Università degli Studi di Padova
Corso di Laurea Magistrale in Bioingegneria

A.A. 2010-2011

INFORMATICA SANITARIA
(Lezione SQL 1)

Barbara Di Camillo
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Padova
Via Ognissanti 72, 35129 Padova
e-mail: barbara.dicamillo@dei.unipd.it
Si ringrazia il Dott. Andrea Facchinetti per gli utili suggerimenti

Lezione SQL 1

- Che cos’è SQL
- Notazione sintassi propria dei lucidi e di SQL
- Creazione e manipolazione di Tabelle
- Interrogazione di Tabelle (QUERY)

SQL (Structured Query Language)

- Linguaggio per basi di dati relazionali
- Linguaggio per la creazione, manipolazione, controllo, interrogazione di un Database relazionale
- Le procedure SQL manipolano delle Tabelle e consentono di interrogare la base dati, estraendo altre tabelle o creandone di nuove.
- SQL utilizza operatori di algebra relazionale, più operatori aggiuntivi

SQL (Structured Query Language)

- SQL è un linguaggio dichiarativo, non procedurale: l'utente specifica cosa fare ma non come deve essere fatto (dichiara quali dati recuperare, non come)

- Immediatamente adottato come standard "de facto": utilizzato dai principali pacchetti per la gestione di basi di dati, come Oracle, Informix, Sybase e, per applicazioni "home made" come Access.

- Esistono vari dialetti

Lezione SQL 1

- Che cos’è SQL
- Notazione sintassi propria dei lucidi e di SQL
- Creazione e manipolazione di Tabelle
- Interrogazione di Tabelle (QUERY)

Notazione Sintassi (lucidi)

- < > isolano i termini della sintassi
- [ ] indicano un termine che può comparire al massimo 1 volta (0 o 1 volta)
- { } indicano un termine che può comparire ripetutamente (≥0)
- | OR

Notazione Sintassi (lucidi)

- Useremo le lettere maiuscole per le PAROLE CHIAVE delle istruzioni (talvolta in blu o in rosso per metterle in evidenza)

- Useremo il colore rosso per i caratteri speciali:

  % * + - ‘ ( ) “ / , : ? . ; < = > | &

- Useremo le lettere minuscole per gli identificatori delle tabelle (relazioni) e delle colonne (attributi)

Notazione Sintassi (SQL)

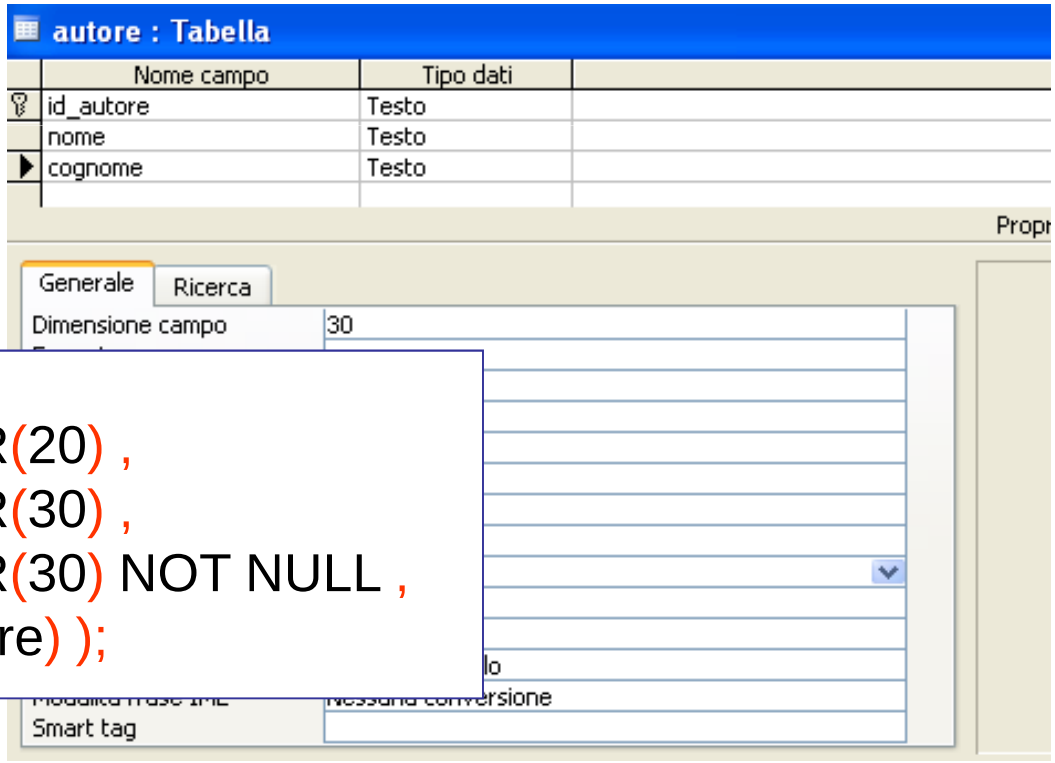
- SQL è case-insensitive
- I nomi degli identificatori devono iniziare con una lettera e non devono superare 18 caratteri
- I nomi degli identificatori non possono corrispondere a parole chiave e non possono contenere caratteri speciali:
  % * + - ‘ ( ) “ / , : ? . ; < = > | &
- Lo spazio è un carattere speciale; quindi, ad esempio, non posso usare l’identificatore: *titolo libro*
  ma posso usare invece: *titolo_libro*
- Ogni istruzione SQL termina con il carattere **;**

Lezione SQL 1

- Che cos’è SQL
- Notazione sintassi propria dei lucidi e di SQL
- Creazione e manipolazione di Tabelle
- Interrogazione di Tabelle (QUERY)

Creazione di Tabelle

```
CREATE TABLE <nome_tabella>
  (<nome_col1>    <TIPO_DATI> [<VINCOLO_COLONNA>]
   {, <nome_col2>  <TIPO_DATI> [<VINCOLO_COLONNA>]}
    , PRIMARY KEY (<nome_col_chiave_primaria>)
  );
```




```
CREATE TABLE autore
   (id_autore      VARCHAR(20) ,
    nome           VARCHAR(30) ,
    cognome        VARCHAR(30) NOT NULL ,
    PRIMARY KEY (id_autore) );
```

Creazione di Tabelle

```
CREATE TABLE <nome_tabella>
  (<nome_col1>    <TIPO_DATI> [<VINCOLO_COLONNA>]
   {, <nome_col2>  <TIPO_DATI> [<VINCOLO_COLONNA>]}
    PRIMARY KEY (<nome_col_chiave_primaria>)
    , FOREIGN KEY (<nome_col_chiave_esterna>)
   REFERENCES  <nome_tab_esterna> (<nome_col_esterne>)
   );
```

IMPORTANTE!!! La colonna esterna DEVE essere un identificatore della Tabella Esterna.

```
CREATE TABLE libri
   (id_libro        INT ,
    titolo          CHAR(50) ,
    id_scrittore    CHAR(20) ,
    id_editore      CHAR(30) NOT NULL ,
    anno  CHAR(20) ,
    PRIMARY KEY (id_libro ) ,
    FOREIGN KEY (id_scrittore) REFERENCES autore (id_autore) ,
    FOREIGN KEY (id_editore) REFERENCES edizioni (id_editore)  );
```

ATTENZIONE!!!
Le tabelle “autore” e “edizioni” devono essere create prima della tabella “libri”, altrimenti il comando non viene eseguito.

Vincoli colonna

- NOT NULL
- UNIQUE
- CHECK (<CONDIZIONE>)
- DEFAULT

- Le chiavi primarie sono di default NOT NULL e UNIQUE

```
CREATE TABLE produzione
   (id_scrittore    CHAR(20) ,
    num_libri       INT   NOT NULL DEFAULT 1 CHECK (num_libri>0),
    PRIMARY KEY (id_scrittore)  );
```

Non disponibile in Access

Tipi di dati in SQL (ACCESS)

- BIT(n): sequenza di n bit 0, 1 (FALSO, VERO)
- INT (da –32.767 a 32.767)
- INTEGER: numeri interi con segno (da -2.147.483.647 a + 2.147.483.647)
- REAL: numero in virgola mobile a precisione singola (32 bit)
- FLOAT: numero in virgola mobile a precisione doppia (64 bit)

L’identificativo del tipo di dato e l’intervallo di validità di alcuni tipi di dati possono cambiare leggermente a seconda della versione di Access/SQL.

Tipi di dati in SQL (ACCESS)

- MONEY: valore intero scalare tra -922.337.203.685.477,5808 e 922.337.203.685.477,5807
- DATE/TIME: data espressa come giorno, mese, anno (15-Apr-2009) e/o orario espresso come ore, minuti, secondi (12:00:00)
- CHAR (o VARCHAR): stringa di n caratteri (da 0 a 255)
- TEXT: testo (da 0 a 2.11 GB)
- IMAGE: oggetto OLE (da 0 a 2.11 GB) (OLE=Object Linking and Embedding)

L'identificativo del tipo di dato e l'intervallo di validità di alcuni tipi di dati possono cambiare leggermente a seconda della versione di Access/SQL.

Eliminazione e modifica di Tabelle

✓Eliminazione:

```
DROP TABLE <nome-tabella>;
```

✓Modifica:

```
ALTER TABLE <nome-tabella>
    ADD <nome_col>  <TIPO_DATI>  [<VINCOLO_COLONNA>]
      |
    {DROP          <nome_col> {, <nome_col> } }
      |
    {ALTER COLUMN <nome_col>
          SET DEFAULT <nuovo_default>
          |
          DROP DEFAULT}
};
```

Non disponibile in Access

Eliminazione e modifica di Tabelle

Se voglio modificare un tipo di dati devo prima cancellare la colonna e poi ridefinirla.
Ad esempio, se data_nascita è INT e voglio modificare il tipo di dati (DATE invece di INT):

```
ALTER TABLE autore
DROP  data_nascita;
```

```
ALTER TABLE autore
ADD   data_nascita      DATE;
```

In Access sono 2 query!

Inserimento di record nelle tabelle

Ora inseriamo i record nella tabella (lavoriamo sulle righe)…

```
INSERT INTO nome_tabella
  (nome-col  {, nome-col} )
 VALUES  (valore {, valore} );
```

```
INSERT INTO edizioni
   (id_editore, nome)
VALUES (12, “pippo”);
```

- Non è indispensabile indicare il nome delle colonne; se esse non vengono specificate, SQL assume che si mantenga l’ordine di creazione della Tabella
- Se si specificano le colonne, si possono mettere in qualunque ordine
- Se mancano colonne, SQL introduce dei NULL
- ATTENZIONE: bisogna inserire dati compatibili con il tipo dati delle colonne (char tra virgolette “valore”)

Inserimento di record nelle tabelle

- Inserendo un dato non corrispondente al tipo e non compreso tra virgolette è possibile realizzare una query interattiva

```
INSERT INTO edizioni
   (id_editore, nome)
VALUES (inserisci_ID_editore, inserisci_nome_editore);
```

ATTENZIONE!!!
È una possibilità fornita solo da Access (non è propria di SQL)

Eliminazione e Modifica di record dalle tabelle

```
DELETE [*] FROM <nome_tabella>
   WHERE <nome_colX> = <valoreX>;

UPDATE <nome_tabella>
   SET <nome_colX> = <valoreX>
   WHERE <nome_colY> = <valoreY>;
```

```
DELETE FROM produzione
  WHERE  id_scrittore="AM122";
```

```
DELETE * FROM produzione
  WHERE  id_scrittore="AM122";
```

```
UPDATE produzione
  SET num_libri=13
  WHERE  id_scrittore="AM122";
```

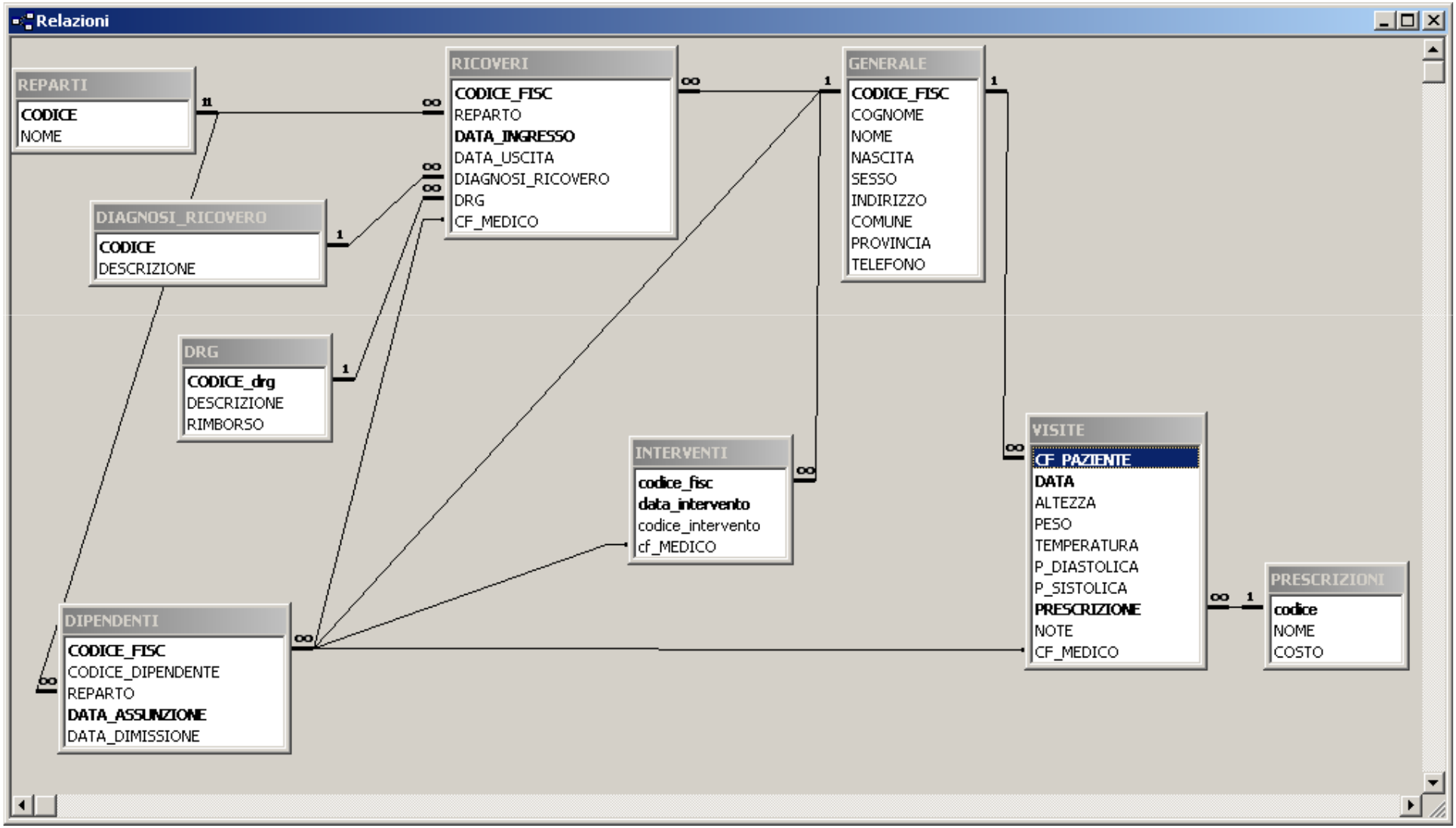
Lezione SQL 1

- Che cos’è SQL
- Notazione sintassi propria dei lucidi e di SQL
- Creazione e manipolazione di Tabelle
- Interrogazione di Tabelle (QUERY)
  - Cos’è una Query?
  - Sintassi dell’istruzione SELECT
  - Ordine Scrittura ≠ Ordine Elaborazione
  - Dettagli delle varie Clausole

La base di dati di esempio

- Sintesi dei ricoveri effettuati da un certo centro ospedaliero
- Analisi delle prestazioni effettuate e dei relativi costi
- Informazioni aggiuntive legate al centro

La base di dati di esempio



Lezione SQL 1

- Interrogazione di Tabelle (QUERY)
  - Cos’è una Query?
  - Sintassi dell’istruzione SELECT
  - Ordine Scrittura ≠ Ordine Elaborazione
  - Dettagli delle varie Clausole

Le QUERY

- Interrogazione dei dati in un database
- Espressioni di maggiore importanza in SQL
- Le query vengono realizzate grazie all'istruzione SELECT
- Il risultato di questa istruzione è sempre una tabella (volendo memorizzabile)

Lezione SQL 1

- Interrogazione di Tabelle (QUERY)
  - Cos’è una Query?
  - Sintassi dell’istruzione SELECT
  - Ordine Scrittura ≠ Ordine Elaborazione
  - Dettagli delle varie Clausole

Sintassi dell'istruzione SELECT

```
SELECT [DISTINCT | ALL ]  <nome_col1> [AS <nuovo_nome_col>]
                                {, <nome_col2> [AS <nuovo_nome_col>] }
   [INTO <nuovo_nome_tab>]

   FROM <nome_tab> {, <nome_tab2>}

  [WHERE <condizione>]

  [GROUP BY <lista_colonne> [HAVING <condizione>] ]

  [ORDER BY <nome_col> [ASC | DESC] {, <nome_col2> [ASC | DESC] }];
```

IMPORTANTE!!!

- ✓ L'ordine delle clausole è fissato: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY.
- ✓ SELECT e FROM obbligatorie
- ✓ HAVING può essere usato solo se viene usato GROUP BY

Esercizio: trovare le SELECT corrette

```
SELECT …
FROM …
ORDER BY …
```

```
SELECT …
WHERE …
ORDER BY …
```

```
SELECT …
FROM …
HAVING …
```

```
SELECT …
FROM …
GROUP BY …
HAVING …
```

```
SELECT …
WHERE …
GROUP BY …
```

```
SELECT …
WHERE …
FROM …
```

```
SELECT …
FROM …
WHERE …
```

```
SELECT …
ORDER BY …
FROM …
GROUP BY …
```

Esercizio: trovare le SELECT corrette

SELECT ...
FROM ...
ORDER BY ...

SELECT ...
WHERE ...
ORDER BY ...

SELECT ...
FROM ...
HAVING ...

SELECT ...
FROM ...
GROUP BY ...
HAVING ...

SELECT ...
WHERE ...
GROUP BY ...

SELECT ...
WHERE ...
FROM ...

SELECT ...
FROM ...
WHERE ...

SELECT ...
ORDER BY ...
FROM ...
GROUP BY ...

Sintassi dell'istruzione SELECT

```
SELECT [DISTINCT ] | [ALL ]  <nome_col1> [AS <nuovo_nome_col>]
                                    {, <nome_col2> [AS <nuovo_nome_col>] }
    [INTO <nuovo_nome_tab>]
    FROM <nome_tab> {, <nome_tab2>}
    [WHERE <condizione>]
    [GROUP BY <lista_colonne> [HAVING <condizione>] ]
    [ORDER BY <nome_col>[ASC | DESC]  {, <nome_col2> [ASC | DESC] }];
```

- ALL è il default. DISTINCT è importante da usare se non si vogliono creare righe uguali in una nuova tabella
- INTO va usato per creare una nuova tabella (usare DISTINCT!!!!)
- AS serve a rinominare le colonne
- ASC e DESC servono a specificare l'ordine crescente (default) o decrescente

Lezione SQL 1

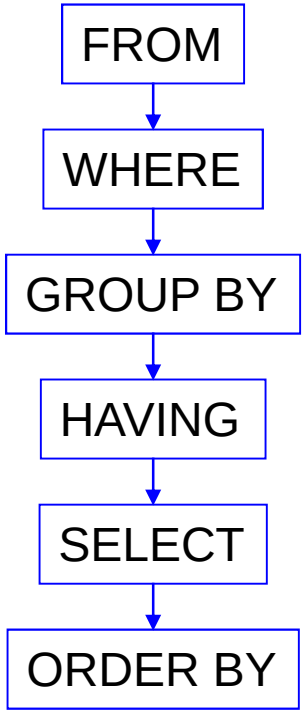
- Interrogazione di Tabelle (QUERY)
  - Cos’è una Query?
  - Sintassi dell’istruzione SELECT
  - Ordine Scrittura ≠ Ordine Elaborazione
  - Dettagli delle varie Clausole

L’ordine delle clausole SELECT

- Ordine di scrittura:

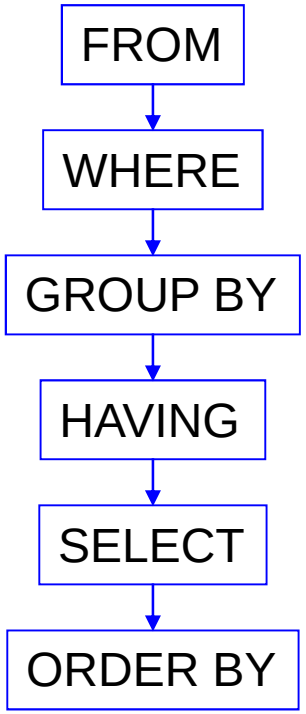
✓ SELECT
✓ FROM
✓ WHERE
✓ GROUP BY
✓ HAVING
✓ ORDER BY

- Ordine di elaborazione:



Interrogazioni in SQL

- Combinando in modo opportuno gli operatori dell'istruzione SELECT possiamo compiere **interrogazioni** sulla base di dati



1. Quali tabelle occorrono?

2. È necessaria una condizione per la selezione delle righe?

3. È necessario raggruppare le righe?

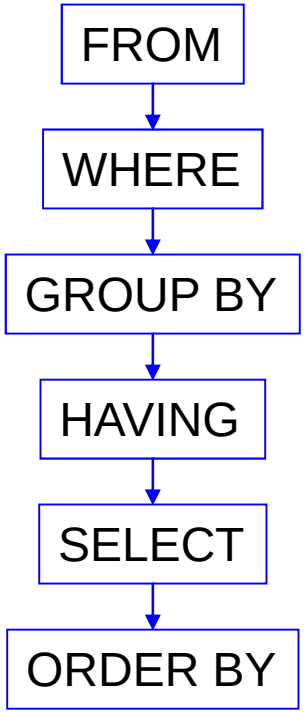
4. È necessaria una condizione sui raggruppamenti?

5. Quali colonne devono apparire sulla tabella di output?

6. È necessario ordinare le righe nella tabella risultato?

Interrogazioni in SQL

Voglio la lista ordinata dei rimborsi corrispondenti ai diversi drg, che siano maggiori di 1000 Euro e che siano presenti più di una volta nella tabella.



1. Quali tabelle occorrono? → drg

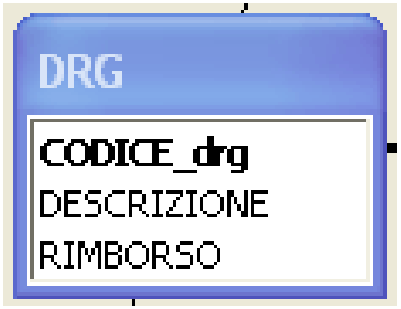
2. È necessaria una condizione per la selezione delle righe? → Sì: rimborso>1000

3. È necessario raggruppare le righe? → Sì: devo contare quante volte appare ogni rimborso

4. È necessaria una condizione sui raggruppamenti? → Sì: il gruppo deve contenere più di un elemento

5. Quali colonne devono apparire sulla tabella di output? → rimborso

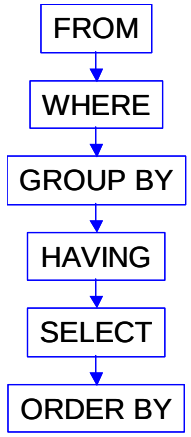
6. È necessario ordinare le righe nella tabella risultato? → Sì: voglio una lista ordinata




```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso
HAVING COUNT(*)>1
ORDER BY rimborso;
```

Esempio istruzione SELECT

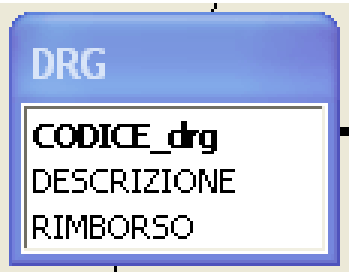
```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```



DRG		
codice_drg	**descrizione**	**rimborso**
3	asma	500
4	attacco ischemico transitorio	1200
7	frattura	500
6	infarto	1000
1	interventi midollo spinale	4000
8	interventi sulla retina	4000
2	interventi vasi extracranici	5000
9	rinoplastica	1200
5	ustioni	1200

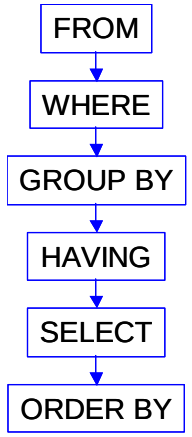
FROM drg

Crea una copia interna della tabella drg



Esempio istruzione SELECT

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```



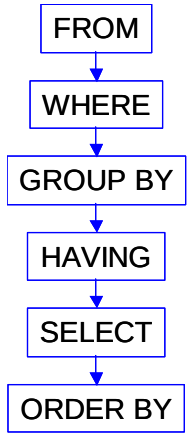
DRG		
codice_drg	**descrizione**	**rimborso**
4	attacco ischemico transitorio	1200
1	interventi midollo spinale	4000
8	interventi sulla retina	4000
2	interventi vasi extracranici	5000
9	rinoplastica	1200
5	ustioni	1200

WHERE rimborso>1000

Seleziona le righe che soddisfano la condizione (rimborso>1000)

Esempio istruzione SELECT

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```



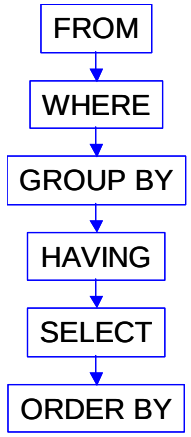
DRG		
codice_drg	**descrizione**	**rimborso**
1	interventi midollo spinale	4000
8	interventi sulla retina	4000
2	interventi vasi extracranici	5000
9	rinoplastica	1200
5	ustioni	1200
4	attacco ischemico transitorio	1200

GROUP BY rimborso

Raggruppa le righe in base a valori uguali nella colonna ‘rimborso’

Esempio istruzione SELECT

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```



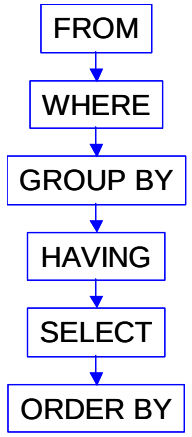
DRG		
codice_drg	**descrizione**	**rimborso**
1	interventi midollo spinale	4000
8	interventi sulla retina	4000
9	rinoplastica	1200
5	ustioni	1200
4	attacco ischemico transitorio	1200

HAVING COUNT(*)>1

Seleziona i gruppi che soddisfano la condizione di avere più di un elemento al loro interno

Esempio istruzione SELECT

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```



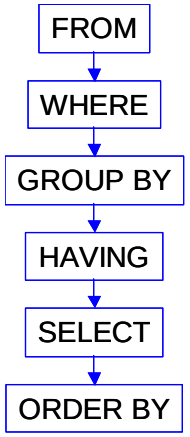
rimborso
4000
1200

SELECT rimborso

Seleziona le colonne che vedrò nella tabella di output della query

Esempio istruzione SELECT

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```



rimborso
1200
4000

ORDER BY rimborso

Ordina le righe della tabella di output in base ai valori contenuti in ‘rimborso’

Lezione SQL 2

- Interrogazione di Tabelle (QUERY)
  - Cos’è una Query?
  - Sintassi dell’istruzione SELECT
  - Ordine Scrittura ≠ Ordine Elaborazione
  - Dettagli delle varie Clausole

SELECT	FROM	WHERE	GROUP BY	HAVING	ORDER BY

Clausola SELECT

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```

Seleziona le colonne che vedrò nella tabella di output della query

SQL permette di fare *elaborazioni sui dati dopo la parola chiave SELECT*:

```
SELECT rimborso/1000 FROM drg ;
SELECT round(rimborso/1000) FROM drg ;
```

…

SELECT	FROM	WHERE	GROUP BY	HAVING	ORDER BY

Funzioni di aggregazione

Dopo la parola chiave SELECT è possibile fare *elaborazioni sui dati*:

In particolare esistono le **funzioni di aggregazione** che hanno come input i valori contenuti in una colonna e come output un unico valore.

- COUNT ([DISTINCT] <nome_colonna>)
- SUM ([DISTINCT] <nome_colonna>)
- AVG ([DISTINCT] <nome_colonna>)
- MAX (<nome_colonna>)
- MIN (<nome_colonna>)

- N.B. I valori NULL non vengono considerati!!!!

Funzioni di aggregazione

Le **funzioni di aggregazione** hanno come input i valori contenuti in una colonna e come output un unico valore.
Vanno scritte tra SELECT e FROM.
L'identificatore di colonna dopo SELECT deve essere la funzione aggregato stessa!!!!!

```
SELECT MAX(rimborso)
FROM drg;
```

Restituisce un solo valore (una sola riga) nella tabella di output.

```
SELECT descrizione, MAX(rimborso)
FROM drg;
```

Più valori

1 valore

ERRORE!!!!!

SELECT	FROM	WHERE	GROUP BY	HAVING	ORDER BY

Funzioni di aggregazione

Le **funzioni di aggregazione** hanno come input i valori contenuti in una colonna e come output un unico valore.
Vanno scritte tra SELECT e FROM.
L'identificatore di colonna dopo SELECT deve essere la funzione aggregato stessa!!!!!

```
SELECT descrizione, MAX(rimborso)
FROM drg;
```

Più valori

1 valore

ERRORE!!!!!

```
SELECT descrizione,rimborso
FROM drg
WHERE rimborso=      Valore massimo di rimborso      ;
```

CORRETTO

Funzioni di aggregazione

Le **funzioni di aggregazione** hanno come input i valori contenuti in una colonna e come output un unico valore.
Vanno scritte tra SELECT e FROM.
L'identificatore di colonna dopo SELECT deve essere la funzione aggregato stessa!!!!!

```
SELECT descrizione, MAX(rimborso)
FROM drg;
```

Più valori

1 valore

ERRORE!!!!!

```
SELECT descrizione,rimborso
FROM drg
WHERE rimborso= SELECT MAX(rimborso) FROM drg);
```

Valore massimo di rimborso

CORRETTO

QUERY ANNIDATA

La Funzione COUNT

COUNT prevede un uso speciale: COUNT(*)
che serve per contare le righe presenti nell'aggregato, compresi i NULL e i duplicati (quindi non posso usare DISTINCT se uso *)

Esempi:

In ACCESS, non posso usare DISTINCT con le funzioni di aggregazione

```
SELECT COUNT (DISTINCT drg)
FROM ricoveri;
```

Conta il numero di drg diversi nei ricoveri effettuati

```
SELECT COUNT (*)
FROM ricoveri;
```

Conta il numero di ricoveri, quindi il numero di righe nella tabella ricoveri

```
SELECT COUNT (*)
FROM ricoveri
WHERE reparto=1;
```

Conta il numero di ricoveri effettuati nel reparto 1

La Clausola WHERE

- Impone una o più condizioni sulle righe da selezionare
- Agisce attraverso
  - **Operatori** (di uguaglianza/diseguaglianza, logici, ecc…), che agiscono direttamente sulle righe della tabella specificata
  - **Predicati** che hanno come argomento una istruzione SELECT (ovvero una tabella “virtuale”)
- Gli operatori e i predicati danno come risultato TRUE/FALSE per ciascuna riga

Gli operatori nella WHERE (1)

- SELECT nome,cognome,nascita
  FROM generale
  WHERE provincia=“PD”;

- SELECT nome,cognome,nascita
  FROM generale
  WHERE provincia=“PD” AND sesso=“F”;

Gli operatori nella WHERE (2)

- =
- <
- >
- <=
- >=
- <>

Operatori di uguaglianza/ disuguaglianza

- NOT
- AND
- OR

Operatori logici

- IS NULL

  SELECT cognome, nome FROM generale
  WHERE nascita IS NULL;

Gli operatori nella WHERE (3)

- La clausola WHERE considera il test logico ad essa associato soddisfatto, quando il risultato del test è TRUE

- Attenzione!!!!
  - TRUE AND NULL = NULL
  - FALSE AND NULL = FALSE
  - TRUE OR NULL = TRUE
  - FALSE OR NULL = NULL

Gli operatori nella WHERE (4)

- BETWEEN:
  a BETWEEN b AND c ha il significato b <= a <= c

```
SELECT nome, cognome, nascita FROM generale
WHERE nascita BETWEEN #01/01/1960# AND #01/01/1964#;
```

- IN: cerca in un elenco

```
SELECT nome, cognome, provincia FROM generale
WHERE provincia  IN (“RO”,”VI”);
```

- LIKE: si usa per la ricerca di stringhe

```
SELECT nome, cognome, nascita FROM generale
WHERE comune LIKE “B*”;
```

I predicati nella WHERE

- È possibile annidare le query, ovvero imporre la condizione WHERE su l'output di una query annidata.

- I predicati della WHERE sono delle funzioni che hanno come argomento una istruzione SELECT (ovvero una tabella "virtuale").

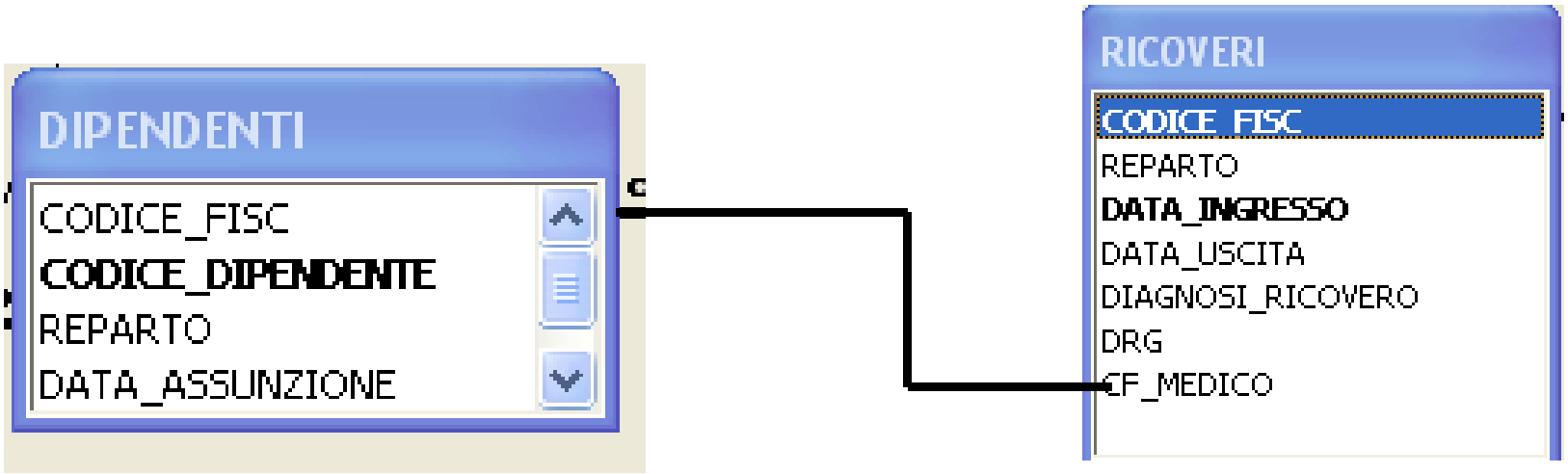
  PREDICATO(tabella_virtuale)→{TRUE, FALSE}

I predicati nella WHERE (1)

- IN: Controlla che il valore sia compreso tra quelli restituiti dalla subquery (SELECT).

Esempio: quali dipendenti sono stati ricoverati?

```
SELECT codice_fisc, reparto
FROM dipendenti
WHERE codice_fisc IN (SELECT codice_fisc FROM
ricoveri);
```

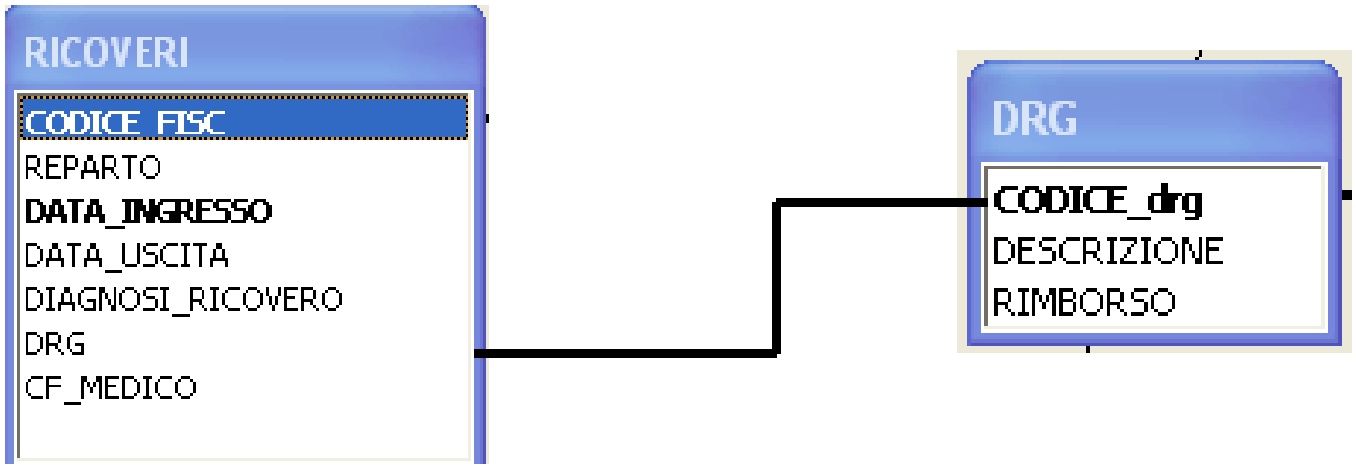


I predicati nella WHERE (2)

- EXISTS: controlla se il predicato restituisce righe (TRUE), altrimenti assume il valore FALSE

Esempio: casi di drg verificati nei ricoveri effettivi?

```
SELECT DISTINCT drg
    FROM  ricoveri
    WHERE  EXISTS (SELECT  ricoveri.drg FROM ricoveri, drg
    WHERE ricoveri.drg=drg.codice_drg);
```



I predicati nella WHERE (2)

- EXISTS: controlla se il predicato restituisce righe (TRUE), altrimenti assume il valore FALSE

Esempio: casi di drg verificati nei ricoveri effettivi?

```
SELECT DISTINCT drg
    FROM  ricoveri
    WHERE  EXISTS (SELECT  ricoveri.drg FROM ricoveri, drg
    WHERE ricoveri.drg=drg.codice_drg);
```

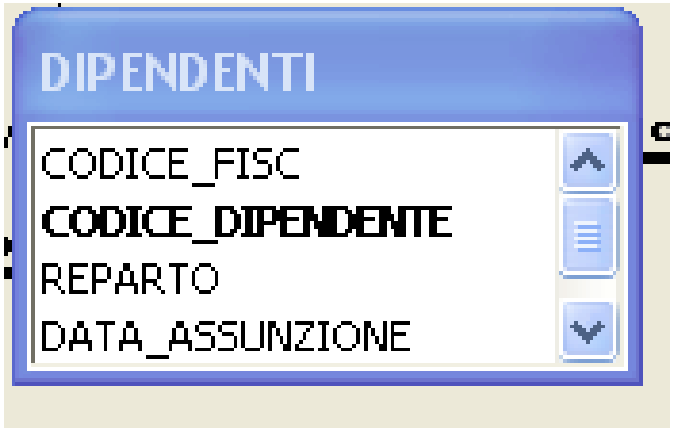
Specifica “full”: tabella e attributo separati da punto
Evita ambiguità e rende più facile la lettura!

I predicati nella WHERE (3)

- ALL
  Esempio: troviamo la persona assunta più di recente:

```
SELECT codice_fisc, reparto
FROM dipendenti
WHERE data_assunzione >= ALL (SELECT
data_assunzione FROM dipendenti);
```

- ANY viceversa è soddisfatto quando almeno un elemento della lista (SELECT) verifica la condizione



Accorgimenti sulle subquery

- Una subquery deve essere sempre racchiusa fra una coppia di parentesi
- Ciascuna subquery deve restituire una tabella dotata di una sola colonna compatibile con l'istruzione
- Tutte le colonne devono essere ben specificate (colonne fully qualified)
- Non esiste un limite di annidamento

SELECT	FROM	**WHERE**	GROUP BY	HAVING	ORDER BY

Esempio

Nome e Cognome dei pazienti ricoverati nel reparto di medicina interna

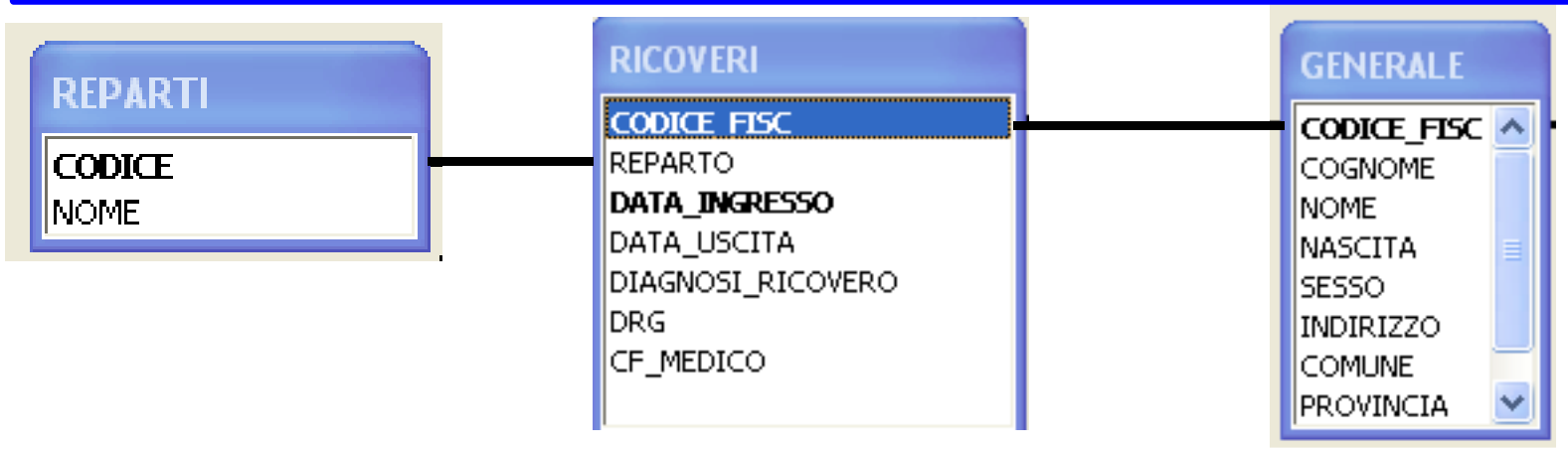
1. Codice del reparto ‘medicina interna’
2. CF dei pazienti ricoverati a medicina interna
3. Nome e cognome dei pazienti con CF ricavato al punto 2

ORDINE LOGICO

ORDINE SCRITTURA

```
SELECT nome, cognome FROM generale WHERE codice_fisc IN
(SELECT codice_fisc FROM ricoveri WHERE reparto = (
SELECT codice FROM reparti WHERE nome='medicina interna'
)
);
```

IMPORTANTE!!!
Ogni subquery DEVE restituire UNA SOLA colonna!!!

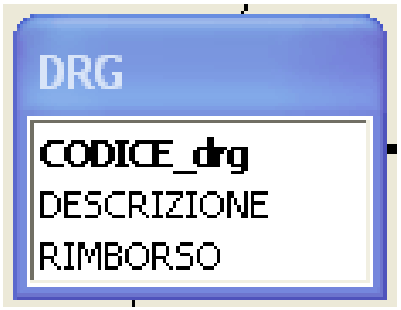


La Clausola GROUP BY

- Raggruppa le righe derivate dall'applicazione delle clausole FROM e WHERE suddividendole in base ai valori della colonna specificata.

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso
HAVING COUNT(*)>1
ORDER BY rimborso;
```

- Se ci sono dei campi NULL in rimborso, tutti i NULL vengono raggruppati



IMPORTANTE!!!

Gli argomenti che stanno qui devono essere:

- ✓ **un sottoinsieme degli argomenti di GROUP BY**
- ✓ **funzioni di aggregazione (SUM, AVG, MAX, MIN, COUNT), che vengono applicate separatemente a ciascun raggruppamento**

Esempio SELECT con GROUP BY

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso
HAVING COUNT(*)>1
ORDER BY rimborso;
```

L'unica colonna selezionabile, compatibilmente a GROUP BY è rimborso

DRG		
codice_drg	**descrizione**	**rimborso**
1	interventi midollo spinale	4000
8	interventi sulla retina	4000
2	interventi vasi extracranici	5000
9	rinoplastica	1200
5	ustioni	1200
4	attacco ischemico transitorio	1200

GROUP BY rimborso

Raggruppa le righe in base a valori uguali nella colonna 'rimborso'

Se specifico più di un attributo devo avere valori uguali su tutti gli attributi specificati

Esempio SELECT con GROUP BY

```
SELECT rimborso, descrizione
FROM drg
WHERE rimborso>1000
GROUP BY rimborso;
```

3 righe

6 righe

ERRATO!!!!

Invece avrei potuto usare:

```
SELECT rimborso, COUNT(*) AS numero_occorrenze
FROM drg
WHERE rimborso>1000
GROUP BY rimborso;
```

✓**funzioni di aggregazione (SUM, AVG, MAX, MIN, COUNT), che vengono applicate separatamente a ciascun raggruppamento**

DRG

codice_drg	descrizione	rimborso
1	interventi midollo spinale	4000
8	interventi sulla retina	4000
2	interventi vasi extracranici	5000
9	rinoplastica	1200
5	ustioni	1200
4	attacco ischemico transitorio	1200

→

TABELLA DI OUTPUT

rimborso	numero_occorrenze
4000	2
5000	1
1200	3

La Clausola HAVING

- E' simile alla WHERE, ma invece di agire sulle righe originali, agisce sulle righe aggregate tramite GROUP BY.
- Esempio: seleziona tutti i gruppi con un numero di elementi maggiore di uno

```
SELECT rimborso
FROM drg
WHERE rimborso>1000
GROUP BY rimborso HAVING COUNT(*)>1
ORDER BY rimborso;
```

rimborso
4000
1200

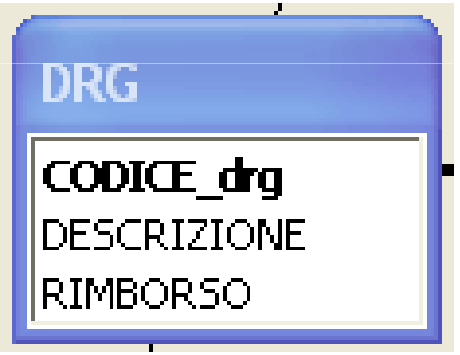
Gli argomenti che stanno qui devono essere:
- ✓ **un sottoinsieme degli argomenti di GROUP BY**
- ✓ **funzioni di aggregazione (SUM, AVG, MAX, MIN, COUNT), che vengono applicate separatemente a ciascun raggruppamento**

La clausola ORDER BY

- Consente di ordinare i record presentati nella tabella dei risultati secondo un ordinamento multi-livello

```
SELECT descrizione,rimborso
FROM drg
ORDER BY rimborso ASC;
```

- ASC è il dafault



Esempio:

```
SELECT descrizione,rimborso FROM drg
ORDER BY rimborso DESC, descrizione;
```