere DISTINCT a questa interrogazione?
- ❖ Qual è l'effetto della sostituzione di V.vid con V.vnome nella clausola SELECT? Farebbe differenza aggiungere DISTINCT a questa variante dell'interrogazione?



Select con asterisco ()*

Data una relazione **R** sugli attributi **A, B, C**

```
select    *  
from      R  
where     cond
```

equivale a

```
select  A, B, C  
from    R  
where   cond
```

Espressioni e stringhe

- ❖ Lista-attributi può essere estesa considerando:

Espressione AS nome_colonna

- ❖ Operatore *LIKE*: supporta il pattern matching

- % zero o più caratteri qualunque
- _ esattamente un carattere qualunque
- Es.: “_AB%” qualunque stringa di almeno tre caratteri con secondo e terzo pari ad A e B



Espressioni e stringhe


-SELECT	V.età, età1 = V.età -5, 2 * V.età AS eta2
-FROM	Velisti V
-WHERE	V.vnome LIKE 'B_%B'

- ❖ Trovare le triple (le età dei velisti e due campi definiti da espressioni) per quei velisti il cui nome inizia e termina con B e contiene almeno tre caratteri)
- ❖ **AS** e **=** sono due modi di dare un nome ai campi del risultato



UNION, INTERSECT ed EXCEPT

- ❖ *UNION, INTERSECT ed EXCEPT: possono essere usati per calcolare rispettivamente l'unione, l'intersezione e la differenza su qualunque coppia di tabelle compatibili rispetto all'unione*
 - Stesso numero di colonne*
 - Colonne prese in ordine dello stesso tipo*



Trovare i nomi dei velisti che hanno prenotato una barca rossa o una barca verde

```
-SELECT V.vnome  
-FROM Barche B, Prenota P, Velisti V  
-WHERE P.bid=B.bid AND P.vid=V.vid  
      AND (B.colore='rosso' OR B.colore='verde')
```

```
-SELECT V.vnome  
-FROM  Barche B, Prenota P, Velisti V  
-WHERE P.bid = B.bid AND P.vid=V.vid  
      AND B.colore = 'rosso'  
-UNION  
-SELECT V.vnome  
-FROM  Velisti V, Barche B, Prenota P  
-WHERE V.vid = P.vid AND P.bid = B.bid  
      AND B.colore = 'verde'
```

Trovare i nomi dei velisti che hanno prenotato una barca rossa e una barca verde

```
-SELECT V.vnome
-FROM  Velisti V, Barche B1, Prenota P1,
-      Barche B2, Prenota P2
-WHERE V.vid = P1.vid AND P1.bid = B1.bid
-      AND V.vid = P2.vid AND P2.bid = B2.bid
-      AND (B1.colore = 'rosso' AND B2.colore = 'verde')
```

```
-SELECT V.vid
-FROM  Velisti V, Barche B, Prenota P
-WHERE V.vid = P.vid AND P.bid = B.bid
-      AND B.colore = 'rosso'
-INTERSECT
-SELECT V.vid
-FROM  Velisti V, Barche B, Prenota P
-WHERE V.vid = P.vid AND P.bid = B.bid
-      AND B.colore = 'verde'
```

NOTA Vid
CHIAVE



UNION ALL, INTERSECT ALL, EXCEPT ALL

- ❖ *INTERSECT, UNION e EXCEPT eliminano i duplicati*
- ❖ *Le versioni UNION ALL, INTERSECT ALL e EXCEPT ALL consentono di mantenere i duplicati nel risultato*



Interrogazioni annidate

- ❖ Un'interrogazione può contenere al suo interno un'altra interrogazione
- ❖ Tipicamente la sottointerrogazione è contenuta nella clausola WHERE (ma sottointerrogazioni possono comparire anche nelle clausole FROM e HAVING)

Interrogazioni annidate

Trovare i nomi dei velisti che hanno prenotato la barca #103:

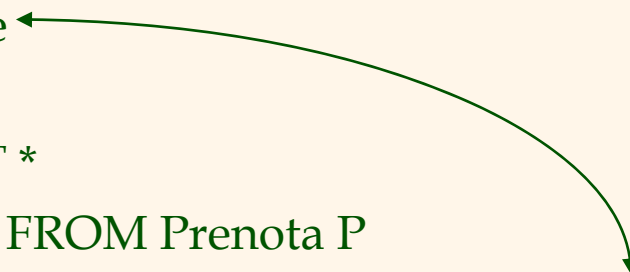
```
-SELECT V.vnome  
-FROM Velisti V  
-WHERE V.vid IN (SELECT P.vid  
- FROM Prenota P  
- WHERE P.bid = 103)
```

- ❖ Per trovare i velisti che *non* hanno prenotato la barca #103, usare NOT IN
- ❖ Per comprendere la semantica delle interrogazioni annidate, pensate alla valutazione dei *cicli annidati*: per ciascuna tupla di *Velisti*, controllare la qualificazione calcolando la sottointerrogazione.

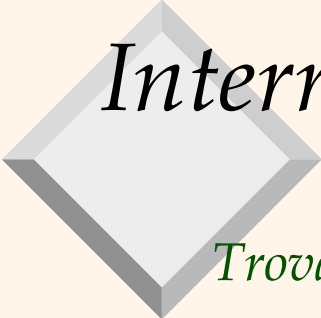
Interrogazioni annidate con correlazione

-Trovare i nomi dei velisti che hanno prenotato la barca #103:

```
-SELECT      V.vnome
-FROM        Velisti V
-WHERE EXISTS (SELECT *
-            FROM Prenota P
-            WHERE P.bid = 103 AND V.vid = P.vid)
```



- ❖ EXISTS è un altro operatore di confronto tra insiemi, come IN
- ❖ Ritorna TRUE se l'insieme a cui è applicato è non vuoto, FALSE altrimenti
- ❖ Illustra perché, in generale, la sottointerrogazione deve essere ri-calcolata per ogni tupla di Velisti



Interrogazioni annidate con correlazione

Trovare i nomi dei velisti che hanno prenotato la barca #103 :

```
-SELECT      V.vnome  
-FROM        Velisti V  
-WHERE UNIQUE (SELECT PID  
-              FROM Prenota P  
-              WHERE P.bid = 103 AND V.vid = P.vid)
```

- ❖ UNIQUE restituisce TRUE se non ci sono tuple duplicate nel risultato, FALSE altrimenti
- ❖ Perché abbiamo sostituito * con P.bid?

Ancora sugli operatori di confronto tra insiemi

- ❖ Abbiamo già visto IN, EXISTS e UNIQUE. Possiamo anche usare NOT IN, NOT EXISTS e NOT UNIQUE
- ❖ Disponibili anche: op ANY, op ALL, op IN $>$, $<$, $=$, \geq , \leq , \neq
 - **any**: vero, se il confronto è vero per **una qualunque** delle tuple risultato dell'interrogazione nidificata
 - **all**: vero, se il confronto è vero per **tutte** le tuple risultato dell'interrogazione nidificata




Ancora sugli operatori di confronto tra insiemi

- ❖ Trovare i velisti la cui esperienza è maggiore di quella di qualche velista chiamato Horatio:

```
-SELECT *  
-FROM   Velisti V  
-WHERE  V.esperienza > ANY (SELECT V2.esperienza  
                             FROM   Velisti V2  
                             WHERE  V2.vnome = 'Horatio')
```

-Se non ci sono velisti chiamati Horatio il confronto restituisce FALSE



Riscrittura delle interrogazioni INTERSECT usando IN

-Trovare i vid dei velisti che hanno prenotato sia una barca rossa che una barca verde:

```
-SELECT V.vnome
-FROM   Velisti V, Barche B, Prenota P
WHERE  V.vid = P.vid AND P.bid = B.bid AND B.colore = 'rosso'
        AND V.vid IN (SELECT   V2.vid
                       FROM     Velisti V2, Barche B2, Prenota P2
                       WHERE  V2.vid = P2.vid AND P2.bid = B2.bid
                           AND B2.colore = 'verde')
```

❖ Analogamente, le interrogazioni EXCEPT si riscrivono usando NOT IN



Divisioni in SQL

-Trovare i velisti che hanno prenotato tutte le barche.

(1) senza EXCEPT:

-SELECT V.vnome

-FROM Velisti V

-WHERE NOT EXISTS (SELECT B.bid
FROM Barche B
WHERE NOT EXISTS

(SELECT P.bid
FROM Prenota P
WHERE P.bid = B.bid
AND P.vid = V.vid))

Divisioni in SQL

-Trovare i velisti che hanno prenotato tutte le barche.

(2) con EXCEPT:

```
-SELECT V.vnome
-FROM   Velisti
-WHERE NOT EXISTS
        ((SELECT   B.bid
          FROM     Barche B)
        EXCEPT
        (SELECT   P.bid
          FROM     Prenota P
          WHERE    P.vid = V.vid))
```



Operatori di aggregazione

- **Importante estensione dell'algebra relazionale.**

-SELECT COUNT(*)


-FROM Velisti V

-SELECT COUNT (DISTINCT V.esperienza)

-FROM Velisti V

-WHERE V.vnome = 'Bob'

-COUNT	(*)
-COUNT	([DISTINCT] A)
-SUM	([DISTINCT] A)
-AVG	([DISTINCT] A)
-MAX	(A)
-MIN	(A)



Operatori di aggregazione: AVG

```
-SELECT  AVG(V.età)  
-FROM    Velisti V  
-WHERE   V.esperienza=10
```

```
-SELECT AVG(DISTINCT V.età)  
-FROM   Velisti V  
- WHERE V.esperienza = 10
```



Trovare nome ed età del/i velista/i più anziano/i

- ❖ La prima interrogazione è illegale! (Ne vedremo le ragioni un po' più tardi, quando discuteremo GROUP BY)
- ❖ La terza interrogazione è equivalente alla seconda, ed è permessa nello standard SQL/92, ma non è supportata in alcuni sistemi

```
-SELECT V.vnome, MAX(V.età)  
-FROM Velisti V
```

```
-SELECT V.vnome, V.età  
-FROM Velisti V  
-WHERE V.età =  
      (SELECT MAX(V2.età)  
       FROM Velisti V2)
```

```
-SELECT V.vnome, V.età  
-FROM Velisti V  
-WHERE (SELECT MAX(V2.età)  
       FROM Velisti V2) = V.età
```



GROUP BY e HAVING

- ❖ Finora abbiamo applicato operatori di aggregazione a tutte le tuple (qualificanti). A volte vogliamo applicarli a ciascuno tra diversi *gruppi* di tuple
- ❖ Consideriamo: Trovare l'età del velista più giovane per ciascun grado di esperienza
 - In generale, non sappiamo quanti gradi di esperienza esistano, e quali siano i loro valori!
 - Supponiamo di sapere che i valori di esperienza variano da 1 a 10: possiamo scrivere 10 interrogazioni che somigliano a queste (!)

	-SELECT	MIN(V.età)
-For $i = 1, 2, \dots, 10$:	-FROM	Velisti V
	-WHERE	V.esperienza = i



Interrogazioni con *GROUP BY* e *HAVING*

-SELECT	[DISTINCT] lista-target
-FROM	lista-relazioni
-WHERE	qualificazioni
-GROUP BY	lista-gruppi
-HAVING	qualificazione-gruppi

- ❖ La *lista-target* contiene (i) nomi di attributi (ii) termini con operazioni di aggregazione (ad esempio, MIN(V.età))
 - La lista di attributi (i) deve essere un sottoinsieme della lista-gruppi. Intuitivamente, ciascuna tupla nella risposta corrisponde a un *gruppo*, e questi attributi devono avere un singolo valore per gruppo (un *gruppo* è un insieme di tuple con lo stesso valore per tutti gli attributi in *lista-attributi*)

Valutazione concettuale

- ❖ Viene calcolato il prodotto cartesiano della *lista-relazioni*, vengono scartate le tuple che non passano la *qualificazione*, i campi “non necessari” vengono cancellati, e le tuple rimanenti sono partizionate in gruppi dai valori degli attributi in *lista-gruppi*.
- ❖ La *qualificazione-gruppi* viene poi applicata per eliminare alcuni gruppi. Le espressioni in *qualificazione-gruppi* devono avere un singolo valore per gruppo!
 - In effetti, un attributo in *qualificazione-gruppi* che non è un argomento di un operatore di aggregazione appare anche in *lista-gruppi* (SQL qui non sfrutta la semantica delle chiavi primarie!)
- ❖ Viene generata una tupla di risposta per ogni gruppo qualificante

Trovare l'età del velista più giovane con età ≥ 18 , per ciascun livello di esperienza con almeno 2 velisti.

```
-SELECT V.esperienza, MIN(V.età)
-FROM   Velisti V
-WHERE  V.età >= 18
-GROUP BY      V.esperienza
-HAVING        COUNT(*) > 1
```


- ❖ Solo V.esperienza e V.età sono menzionati nelle clausole SELECT, GROUP BY e HAVING; altri attributi sono “non necessari”
- ❖ La seconda colonna del risultato non ha un nome (usare AS per darle un nome)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

-Relazione risultato



Per ciascuna barca rossa, trovare il numero di prenotazioni per tale barca.

```
-SELECT      B.bid, COUNT(*) AS vconta  
-FROM        Velisti V, Barche B, Prenota P  
-WHERE       V.vid = P.vid AND P.bid = B.bid AND B.colore = 'rosso'  
-GROUP BY   B.bid
```

- ❖ Raggruppamento su un join di tre relazioni
- ❖ Cosa otteniamo se rimuoviamo B.colore = 'rosso' dalla clausola WHERE e aggiungiamo una clausola HAVING con tale condizione?
- ❖ Che succede se eliminiamo Velisti e la condizione che coinvolge V.vid?

Trovare l'età del velista più giovane con età >18 per ciascun livello di esperienza con almeno 2 velisti (di qualunque età)

-SELECT V.esperienza, MIN(V.età)

-FROM Velisti V

-WHERE V.età > 18

-GROUP BY V.esperienza

-HAVING 1 < (SELECT COUNT(*)

- FROM Velisti V2

- WHERE V.esperienza = V2.esperienza)

- ❖ Mostra come la clausola HAVING possa anche contenere una sottointerrogazione
- ❖ Confrontate questa interrogazione con quella in cui consideravamo solo i livelli di esperienza con due velisti con più di 18 anni!
- ❖ Che succede se la clausola HAVING viene sostituita con
 - HAVING COUNT(*) > 1

*Trovare quei livelli di esperienza per cui
l'età media è minima su tutti i livelli*

- Gli operatori di aggregazione non possono essere annidati! SBAGLIATO:

```
-SELECT      V.esperienza
-FROM        Velisti V
-WHERE       V.età = (SELECT MIN(AVG(V2.età)) FROM Velisti V2)
```

- ❖ Soluzione corretta (in SQL/92):

```
-SELECT      Temp.esperienza. Temp.etamedia
-FROM        (SELECT      V.esperienza, AVG(V.età) AS etamedia
-              FROM      Velisti V
-              GROUP BY  V.esperienza) AS Temp
-WHERE       Temp.etamedia = (SELECT MIN(Temp.etamedia)
-                               FROM Temp)
```

Valori Null

- ❖ I valori dei campi di una tupla sono a volte *sconosciuti* (ad esempio, non è ancora stato stabilito un livello di esperienza) oppure *inapplicabili* (ad esempio, nessuna moglie)
 - SQL fornisce uno speciale valore *null* per tali situazioni
- ❖ La presenza di *null* complica parecchie cose. Ad esempio:
 - operatori speciali sono necessari per controllare se un valore è/non è *null*
 - *esperienza > 8* è vera o falsa quando *esperienza* è *null*? Che si può dire dei connettori AND, OR e NOT?
 - Abbiamo bisogno di una logica a 3 valori (*vero, falso e sconosciuto*)
 - Il significato dei costrutti deve essere attentamente definito (ad esempio la clausola WHERE elimina le righe che non vengono valutate come *vero*)
 - Nuovi operatori (in particolare di join esterno) sono possibili/necessari

Sommario

- ❖ Un fattore importante nel rapido sviluppo del modello relazionale; più naturale che in precedenza, linguaggi di interrogazione procedurali
- ❖ Relazionalmente completo; di fatto, potere espressivo significativamente superiore all'algebra relazionale
- ❖ Persino le interrogazioni che possono essere espresse in AR possono spesso essere espresse in maniera più naturale con SQL
- ❖ Molti modi alternativi di scrivere una interrogazione; l'ottimizzatore dovrebbe cercare il piano di valutazione più efficiente
 - Nella pratica, gli utenti devono essere consci di come le interrogazioni sono ottimizzate e valutate per ottenere risultati migliori



Esercizio

- Fornitori(fid:integer, fnome:string, indirizzo:string)
 - Pezzi(pid:integer, pnome:string, colore:string)
 - Catalogo(fid:integer, pid:integer, costo:real)
-
1. Trovare i pnome dei pezzi per cui esiste un qualche fornitore
 2. Trovare gli fnome dei fornitori che forniscono ogni pezzo
 3. Trovare gli fnome dei fornitori che forniscono tutti i pezzi rossi



-Esercizio (segue)

4. Trovare i *pnome* dei pezzi forniti dalla Acme e da nessun altro
5. Trovare i *fid* dei fornitori che ricaricano su alcuni pezzi più del costo medio di quel pezzo
6. Per ciascun pezzo, trovare gli *fnome* dei fornitori che ricaricano di più su quel pezzo
7. Trovare i *fid* dei fornitori che forniscono solo pezzi rossi
8. Trovare i *fid* dei fornitori che forniscono un pezzo rosso e un pezzo verde
9. Trovare i *fid* dei fornitori che forniscono un pezzo rosso o uno verde

Soluzioni

1. SELECT P.pnome
FROM Pezzi P, Catalogo C
WHERE P.pid = C.pid

2. SELECT F.fnome
FROM Fornitori F
WHERE NOT EXISTS ((SELECT P.pid FROM Pezzi P)
EXCEPT
(SELECT C.pid
FROM Catalogo C
WHERE C.fid = F.fid))

Soluzioni (segue)

3.

```
SELECT    F.fnome
FROM      Fornitori F
WHERE NOT EXISTS ((SELECT * FROM Pezzi P)
                  WHERE    P.colore = 'rosso')
EXCEPT
(SELECT    C.pid
FROM      Catalogo C, Pezzi P
WHERE     C.fid = F.fid AND
          C.pid = P.pid AND P.colore = 'rosso'))
```
4.

```
SELECT    P.pnome
FROM      Pezzi P, Catalogo C, Fornitori F
WHERE     P.pid = C.pid AND C.fid = F.fid
          AND F.fnome = 'Acme'
AND NOT EXISTS (SELECT * FROM Catalogo C1, Fornitori F1)
              WHERE P.pid = C1.pid AND C1.fid = F1.fid
                  AND F1.fnome <> 'Acme'
```

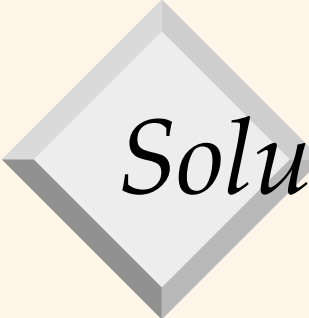
Soluzioni (segue)

5.

```
SELECT      DISTINCT C.fid
FROM        Catalogo C
WHERE       C.costo > (SELECT AVG(C1.costo)
                       FROM Catalogo C1
                       WHERE C1.pid = C.pid)
```
6.

```
SELECT      P.pid, F.fnome
FROM        Fornitori F, Catalogo C
WHERE       C.fid = F.fid
AND C.costo = (SELECT  MAX(C1.costo)
               FROM    Catalogo C1
               WHERE C1.pid = C.pid))
```
7.

```
SELECT      DISTINCT C.fid
FROM        Catalogo C
WHERE NOT EXISTS (SELECT * FROM Pezzi P, Catalogo C1
                 WHERE P.pid = C.pid AND P.colore <> 'rosso'
                 AND C.fid = C1.fid)
```



Soluzioni (segue)

8. SELECT DISTINCT C.fid
FROM Catalogo C, Pezzi P
WHERE C.pid = P.pid AND P.colore = 'rosso'
INTERSECT
SELECT DISTINCT C.fid
FROM Catalogo C1, Pezzi P1
WHERE C1.pid = P1.pid AND P1.colore = 'verde'
9. SELECT DISTINCT C.fid
FROM Catalogo C, Pezzi P
WHERE C.pid = P.pid AND P.colore = 'rosso'
UNION
SELECT DISTINCT C1.fid
FROM Catalogo C1, Pezzi P1
WHERE C1.pid = P1.pid AND P1.colore = 'verde'