INTRODUZIONE AD SQL
(CAPITOLO 4)

R. Basili

a.a. 2013-2014

SQL

Dopo aver eseguito l'installazione del DBMS e di uno o più database:

	Oracle (7-9i)	MySQL (3.23-4.15)
Accesso al DBMS	> sqlplus nome_utente/pwd	> mysql -u[nome_user] -p[pwd] > show databases; > use [nome_database]
Interpretazione di uno script	SQL> @nome file	> mysql -u[nome_user] -p[pwd] [nome_db] <[script.sql]
Descrizione dello schema di una tabella	SQL> desc nome_tbl	> show create table nome_tbl;
Esecuzione Comando di SO	SQL>!cmd Unix/Linux SQL> $cmd Dos	

L'espressione SELECT in SQL

- La struttura base di una interrogazione in SQL e' la seguente:

```
SELECT <Attribute List>
FROM <Relation List>
WHERE <Condition>;
```

- Ad es:
  - `SELECT * FROM STUDENTI;`
  - `SELECT age from SAILORS;` (*)

  - (*) SQL non e' case sensitive.

Select(1) – semantica

```
select [DISTINCT] <target_list>
from <relation_list>
[ where <qualification> ]
[ order by <column(s) [asc\desc]> ]
```

Semantica

- Eseguire il prodotto cartesiano della `relation_list`
- Scartare le tuple che non soddisfano le condizioni della `qualification`
- Rimuovere gli attributi che non sono presenti nella `target_list`
- Se e' specificata la `DISTINCT` rimuovere le tuple duplicate

WHERE clause

- Es
  - `SELECT * FROM STUDENTI WHERE cognome='Rossi'`
  - `SELECT * FROM STUDENTI WHERE cognome=Rossi`
  -
```
SELECT * FROM STUDENTI
    WHERE Cognome='Rossi' AND
                (Nome='Paolo' OR Nome='Mario')
```
  - `SELECT * FROM Studenti ORDER BY Cognome`
  - `SELECT * FROM Studenti ORDER BY Cognome DESC`

Operatori

Operatore	Descrizione
=	Uguaglianza
<>	Not Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	Membership in un insieme di valori

Operatori IN, BETWEEN

- ```
  SELECT * FROM STUDENTI
  WHERE COGNOME IN (‘Rossi',‘Basili')
  ```
- ```
  SELECT * FROM STUDENTI
  WHERE Cognome NOT IN (‘Abi',‘Verdi')
  ```
- ```
  SELECT * FROM STUDENTI
  WHERE Cognome NOT BETWEEN
            ‘Abate' AND ‘Babbabei'
  ```
  ```

Alias con Variabili di RANGE

- ```
  SELECT Ord_Prodotto.OrderID,
       Persona.Cogn, Persona.Nome
       FROM Persona,
            Ord_Prodotto
       WHERE Persona.Cogn='Rosi' AND
              Persona.Nome='Paolo' AND
              Ord_Prodotto.CF=Persona.CF
  ```

- ```
  SELECT OP.OrderID, P.Cogn, P.Nome
       FROM Persona AS P, Ord_Prodotto AS OP
       WHERE P.Cogn='Rosi' AND P.Nome='Paolo'
                              AND OP.CF=P.CF
  ```

JOIN in SQL

- La query:
  - Trovare tutti i codici dei corsi seguiti da uno studente *X Y*
- E' espressa dalla seguente relazione

$$\pi_{CID}(\sigma_{X,Y}\text{Studente} \rhd\lhd \text{Segue})$$

- In SQL
  - SELECT Se.CID
    FROM Studente AS St, Segue AS Se
    WHERE St.Cogn='X' AND St.Nome='Y'
    AND St.StudID=Se.StudID

Tipi di JOIN

- JOIN:
  - Restituisce le righe per cui esiste almeno un match in entrambe le tabelle
- LEFT JOIN:
  - Restituisce tutte le tuple dalla tabella di sinistra anche se non ci sono match in quella destra
- RIGHT JOIN:
  - Restituisce tutte le tuple dalla tabella di destra anche se non ci sono match in quella sinistra

Operatore UNION

```
Select  S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='rossa'
UNION
Select S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
      AND B.color='verde'
```

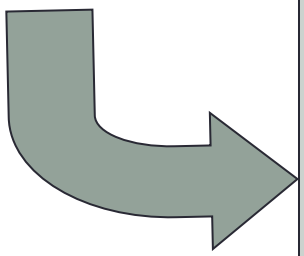
Select- qualification where …

Nested Query:

<column> [not] ***in*** *(<query>)*

oppure

EXISTS *(<query>)*



```
Select  S.sname
FROM Sailors S
WHERE S.sid IN
   (Select R.sid
    FROM Reserves R
    WHERE R.bid=103
              AND S.Sid= R.sid)
```

Null value:

<column> ***is [not] null,***

Domain conditions:

<column> ***[not] between*** *<lower bound> and <upper bound>*

String conditions:

<column> LIKE '<string oppure espr con % o _>'

Q1 e Q2

Trova il nome dei marinai (tabella S) che hanno riservato la barca numero 103.

```
SELECT S.sname
FROM Sailors S,Reserves R
WHERE S.sid=R.sid AND R.bid=103
```

Trova il nome dei marinai (tabella S) che hanno riservato una barca rossa.

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.color='rossa'
```

Q4 e Q5

*Trova il nome dei marinai (tabella S) che hanno riservato una barca rossa **oppure** una barca verde.*

```
SELECT S.sname
FROM Sailors S, Reserves R, B Boats
WHERE S.sid=R.sid AND R.bid = B.bid
AND  (B.color='rossa' OR B.color='verde')
```

*Trova il nome dei marinai (tabella S) che hanno riservato una barca rossa **ed** una barca verde.*

```
SELECT S.sid, S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='rossa'
INTERSECT
SELECT S2.sid, S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid=R2.sid AND R2.bid=B2.bid
      AND B2.color='verde'
```

Q9

Trova il nome dei marinai (tabella S) che hanno riservato ***tutte*** *le barche*

Q9

*Trova il nome dei marinai (tabella S) che hanno riservato **tutte** le barche*

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS ((SELECT B.bid FROM Boats B)
                    EXCEPT(SELECT R.bid
                     FROM Reserves R WHERE R.sid=S.sid))
```

oppure:

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid FROM Boats B
            WHERE NOT EXISTS (SELECT R.bid
                              FROM Reserves R
                              WHERE R.bid=B.bid
                                   AND R.sid=S.sid))
```

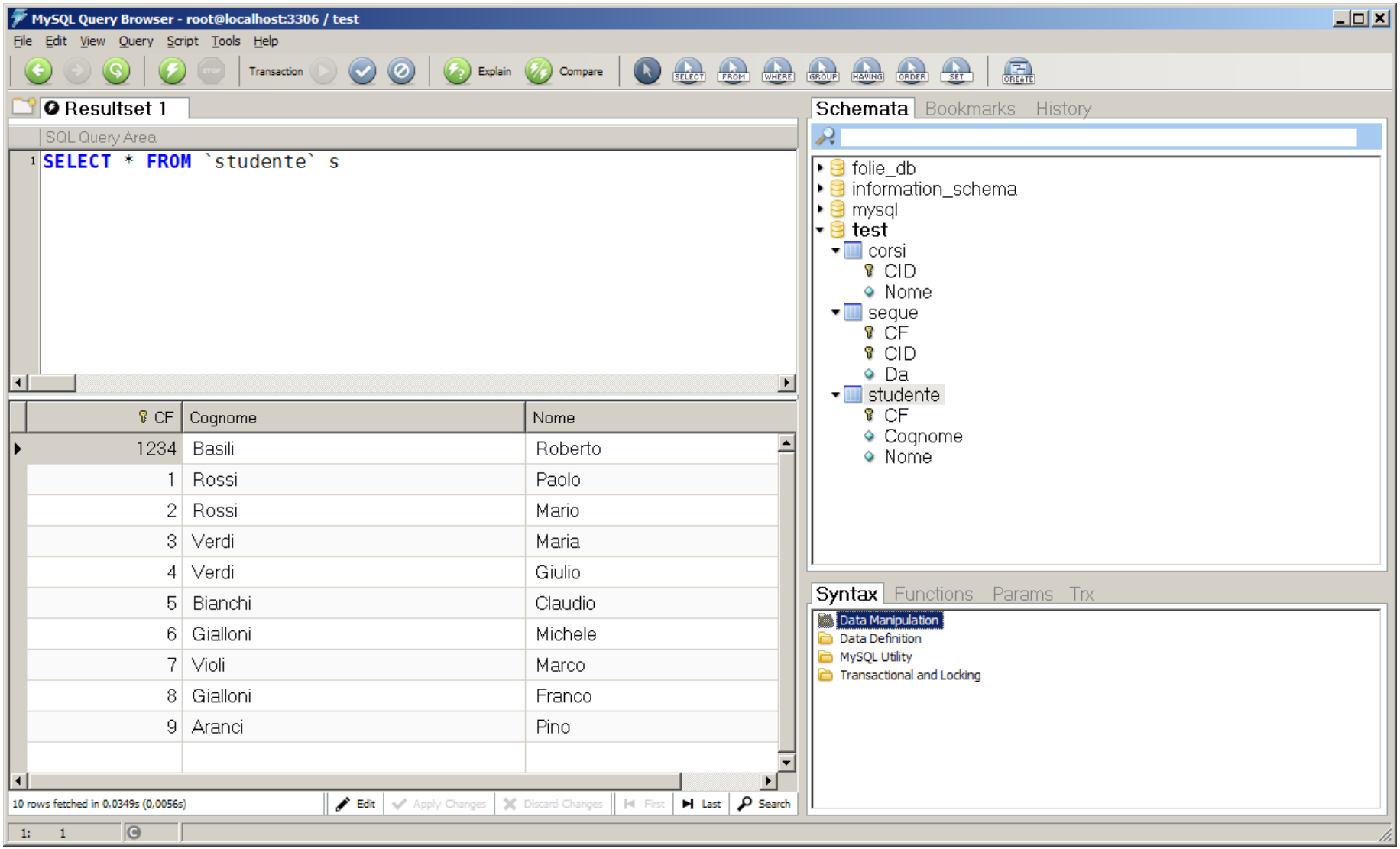
Esempio (da ER precedenti)

- Due entità: Studenti e Corsi
- Una relazione: Segue
- Studio di: JOIN, LEFT e RIGHT (OUTER) JOIN, Nested queries, Divisione

ENTITA': STUDENTE

```
-- Table structure for table `studente`
CREATE TABLE `studente` (
  `CF` int(10) unsigned NOT NULL auto_increment,
  `Cognome` varchar(45) NOT NULL,
  `Nome` varchar(45) NOT NULL,
  PRIMARY KEY  (`CF`)
);
-- Data for table `studente`
INSERT INTO `studente` VALUES
  (1234,'Basili','Roberto'),(1,'Rossi','Paolo'),(
  2,'Rossi','Mario'),(3,'Verdi','Maria'),(4,'Verd
  i','Giulio'),(5,'Bianchi','Claudio'),(6,'Giallo
  ni','Michele'),(7,'Violi','Marco');
```

MySQL Query Bowser

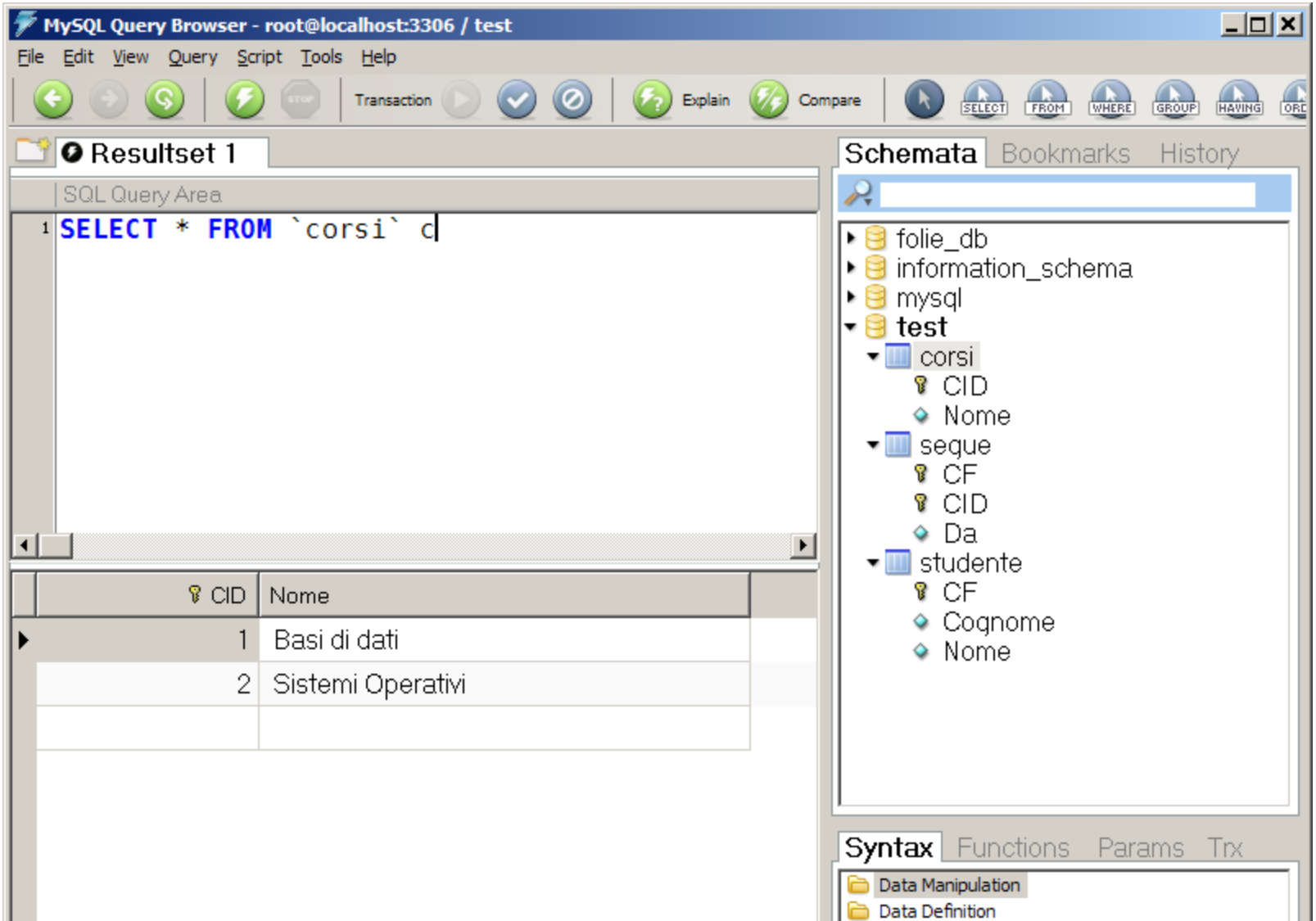


ENTITA': Corsi

```
--
DROP TABLE IF EXISTS `corsi`;
CREATE TABLE `corsi` (
  `CID` int(10) unsigned NOT NULL auto_increment,
  `Nome` varchar(45) NOT NULL,
  PRIMARY KEY  (`CID`)
);

INSERT INTO `corsi` VALUES (1,'Basi di
 dati'),(2,'Sistemi Operativi');
```

Istanza di “Corsi”

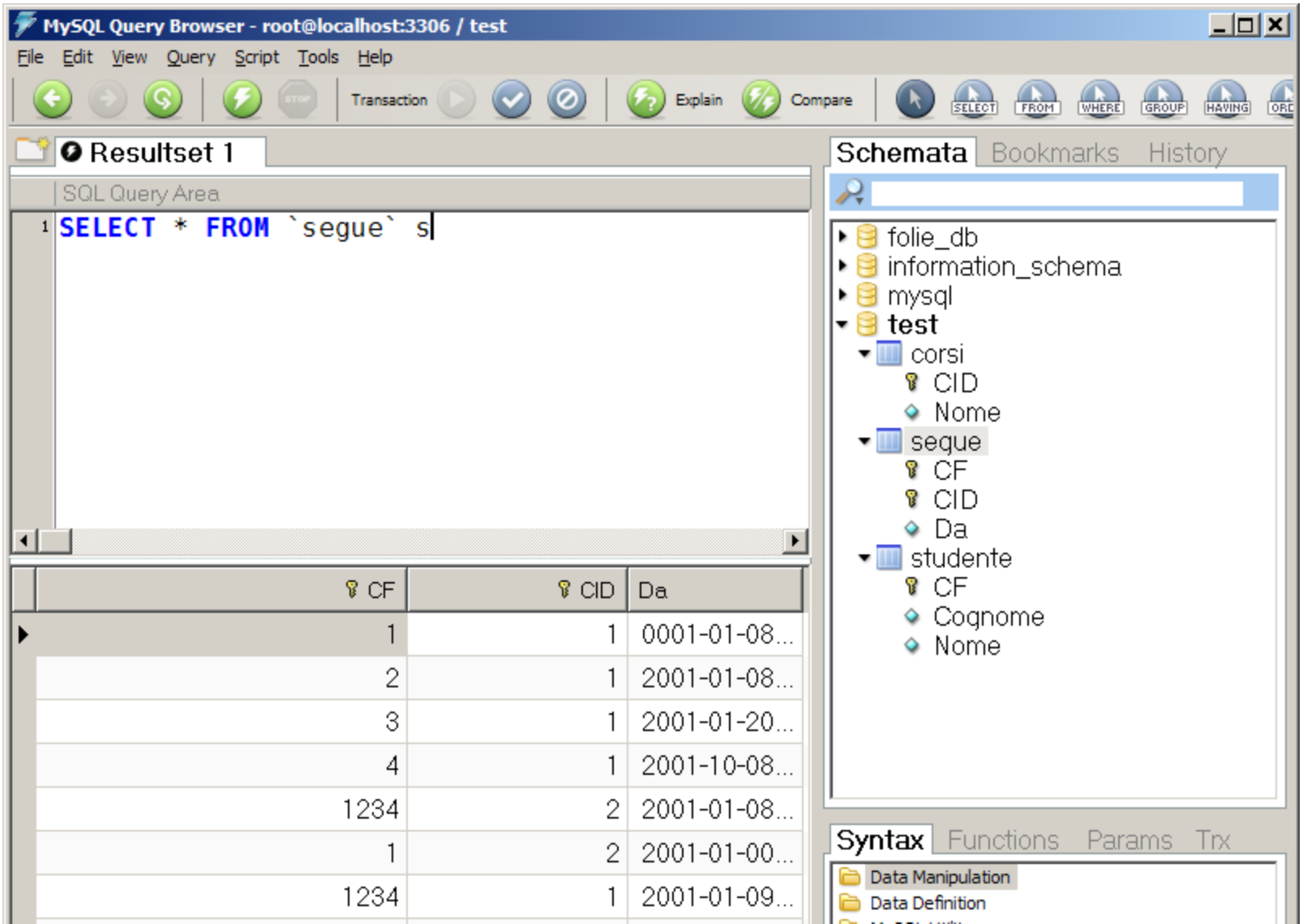


Relazione: Segue

```
-- -- Table structure for table `segue`

CREATE TABLE `segue` (
  `CF` int(10) unsigned NOT NULL auto_increment,
  `CID` int(10) unsigned NOT NULL,
  `Da` datetime NOT NULL,
  PRIMARY KEY  (`CF`,`CID`)
);
-- Dumping data for table `segue`
INSERT INTO `segue` VALUES
  (1,1,'0001-01-08'),(2,1,'2001-01-08'),
  (3,1,'2001-01-20 '),(4,1,'2001-10-08'),
  (1234,2,'2001-01-08 '),(1,2,'2001-01-00'),
  (1234,1,'2001-01-09');
```

MySQL Query Browser



Interrogazioni: JOIN

```
--- QUERY:
/* JOIN senza operatore */
SELECT Se.CID, St.CF, St.Cognome, St.Nome
FROM Studente AS St, Segue AS Se
WHERE (St.Cognome='Rossi' AND
                              St.CF=Se.CF);
```

Interrogazioni: JOIN (2)

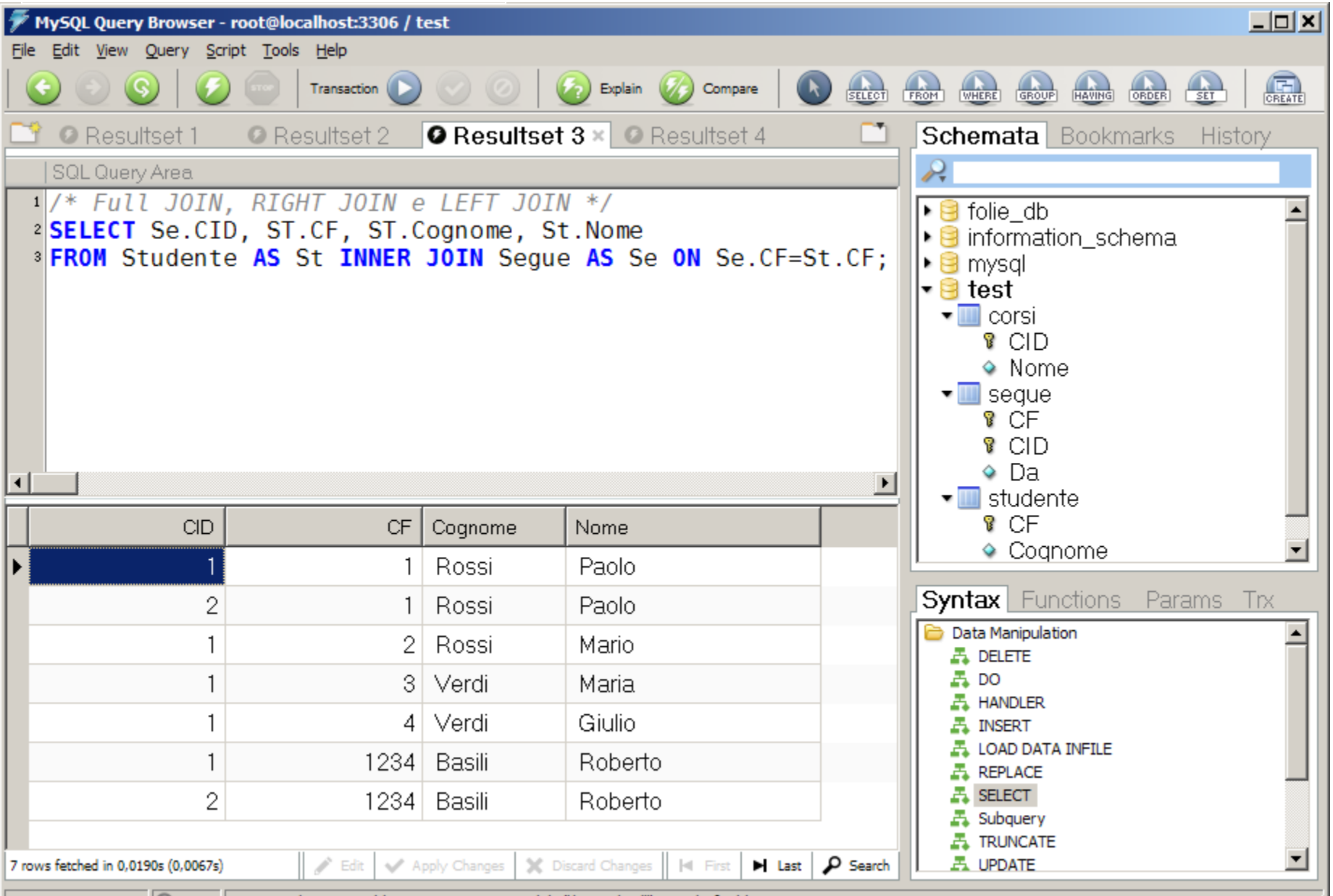
```
/* operatore JOIN con condizione */
SELECT *
FROM Studente AS St
JOIN Segue AS Se ON St.CF=Se.CF
WHERE St.Cognome='Rossi';
```

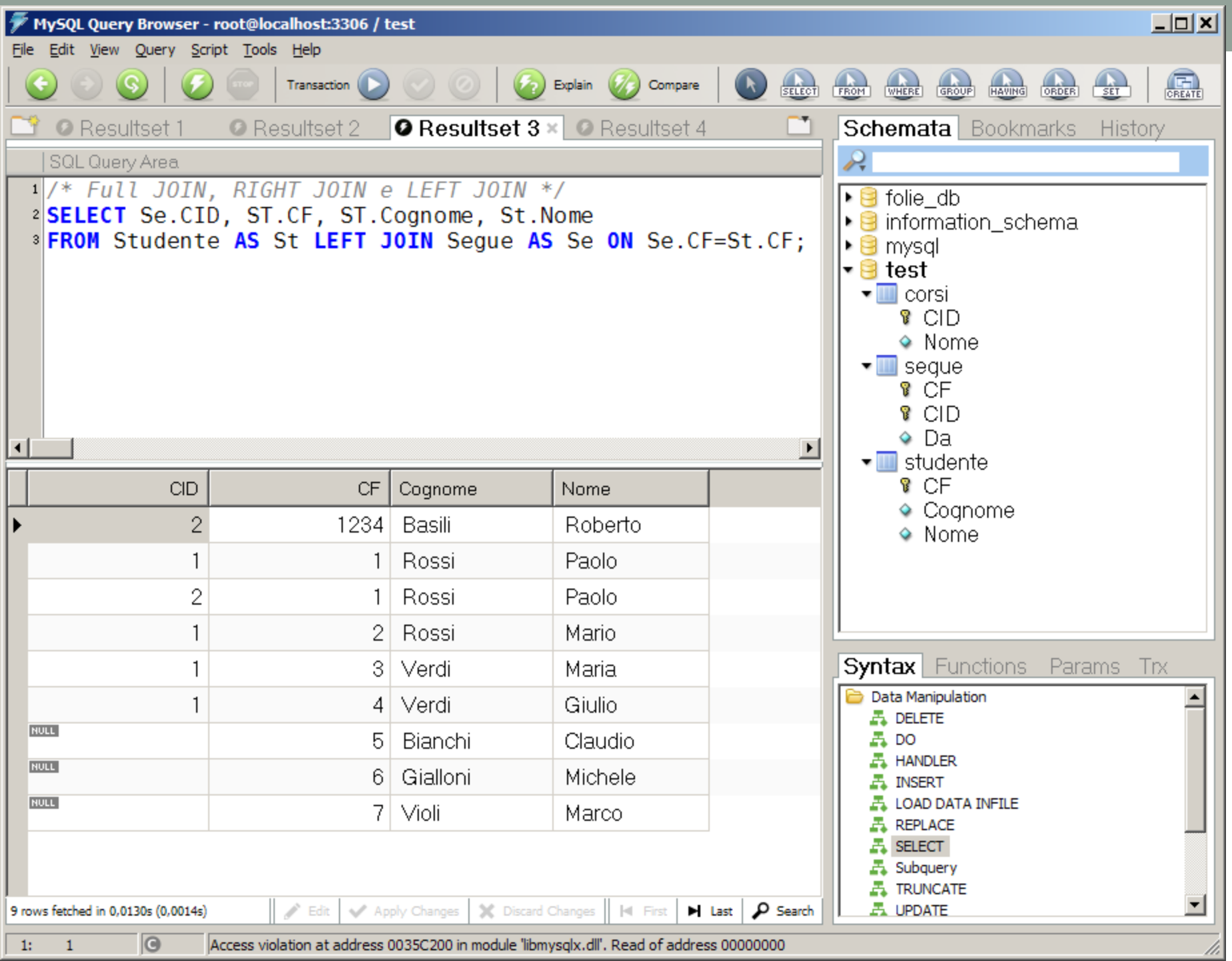
Interrogazioni: OUTER JOIN

```
/* Full JOIN, RIGHT JOIN e LEFT JOIN */
SELECT Se.CID, ST.CF, ST.Cognome, St.Nome
FROM Studente AS St
JOIN Segue AS Se ON Se.CF=St.CF;

SELECT Se.CID, ST.CF, ST.Cognome, St.Nome
FROM Studente AS St
LEFT JOIN Segue AS Se ON Se.CF=St.CF;
```

Join


MySQL Query Browser - root@localhost:3306 / test
File Edit View Query Script Tools Help
Transaction
Explain
Compare
SELECT FROM WHERE GROUP HAVING ORDER SET CREATE
Resultset 1
Resultset 2
Resultset 3
Resultset 4
SQL Query Area
1 /* Full JOIN, RIGHT JOIN e LEFT JOIN */
2 SELECT Se.CID, ST.CF, ST.Cognome, St.Nome
3 FROM Studente AS St INNER JOIN Segue AS Se ON Se.CF=St.CF;
CID CF Cognome Nome
1 1 Rossi Paolo
2 1 Rossi Paolo
1 2 Rossi Mario
1 3 Verdi Maria
1 4 Verdi Giulio
1 1234 Basili Roberto
2 1234 Basili Roberto
7 rows fetched in 0,0190s (0,0067s)
Edit
Apply Changes
Discard Changes
First
Last
Search
Schemata
Bookmarks
History
folie_db
information_schema
mysql
test
corsi
CID
Nome
segue
CF
CID
Da
studente
CF
Cognome
Syntax
Functions
Params
Trx
Data Manipulation
DELETE
DO
HANDLER
INSERT
LOAD DATA INFILE
REPLACE
SELECT
Subquery
TRUNCATE
UPDATE


MySQL Query Browser - root@localhost:3306 / test
File Edit View Query Script Tools Help
Transaction
Explain
Compare
SELECT
FROM
WHERE
GROUP
HAVING
ORDER
SET
CREATE
Resultset 1
Resultset 2
Resultset 3
Resultset 4
SQL Query Area
1 /* Full JOIN, RIGHT JOIN e LEFT JOIN */
2 SELECT Se.CID, ST.CF, ST.Cognome, St.Nome
3 FROM Studente AS St LEFT JOIN Segue AS Se ON Se.CF=St.CF;
CID CF Cognome Nome
2 1234 Basili Roberto
1 1 Rossi Paolo
2 1 Rossi Paolo
1 2 Rossi Mario
1 3 Verdi Maria
1 4 Verdi Giulio
NULL 5 Bianchi Claudio
NULL 6 Gialloni Michele
NULL 7 Violi Marco
9 rows fetched in 0,0130s (0,0014s)
Edit
Apply Changes
Discard Changes
First
Last
Search
Schemata
Bookmarks
History
folie_db
information_schema
mysql
test
corsi
CID
Nome
segue
CF
CID
Da
studente
CF
Cognome
Nome
Syntax
Functions
Params
Trx
Data Manipulation
DELETE
DO
HANDLER
INSERT
LOAD DATA INFILE
REPLACE
SELECT
Subquery
TRUNCATE
UPDATE
1: 1
Access violation at address 0035C200 in module 'libmysqlx.dll'. Read of address 00000000

Interrogazioni: JOIN MULTIPLE

```
/* Multiple JOIN */
SELECT Se.CID, Co.Nome, ST.CF, ST.Cognome, St.Nome
FROM Studente AS St
left JOIN Segue AS Se ON Se.CF=St.CF
left JOIN Corsi As Co ON Co.CID = Se.CID;

SELECT Se.CID, Co.Nome, ST.CF, ST.Cognome, St.Nome
FROM Studente AS St
JOIN Segue AS Se ON Se.CF=St.CF
left JOIN Corsi As Co ON Co.CID = Se.CID;
```

Interrogazioni: QUERY ANNIDATE

```
/* Query ANNIDATE: Uso dell'operatore IN */
Select  St.Nome, St.Cognome, St.CF
FROM Studente St
WHERE St.CF IN (
    Select Se.CF
    FROM Segue Se, Corsi Co
    WHERE Co.Nome='Basi di Dati'
            AND Se.CID=Co.CID );
```

Operazione di Divisione

$$A/B = \pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$$

*Trova i marinai che hanno prenotato **tutte** le barche*

Pren/B={ ‹x› | ∃‹x,y›∈A ∀‹y›∈B }

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s3	p2

Pren

pno
p2

B1

pno
p2
p4

B2

Pren/B1 →

sno
s1
s3

Pren/B2 →

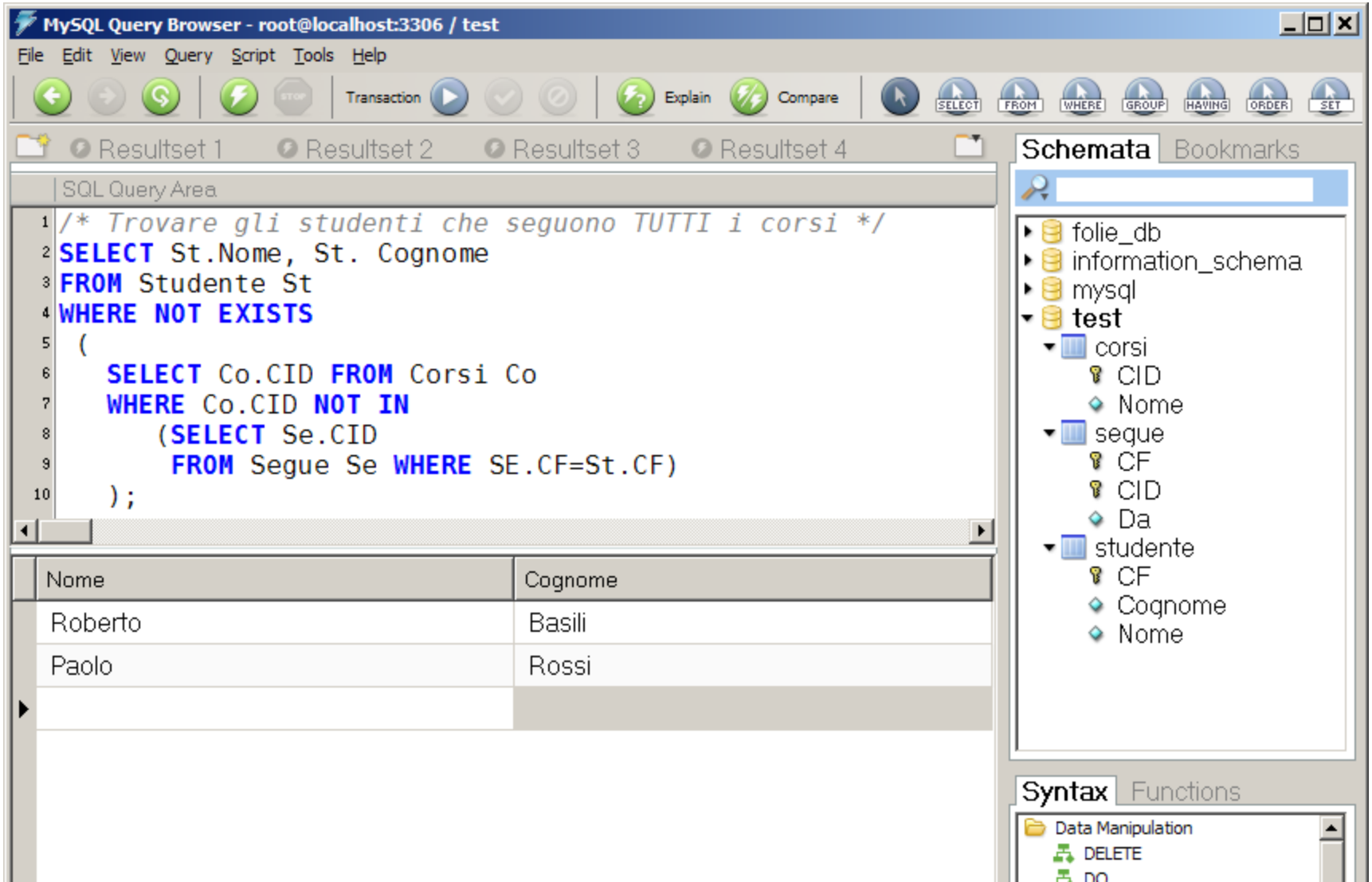
sno
s1

Interrogazioni: La divisione in SQL

```
/* Trovare gli studenti che seguono TUTTI i corsi */
SELECT St.Nome, St. Cognome
FROM Studente St
WHERE NOT EXISTS
 (
   SELECT Co.CID FROM Corsi Co
   WHERE Co.CID NOT IN
       (SELECT Se.CID
        FROM Segue Se WHERE SE.CF=St.CF)
   );
```

$$= \pi_{SID}(St) - \pi_{SID}((\pi_{SID}(Se) \times Co) - Se)$$

Divisione


MySQL Query Browser - root@localhost:3306 / test
File Edit View Query Script Tools Help
Transaction
Explain
Compare
SELECT FROM WHERE GROUP HAVING ORDER SET
Resultset 1
Resultset 2
Resultset 3
Resultset 4
SQL Query Area
/* Trovare gli studenti che seguono TUTTI i corsi */
SELECT St.Nome, St. Cognome
FROM Studente St
WHERE NOT EXISTS
(
SELECT Co.CID FROM Corsi Co
WHERE Co.CID NOT IN
(SELECT Se.CID
FROM Segue Se WHERE SE.CF=St.CF)
);
Nome
Cognome
Roberto
Basili
Paolo
Rossi
Schemata
Bookmarks
folie_db
information_schema
mysql
test
corsi
CID
Nome
segue
CF
CID
Da
studente
CF
Cognome
Nome
Syntax
Functions
Data Manipulation
DELETE

Operatori di Aggregazione

```
count(*|[DISTINCT]Attr)
sum([DISTINCT]Attr)
AVG ([DISTINCT]Attr)
MIN(A)
MAX(A)
```

non esprimibili in algebra relazionale!

1. **count(*|[DISTINCT]Attr)** restituisce il numero di righe count(*), con **Distinct** restituisce il numero di distinti valori di **Attr**
2. **sum([DISTINCT]Attr)** somma dei valori dell'attributo **Attr** su tutte le tuple della rel.
3. **avg([DISTINCT]Attr)** media dei valori dell'attributo **Attr** su tutte le tuple
4. **MAX(Attr)** massimo valore di **Attr** su tutte le tuple
5. **MIN(Attr)** minimo valore di **Attr** su tutte le tuple

Query con operazioni di aggregazione

Trova l'età massima dei marinai che hanno rating uguale a 10

```
SELECT MAX(S.age)
FROM Sailors S
WHERE S.rating=10
```

Trova il nome e l'età del marinaio più vecchio

```
SELECT S.sname, MAX(S.age)
FROM Sailors S
```

Query ILLEGALE!

- Quando in un argomento della select compaiono delle funzioni di aggregazione, allora non possono comparire **espressioni a livello di riga**, come ad esempio il nome di un attributo

```
SELECT S.sname, S.age
FROM Sailors S
WHERE (SELECT MAX (S2.age)
       FROM Sailors S2) = S.age
```

Clausole di GROUP BY ed HAVING

```
SELECT [DISTINCT]
  <target_list>
FROM <relation_list>
WHERE <qualification> ]
GROUP BY <grouping-list>
HAVING <group-qualification>
```

- Se non vogliamo applicare l'operazione di aggregazione a tutte le tuple che soddisfano la condizione di **`where`**, possiamo applicarla a sottoinsiemi (gruppi) di righe distinti in base al valore degli attributi presenti nella *grouping-list* della clausola **`GROUP BY`**.
- Possiamo inoltre imporre delle condizioni di selezione su questi gruppi specificandole nella clausola **`HAVING`**.

Valutazione Concettuale

```
SELECT [DISTINCT] <target_list>
FROM <relation_list>
WHERE <qualification> ]
GROUP BY <grouping-list>
HAVING <group-qualification>
```

Solamente attributi presenti nella group by oppure come argomenti di operazioni di aggregazione

1. Viene calcolato il prodotto cartesiano delle tabelle presenti nella ***relation_list***, vengono scartate le tuple che non soddisfano le condizioni di ***qualification***.
2. Divido secondo la ***grouping-list*** la relazione ottenuta dalla fase 1 in gruppi di tuple basandomi sulla ***gruoping-list***.
3. Applico la <***group-qualification***> per eliminare alcuni gruppi ed ottengo una tupla (risposta)per ogni gruppo

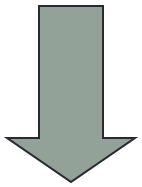
Q33

Per ogni barca rossa, trova il numero delle prenotazioni

```
SELECT B.bid,COUNT(*) AS numeroprenot
FROM Boats B, Reservations R
WHERE R.bid=B.bid
GROUP BY B.bid
HAVING B.color='rossa'
```

Query ILLEGALE!

E' rifiutata anche se il colore della barca e' unico per ogni gruppo (B.bid e' chiave)



```
SELECT B.bid,COUNT(*) AS numeroprenot
FROM Boats B, Reservations R
WHERE R.bid=B.bid AND B.color='rossa'
GROUP BY B.bid
```

Query CORRETTA

Trova l'età del più giovane marinaio maggiorenne per ogni livello di rating con almeno due di tali marinai

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1
```

Trova l'età del marinaio maggiorenne più giovane per ogni livello di rating con almeno due marinai (di ogni età)

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT(*)  FROM Sailors S2
                  WHERE S.rating=S2.rating)
```

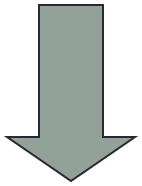
Q37

Trova i rating per i quali l'età media e' la minima su tutti i rating

```
SELECT S.rating
FROM Sailors S
WHERE S.age=(SELECT MIN(AVG(S2.age)) FROM Sailors S2)
```

Query ILLEGALE!

Le operazioni di aggregazione non possono essere annidate



Soluzione alternativa

```
SELECT Temp.rating, Temp.media
FROM (SELECT S.rating, AVG(S.age) AS media
            FROM Sailors S GROUP BY S.rating) AS Temp
WHERE Temp.media=(SELECT MIN(Temp.media) FROM Temp)
```

Query CORRETTA

Valori Nulli

In SQL si utilizza un valore chiamato ***null*** per indicare che il valore di un determinato campo e' non definito.

E' necessario definire una *logica a tre valori* (*true, false e null*) per definire il comportamento di alcune operazioni quando queste vengono applicate a campi ***null***

- Operatori di confronto (es:Attr>8)
- Operatori logici (***AND***, ***OR*** e ***NOT***)

La clausola ***WHERE*** deve essere rianalizzata alla luce dei possibili valori ***null***

- Deve essere definito un nuovo operatore

IS [NOT] NULL

per verificare quando un campo [non] ha valore *null.*

Considerazioni sul *null*

- La clausola **`WHERE`** fallisce su valori sconosciuto (in presenza di **`NULL`**) e le corrispondenti tuple vengon scartate
- **`COUNT(*)`** conta ANCHE i valori **`NULL`**
- **`SUM()`** e le altre op. di aggregazione agiscono sul sottoinsieme dei valori non **`NULL`**

Valori Nulli ed OUTER JOIN

La *logica a tre valori* (*true*, *false* e *null*) ci porta a definire un nuovo operatore l'**OUTER JOIN**

Considerando il join di due tabelle S $\rhd\lhd_c$ R, aggiungo nel risultato anche le tuple di S che non hanno corrispettivo in R, gli attributi di R vengono ugualmente inseriti nel risultato ma con valori *null*.

```
SELECT S.sid, R.bid
FROM Sailors S NATURAL LEFT OUTER JOIN Reserves R
```

sid	sname	rating	age
22	Davide	7	45.0
31	Luca	8	55.5
58	Remo	10	35.0

Sailors

sid	bid	day
22	101	10/10/03
58	103	11/12/02

Reserves

sid	bid
22	101
31	*null*
58	103

Table Constraints

Quando e' necessario specificare dei constraint complessi si utilizza la clausola CHECK <condizione>

```
CREATE TABLE Sailors ( sid     INTEGER,
                       sname   CHAR(10),
                       rating  INTEGER,
                       age     REAL,
                     PRIMARY KEY(sid),
                   CHECK (rating>=1 AND <=10))
```

oppure:

```
CREATE DOMAIN ratingVal INTEGER DEFAULT 1
            CHECK ( VALUE>=1 AND VALUE<=10)
```

DOMINI

E' possibile definire dei domini di valori dedicati dipendenti da specifico dominio applicativo utilizzando la clausola

CHECK <Condizione>

```
CREATE DOMAIN ValoriEx INTEGER DEFAULT 1
      CHECK (VALUE >=1 AND VALUE <=10))
```

Oss: Expertise ha tipo INTEGER e domini diversi di tipo INTEGER sono SEMPRE compatibili
Alternativa:

```
CREATE TYPE Expertise AS INTEGER
CREATE DOMAIN ValExp1 Expertise …
CREATE DOMAIN ValExp2 Expertise …
CREATE DOMAIN SAILOR_AGE INTEGER …
```

Asserzioni:Integrity Constraints su piu' tabelle

L'asserzione e' un IC definito separatamente dallo schema delle tabelle a cui fa riferimento e di cui definisce i vincoli di integrità

1

```
CREATE ASSERTION  SmallClub
CHECK ((SELECT COUNT(S.sid) FROM Sailors S)
          + (SELECT COUNT(B.bid) FROM Boats B) < 100)
```

2

```
CREATE TABLE Sailors (sid INTEGER, sname CHAR(10),
                                  rating INTEGER, age REAL
                                  PRIMARY KEY (sid),
 CHECK(rating >= 1 AND rating <= 10)
 CHECK((SELECT COUNT(S.sid) FROM Sailors S) +
          (SELECT COUNT(B.bid) FROM Boats B) < 100))
```

I Trigger

Un **trigger** è una procedura che viene attivata automaticamente a seguito di cambiamenti sul DBMS. Un database in cui sono stati definiti dei trigger si dice **ATTIVO**

I trigger seguono il paradigma **ECA**:

- **E**vento che attiva il trigger
- **C**ondizione che deve essere soddisfatta affinché il trigger venga eseguito
- **A**zione: le **operazioni** che esegue il trigger

Può essere visto come uno script daemon che monitorizza il database.

Esempio di trigger

EVENTO: Update della quota disponibile in magazzino
CONDIZIONE: controllo se la quantità disponibile e' inferiore alla quantità di riordino
AZIONE: emissione di un ordine altrimenti eccezione

```
CREATE TRIGGER gestione_magazzino
AFTER UPDATE OF qta-disp ON Magazzino
WHEN (NEW.qta-disp < NEW.qta-riord)
FOR EACH ROW
X EXCEPTION
BEGIN
IF NEW.QTA-DISP<0 THEN RAISE(X);
INSERT INTO RIORDINO
VALUES(NEW.COD-PROD, SYSDATE,
                          NEW.QTA-RIORD)
END
```

Evento
Condizione

Azione