STRUCTURED QUERY LANGUAGE

Vincenzo Calabrò

Linguaggi per basi di dati

Possibili classificazioni:

- formali o programmativi
- procedurali o dichiarativi

Algebra

- formale
- procedurale

SQL

- programmativo
- dichiarativo

Linguaggio di riferimento per basi di dati relazionali

- originariamente acronimo di Structured Query Language, ora considerato nome proprio

- storia:
  - prima proposta: linguaggio di interrogazione di System R (IBM Research 1974)
  - prima implementazione commerciale: SQL/DS (IBM 1981)

Standardizzazione di SQL

- 1983: standard de facto
- 1986: prima versione ufficiale (SQL-1)
- 1989: revisione ad SQL-1 (SQL-89)
- 1992: seconda versione (SQL-2 o SQL-92)
- 1999: terza versione (SQL-3 o SQL:1999)

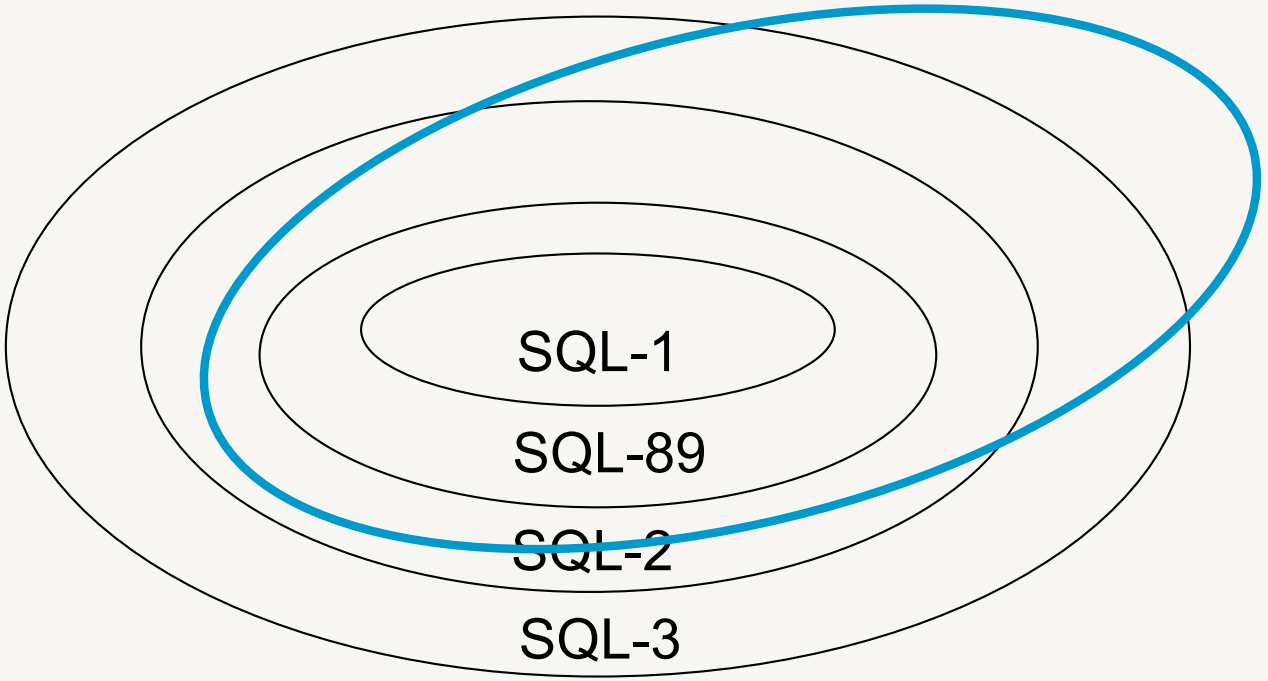
Ci riferiremo a SQL-2 riferimento per i sistemi esistenti (SQL-3 ancora lontano dall'essere comunemente adottato)

Distingue tre livelli

- Entry SQL
  - equivalente a SQL-89
- Intermediate SQL
  - caratteristiche ritenute più importanti per le esigenze di mercato, offerto dalla maggior parte dei DBMS
- Full SQL
  - SQL-92, include tutte le funzioni avanzate che i sistemi stanno progressivamente aggiungendo

SQL nei sistemi commerciali

La maggior parte dei sistemi è conforme a Entry SQL e offre estensioni proprietarie per le funzioni avanzate



SQL: DDL e DML

Data Definition Language (DDL)

- linguaggio di definizione
  - domini, tabelle, indici, autorizzazioni, viste, vincoli, procedure, trigger

Data Manipulation Language (DML)

- linguaggio di manipolazione
  - query
  - modifiche
  - comandi transazionali

Notazione

Useremo la seguente notazione:

corsivo	per scrivere variabili
`courier`	per scrivere termini del linguaggio
< >	isolano un termine della sintassi
[ ]	termine all'interno è opzionale
{ }	termine all'interno può ripetersi 0 o più volte
|	termini in alternativa

Domini

Specificano i valori ammissibili per gli attributi

- Elementari, predefiniti dallo standard o built-in, includono:
  - caratteri
  - tipi numerici (esatti/approssimati)
  - data e ora (date, time, timestamp)
  - intervalli temporali
  - boolean (in SQL:1999)
  - blob, binary large object (in SQL:1999)
  - clob, character large object (in SQL:1999)
- Definiti dall'utente, sulla base di altri domini

Domini definiti dall'utente (1)

`create domain` *NomeDominio* `as` *TipoDiDato*
[*ValoreDiDefault*] [*Vincoli*]

- *NomeDominio:* nome da dare al dominio
- *TipoDiDato:* dominio su cui costruiamo
- *ValoreDiDefault:* da assumere se nulla è specificato
- *Vincoli:* condizioni che i valori del dominio devono rispettare

Domini definiti dall'utente (2)

ValoreDiDefault:

`default` <*ValoreGenerico* | `user` | `null` >

- *ValoreGenerico:* qualsiasi valore compatibile con
  il dominio
- `user`: login dell'utente che effettua il comando
- `null`: valore nullo, polimorfico:
  - esiste ma non noto (es., data di nascita)
  - non esiste (es., num patente per un minorenne)
  - non si sa (es., num patente per un maggiorenne)

Domini definiti dall'utente: esempi

```
create domain PrezzoQuotidiani as decimal(2)
     default 1.00
     not null

create domain OreLezione as smallint
     default 48
```

Schemi

```
create schema [NomeSchema]
      [ [authorization] Autorizzazione]
      {DefElementoSchema}
```

- *NomeSchema*, nome dello schema
- *Autorizzazione*, nome dell'utente proprietario
- *DefElementoSchema*, definizione di elementi nello schema
  - domini, tabelle, indici, asserzioni, viste, privilegi, …

Tabelle

```
create table NomeTabella
( NomeAttributo Dominio [ValoreDiDefault] [Vincoli]
{, NomeAttributo Dominio [ValoreDiDefault] [Vincoli]}
{AltriVincoli} )
```

- *Vincoli*:
  - intra-relazionali
  - inter-relazionali

Vincoli intra-relazionali

Coinvolgono una relazione, devono essere verificati da ogni istanza della relazione

- `not null` (su un singolo attributo)
- `unique`, permette la definizione di chiavi candidate
  - per un solo attributo, `unique` dopo il dominio
  - per più attributi: `unique`(*Attributo* {, *Attributo*})
- `primary key`, definisce la chiave primaria (implica `unique` e `not null`)
- `check`, specifica vincoli

Vincoli intra-relazionali: esempi (1)

- Ogni coppia (Nome, Cognome) identifica univocamente una tupla

```
Nome character(20) not null
Cognome character(20) not null,
unique(Nome,Cognome)
```

- Il nome o il cognome singolarmente identificano univocamente una tupla

```
Nome character(20) not null unique
Cognome character(20) not null unique
```

Vincoli intra-relazionali: esempi (2)

```
create table Studenti
(  Matr    char(6)     primary key,
   Cognome varchar(30) not null,
   Nome    varchar(30) not null,
   CdL     char(5)     not null)


create table Corsi
(  Codice  char(6)     primary key,
   Titolo  varchar(30) not null unique,
   Cognome varchar(20),
   Nome    varchar(20))
```

Vincoli inter-relazionali

Coinvolgono più relazioni, devono essere verificati da ogni istanza delle relazioni

- `check`, vincoli generici
- `references` e `foreign key`, integrità referenziale
  - per un solo attributo, `references` dopo il dominio
  - per più attributi:
    `foreign key`(*Attributo* {, *Attributo*}) `references` …

  si possono associare delle politiche di reazione ai vincoli di integrità referenziale

Vincoli inter-relazionali: esempi (1)

```
create table Docenti
(  Cognome varchar(20),
   Nome    varchar(20),
   Dip     varchar(30),
   primary key (Cognome,Nome))


create table Corsi
(  Codice  char(6)     primary key,
   Titolo  varchar(30) not null unique,
   Cognome varchar(20),
   Nome    varchar(20),
   foreign key (Cognome,Nome)
     references Docenti(Cognome,Nome))
```

Vincoli inter-relazionali: esempi (2)

```
create table Esami
(  Matr       char(6)  references
Studenti(Matr),
   Cod-corso char(6)  references
Corsi(Codice),
   Data       date     not null,
   Voto       smallint not null,
   primary key (Matr,Cod-corso))
```

Integrità referenziale

I vincoli possono essere violati a seguito di:

- operazioni sulla tabella interna (che referenzia)
  - aggiunta/modifica di tuple (che non hanno corrispondenza in una tabella esterna)
  - nel caso di violazione il sistema rifiuta l'operazione
- operazioni sulla tabella esterna (referenziata)
  - cancellazione/modifica di tuple (che sono correntemente referenziate da qualche tabella)
  - SQL permette di specificare la politica di reazione nel caso di violazione

Integrità referenziale: politiche di reazione

```
on <delete|update>
   <cascade|set null|set default|no
action>
```

- `cascade`: propaga la modifica/cancellazione
- `set null`: mette a null l'attributo che rimarrebbe pendente
- `set default`: mette al valore di default l'attributo che rimarrebbe pendente
- `no action`: impedisce la modifica/cancellazione

Politiche di reazione: esempi (1)

```
create table Esami
(  Matr      char(6)  references
Studenti(Matr)
     on delete cascade
     on update cascade,
   Cod-corso char(6)  references
Corsi(Codice)
     on delete no action
     on update no action,
   Data      date     not null,
   Voto      smallint not null,
   primary key (Matr,Cod-corso))
```

Politiche di reazione: esempi (2)

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)

Cosa succede agli esami di uno studente se si cancella uno studente?

- `cascade`: si cancellano
- `set null`: si pone a null la matricola (violerebbe non nullità di chiave)
- `set default`: si pone al valore di default la matricola
- `no action`: si impedisce la cancellazione dello studente

Politiche di reazione: esempi (3)

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)

Cosa succede agli esami di uno studente se si modifica la matricola dello studente (nella tabella Studenti)?

- `cascade`: si modifica la matricola degli esami
- `set null`: si pone a null la matricola (violerebbe non nullità di chiave)
- `set default`: si pone al valore di default la matricola
- `no action`: si impedisce la modifica dello studente

Modifiche agli schemi

Permettono di modificare definizioni di elementi dello schema precedentemente introdotti

- `alter`, per modificare domini e schemi di tabelle
  - se si aggiunge un nuovo vincolo questo deve essere soddisfatto dai dati già presenti
- `drop`, per rimuovere elementi dallo schema della base di dati

Alter

```
alter domain NomeDominio
    < set default ValoreDiDefault |
      drop default |
      add constraint DefVincolo |
      drop constraint NomeVincolo >


alter table NomeTabella
   < alter column NomeAttributo
    <set default ValoreDiDefault |drop
default> |
      add constraint DefVincolo |
      drop constraint NomeVincolo |
      add column DefAttributo |
      drop column NomeAttributo >
```

Alter: esempi

```
alter domain PrezzoQuotidiani
    set default 1.20

alter table Esami
    add column Sessione smallint
```

Drop

```
drop < schema | domain | table | view |
       assertion > NomeElemento
      [ restrict  | cascade ]
```

- `restrict`: il comando non deve essere eseguito in presenza di oggetti non vuoti o utilizzati (è il default)
- `cascade`: se si rimuove un oggetto non vuoto o referenziato tutti quelli che contiene o si riferiscono vengono rimossi
  - se si rimuove un dominio *D* gli attributi che lo utilizzavano vengono riferiti al dominio elementare su cui *D* era stato definito

Cataloghi relazionali

Catalogo: dizionario dei dati, che descrive la struttura degli elementi della base di dati

- basato su una struttura relazionale (riflessività)
- due livelli:
  - Definition_Schema: tabelle che contengono la descrizione di tutte le strutture della base di dati
    - non vincolante
  - Information_Schema: composto da viste sul Definition_Schema
    - vincolante

Linguaggio dichiarativo

- l'utente specifica l'informazione che vuole ritrovare, non la procedura per estrarla

- le interrogazioni vengono tradotte dall'ottimizzatore all'interno del DBMS
  - il programmatore si focalizza sulla leggibilità, non sull'efficienza

Interrogazioni semplici (1)

`select` *ListaAttributi*
`from` *ListaTabelle*
[`where` *Condizione*]

- ritorna la *ListaAttributi* nel prodotto cartesiano di *ListaTabelle* per le tuple che soddisfano la *Condizione*

Equivale (**a meno di duplicati**) a

$\Pi_{ListaAttributi}(\sigma_{Condizione}\ T_1 \times T_2 \times \ldots \times T_n)$

- $T_1, T_2, \ldots, T_n$ sono le tabelle in *ListaTabelle*

Interrogazioni semplici (2)

`select` *AttrEspr* [[`as`] *Alias*] {, *AttrEspr* [[`as`] *Alias*]}
`from` *Tabella* [[`as`] *Alias*] {, *Tabella* [[`as`] *Alias*]}
[`where` *Condizione*]

Esempio

STUDENTI(Matr,Cognome,Nome,CdL)

- ```
  select Cognome, Nome, CdL as
  CorsoDiLaurea
   from Studenti
   where Matr='123456'
  ```
  ```

Clausola select

AttrEspr:

- carattere speciale * include tutti gli attributi
  - `select *`
- può anche essere una formula
  - es., `select Tasse/12 as TasseMensili`
- se preceduta da `distinct` elimina i duplicati (altrimenti mantenuti da SQL, `all` default)
  - es., `select distinct CdL as CorsiDiLaurea`
  - es., `select distinct Cognome, Nome`
  - es., `select all CdL`

Clausola from

ListaTabelle

- lista le tabelle da unire in prodotto cartesiano, ognuna delle quali può essere ridenominata
  - es., `from Studenti as S, Esami as E`

- SQL-92 permette la specifica del join nella clausola from
  - es., `from Studenti join Esami on Studenti.Matr=Esami.Matr`

Clausola where

Condizione

- espressione booleana di predicati semplici, come in algebra
- alcuni predicati aggiuntivi
  - `like`, confronto di stringhe

    es., `Corsi.Titolo like 'Basi%'`

    `Matr like '12_45_'`
  - `between`, confronto di date

    es., `Esami.Data between 1-1-2005`

    `and 31-12-2005`

Interrogazioni semplici: esempi

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

- ```
  select *
  from Studenti
  where CdL='SSRI'or CdL='SSRIonline'
  ```
- ```
  select distinct Data
  from Esami
  where Voto=30
  ```
- ```
  select distinct Cognome,Nome
  from Studenti,Esami
  where Studenti.Matr=Esami.Matr and
  Voto=30
  ```
  ```

Valori nulli nelle interrogazioni

NULL, valore polimorfo

- esiste ma non noto (es., data di nascita)
- non esiste (es., numero patente per un minorenne)
- non si sa (es., numero patente per un maggiorenne)

SQL-89, logica a due valori

- confronto con NULL restituisce false

SQL-2, logica a tre valori

- confronto con NULL restituisce unknown
- per verifica su valori nulli:
  - *Attributo* `is [not] null`

Logica a tre valori

and	V	U	F
V	V	U	F
U	U	U	F
F	F	F	F

or	V	U	F
V	V	V	V
U	V	U	U
F	V	U	F

not	
V	F
U	U
F	V

- SQL ritorna solo le tuple per cui la condizione è `Vera`

- la differenza fra logica a due e tre valori emerge in presenza di espressioni complicate

Valori nulli nelle interrogazioni: esempio

STUDENTI(Matr,Cognome,Nome,CdL)

- ```
  select *
  from Studenti
  where CdL='SSRI'or CdL<>'SSRI'
  ```

è equivalente a

- ```
  select *
  from Studenti
  where CdL is not null
  ```

Join (1)

SQL-92: permette di rappresentare i join esplicitandoli nella clausola `from`

`select` *AttrEspr* [[`as`] *Alias*] {, *AttrEspr* [[`as`] *Alias*]}
`from` *Tabella* [[`as`] *Alias*]
{ [*TipoJoin* [`join`] *Tabella* [[`as`]*Alias*] `on` *CondizioneJoin* }
[`where` *AltraCondizione*]

- *TipoJoin*:
  - `inner`, default (theta join)
  - `right [outer]`
  - `left [outer]`
  - `full [outer]`
  - `natural`, implementato raramente

Join (2)

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

- ```
  select S.Matr,Cognome,Nome
  from Studenti as S join Esami as E
    on S.Matr=E.Matr
  where Voto=30
  ```

è equivalente a

- ```
  select S.Matr,Cognome,Nome
  from Studenti as S, Esami as E
  where S.Matr=E.Matr and Voto=30
  ```

Join (3)

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

- ```
  select S.Matr,Cognome,Nome
  from Studenti as S join Esami as E
   on S.Matr=E.Matr
  ```

- ```
  select S.Matr,Cognome,Nome,
         Cod-corso,Voto
  from Studenti as S join Esami as E
   on S.Matr=E.Matr
  ```

(Se uno studente non ha esami non compare nel risultato!)

Outer Join

Permettono di mantenere le tuple di una tabella anche se non hanno corrispondenza nell'altra (concatenandole con valori nulli)

- `right [outer]`
  - mantiene le tuple della tabella di destra
- `left [outer]`
  - mantiene le tuple della tabella di sinistra
- `full [outer]`
  - mantiene le tuple di entrambe le tabelle

Outer Join: esempi

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)
CORSI(Codice,Titolo,Docente)

- seleziona *tutti* gli studenti con l'eventuale elenco degli esami (cod-corso e voto)

```
select S.*,Cod-corso,Voto
from Studenti as S leftjoin Esami as E
  on S.Matr=E.Matr
```

- seleziona *tutti* i corsi (codice e titolo) con l'eventuale elenco delle matricole degli studenti che hanno fatto l'esame

```
select Codice,Titolo,Matr
from Esami rightjoin Corsi on Cod-corso=Codice
```

Uso di variabili

SQL permette di specificare alias per le tabelle da interrogare

- possono essere utilizzate come variabili
  - distinguere diverse occorrenze di una stessa tabella
  - query nidificate

Uso di variabili: esempio

IMPIEGATI(Matr,Cognome,Nome,Dip,Matr-manager)

- elenca gli impiegati del dipartimento A indicando anche Cognome e Nome del loro manager (se esiste)

```
select I1.*,I2.Cognome, I2.Nome
from Impiegati as I1 leftjoin Impiegati as
I2
        on I1.Matr-manager=I2.Matr
where I1.Dip='A'
```

Interrogazioni complesse

Interrogazioni:

- con ordinamento
- con aggregazioni
- con raggruppamento
- insiemistiche
- nidificate
- con funzioni scalari

Ordinamento

`orderby` *AttrDiOrdinamento* [`asc` | `desc`]
{, *AttrDiOrdinamento* [`asc` | `desc`]}

Ordina le righe in ordine ascendente (`asc` default) o discendente (`desc`) degli attributi specificati

Ordinamento: esempio

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

- elenca gli studenti di SSRIonline che hanno fatto esami, in ordine ascendente di matricola e, per ognuno, indicando gli esami in ordine discendente di voto

```
select S.Matr,Cognome,Nome,Cod-corso,Voto
from Studenti as S join Esami as E
  on S.Matr=E.Matr
where CdL='SSRIonline'
order by S.Matr, Voto desc
```

Funzioni aggregate

- Importante estensione rispetto all'algebra

- SQL-2 offre cinque operatori aggregati:
  - `count`, cardinalità
  - `sum`, sommatoria
  - `max`, massimo
  - `min`, minimo
  - `avg`, media

Count

`count (< *|[distinct | all]` *ListaAttributi* `>)`

Restituisce il numero di righe (`*`) o valori degli *ListaAttributi*

- `distinct` considera i valori diversi
- `all` considera i valori non nulli (default)

Count: esempio

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)

- calcola il numero di studenti iscritti a SSRIonline

```
select count(*)
from Studenti
where CdL='SSRIonline'
```

- calcola il numero di studenti che hanno fatto almeno un esame

```
select count(distinct Matr)
from Esami
```

Sum, max, min, avg

`<sum|max|min|avg>` `(` `[distinct|all]` *AttrEspr* `)`

calcola il risultato dell'operazione su *AttrEspr*

- `distinct` valori diversi, considera una sola volta ciascun valore
  - utile solo per le funzioni `sum` e `avg`
- `all` valori non nulli (default)

Sum, max, min, avg: esempi (1)

ESAMI(Matr,Cod-corso,Data,Voto)

- calcola il voto più alto e più basso dell'esame di Basidati, il cui codice è BD

```
select max(Voto) as Alto, min(Voto) as Basso
from Esami
where Cod-corso='BD'
```

- calcola la media dei voti della matricola 123456

```
select avg(Voto)as MediaVoti
from Esami
where Matr='123456'
```

Sum, max, min, avg: esempi (2)

ESAMI(Matr,Cod-corso,Data,Voto)

- Query scorretta!

```
select Matr, max(Voto)
from Esami
where Cod-corso='BD'
```

La target list deve essere omogenea

Raggruppamento

Si possono applicare operatori aggregati a sottoinsiemi di righe

- clausole:
  - `group by`, raggruppamento
    (`group by` *ListaAttributi*)
  - `having`, selezione dei gruppi
    (`having` *Condizione*)

Raggruppamento: esempi (1)

ESAMI(Matr,Cod-corso,Data,Voto)

- calcola per ogni studente la media dei voti degli esami nel 2005

```
select Matr, avg(Voto) as MediaVoti
from Esami
where Data between 1-1-2005 and 31-12-2005
group by Matr
```

- calcola, per ogni studente che ha fatto almeno tre esami, la media dei voti

```
select Matr, avg(Voto) as MediaVoti
from Esami
group by Matr
having count(*) >= 3
```

Raggruppamento : esempi (2)

Nelle interrogazioni con `group by`, la target list non può contenere attributi che non siano elencati nella clausola `group by`

Esempio

- Query scorretta!

```
select Cod-corso,Data,avg(Voto) as MediaVoti
from Esami
group by Cod-corso
```

Raggruppamento: esempi (3)

- Query scorretta!

```
select S.Matr, Cognome, Nome,
       avg(Voto) as MediaVoti
from Esami as E join Studenti as S
  on S.Matr=E.Matr
group by S.Matr
```

- Query corretta

```
select S.Matr, Cognome, Nome,
       avg(Voto)as MediaVoti
from Esami as E join Studenti as S
        on S.Matr=E.Matr
group by S.Matr, Cognome, Nome
```

Where o having?

Soltanto i predicati che richiedono la valutazione di funzioni aggregate dovrebbero comparire nella clausola `having`

Esempio
ESAMI(Matr,Cod-corso,Data,Voto)

- trova i corsi in cui la media dei voti assegnati nel 2005 è maggiore di 25

```
select Cod-corso
from Esami
where Data between 1-1-2005 and 31-12-2005
group by Cod-corso
having avg(Voto) > 25
```

Raggruppamento e ordinamento: esempio

ESAMI(Matr,Cod-corso,Data,Voto)

- trova i corsi e la media dei voti, per i corsi in cui il numero di esami è maggiore di 3; ordina il risultato in ordine decrescente di media

```
select Cod-corso, avg(Voto)
from Esami
group by Cod-corso
having count(*) > 3
order by avg(Voto) desc
```

Insiemistiche (1)

SelectSQL {<`union` | `intersect` | `except`>
[`all`] *SelectSQL*}

- combina il risultato di due interrogazioni:
  - `union`, unione
  - `intersect`, intersezione
  - `except`, differenza
- per default elimina i duplicati
  - `all`, per mantenere i duplicati

Insiemistiche (2)

Gli operandi devono avere attributi in ugual numero e con domini compatibili

- la corrispondenza fra gli attributi si basa sulla posizione, non sul nome
- usualmente il risultato prende i nomi degli attributi del primo operando

Insiemistiche: esempio (1)

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

- trova le matricole degli studenti iscritti a SSRIonline che non hanno fatto neanche un esame

```
select Matr
from Studenti
where CdL='SSRIonline'
   except
select Matr
from Esami
```

Insiemistiche: esempio (2)

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

CORSI(Codice,Titolo,Docente)

- trova le matricole degli studenti iscritti a SSRIonline che hanno fatto Basidati

```
select Matr
from Studenti
where CdL='SSRIonline'
   intersect
select Matr
from Esami join Corsi on Cod-corso=Codice
where Titolo='Basidati'
```

Interrogazioni nidificate (1)

Nella clausola `where` possono comparire predicati che:

- confrontano un attributo (o un'espressione sugli attributi) con il risultato di una query SQL
- usano il quantificatore esistenziale (∃) sul risultato di una query

L'interrogazione che compare nella clausola where è chiamata interrogazione nidificata

Interrogazioni nidificate (2)

Confrontano un attributo (o un'espressione sugli attributi) con il risultato di una query SQL

ValScalare <*Operator* [any|all] | [not] in> *SelectSQL*

- *ValScalare*: valore scalare o attributo con valore scalare
- *Operator*: uno qualsiasi tra =, <>, <, =<, >, >=
- any, il predicato è vero se almeno una tupla restituita da *SelectSQL* soddisfa il confronto
- all, il predicato è vero se tutte le tuple restituite da *SelectSQL* soddisfano il confronto
- in, equivalente a =any
- not in, equivalente a <>all

Interrogazioni nidificate (3)

Usano il quantificatore esistenziale (∃) sul risultato di una query

`[not] exists` *SelectStarSQL*

- *SelectStarSQL*: query della forma `select *`

Spesso utilizzate con passaggio di variabili fra le interrogazioni

Interrogazioni nidificate: esempio (1)

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)

- trova cognome e nome degli studenti che hanno preso almeno un 28 o un 30

```
select Cognome, Nome
from Studenti
where Matr = any (select Matr
                  from Esami
                  where Voto=28 or Voto=30)
```

`= any` poteva anche essere scritto come `in`

Interrogazioni nidificate: esempio (2)

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)

- trova cognome e nome degli studenti che non hanno mai preso 30

```
select Cognome, Nome
from Studenti
where Matr <> all (select Matr
                   from Esami
                   where Voto=30)
```

`<> all` poteva anche essere scritto come `not in`

Interrogazioni nidificate: esempio (3)

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)
CORSI(Codice,Titolo,Docente)

- trova cognome e nome degli studenti che hanno fatto almeno un esame con il docente Rossi

```
select Cognome, Nome
from Studenti
where Matr in
      (select Matr
       from Esami
       where Cod-corso in
             (select Codice
              from Corsi
              where Docente='Rossi'))
```

Interrogazioni nidificate: esempio (4)

STUDENTI(Matr,Cognome,Nome,CdL)

- trova tutti gli studenti che hanno degli omonimi

```
select *
from Studenti as S1
where (Nome,Cognome) in
      (select Nome,Cognome
       from Studenti as S2
       where S1.Matr <> S2.Matr)
```

- trova tutti gli studenti che non hanno degli omonimi
  - come sopra ma con `not in`

Interrogazioni nidificate: esempio (5)

ESAMI(Matr,Cod-corso,Data,Voto)

- trova la matricola degli studenti che hanno preso il voto più alto

```
select Matr
from Esami
where Voto = (select max(Voto)
                  from Esami)
```

oppure

```
select Matr
from Esami
where Voto >= all (select Voto
                       from Esami)
```

Interrogazioni nidificate: esempio (6)

ESAMI(Matr,Cod-corso,Data,Voto)

- trova la matricola degli studenti che hanno fatto più di un esame in uno stesso giorno

```
select Matr
from Esami as E1
where exists
      (select *
       from Esami as E2
       where E1.Matr = E2.Matr and
             E1.Data = E2.Data and
             E1.Cod-corso <> E2.Cod-corso)
```

Interrogazioni nidificate: esempio (7)

STUDENTI(Matr,Cognome,Nome,CdL)

- trova tutti gli studenti che hanno degli omonimi (soluzione alternativa)

```
select *
from Studenti as S1
where exists
      (select *
       from Studenti as S2
       where S1.Cognome = S2.Cognome and
             S1.Nome = S2.Nome and
             S1.Matr <> S2.Matr)
```

- trova tutti gli studenti che non hanno degli omonimi
  - come sopra ma con `not exists`

Interrogazioni nidificate: limiti

- le interrogazioni nidificate non possono contenere operatori insiemistici
  - operatori insiemistici si applicano solo a livello esterno

- regole di visibilità delle variabili
  - una variabile è visibile solo all'interno dell'interrogazione in cui è definita ed entro le sue eventuali nidificazioni
  - se il nome di una variabile è omesso si assume il riferimento alla variabile più vicina

Visibilità di variabili: esempio

ESAMI(Matr,Cod-corso,Data,Voto)

- Query scorretta!

```
select *
from Studenti
where Matr in (select Matr
                   from Esami as E1
                   where Voto=28)
   or Matr in (select Matr
                   from Esami as E2
                   where E2.Matr = E1.Matr)
```

Funzioni scalari

Previste dallo standard:

- temporali (es., `current_date`, `current_time`)
- manipolazione di stringhe (es., `char_lenght`)
- conversione di domini (`cast`)
- condizionali

Offerte da diverse implementazioni, ad esempio:

- formattazione output
- funzioni matematiche (es., `sqrt`)
- funzioni di accesso ai servizi del SO

Funzioni condizionali

Non estendono il potere espressivo del linguaggio ma permettono di realizzare comandi in modo più compatto

- `coalesce`
- `nullif`
- `case`

Coalesce

Ammette come argomento una sequenza di espressioni e restituisce il primo valore non nullo

Esempio

STUDENTI(Matr,Cognome,Nome,CdL)

- trova il CdL di ogni studente, utilizzando la stringa “non ins.” nel caso non sia inserito

```
select Matr,Cognome,Nome,coalesce(CdL,'non ins.')
from Studenti
```

Nullif

Richiede come argomento una espressione e un valore costante, restituisce

- null se l'espressione valuta pari al valore costante
- il valore dell'espressione altrimenti

Esempio

ESAMI(Matr,Cod-corso,Data,Voto)

- trova gli esami, utilizzando il valore nullo nel caso il voto sia "ritirato"

```
select Matr,Cod-corso,Data,nullif(Voto,'ritirato')
from Esami
```

Case

Permette di specificare espressioni condizionali che generano i valori sulla base di generici predicati SQL

```
case when Condizione then Espressione
    { when Condizione then Espressione}
      else Espressione
end
```

La funzione `case` permette di scrivere una medesima operazione (che considera valori diversi) con una sola istruzione

Case: esempio

ESAMI(Matr,Cod-corso,Data,Voto)

- trova gli esami convertendo i voti in fasce di punteggio (A,B,C)

```
select Matr,Cod-corso,
   case
   when Voto < 24 then ‘C’
   when Voto >= 24 and Voto < 27 then ‘B’
   else ‘A’
   end as Fascia
from Esami
```

Comandi di modifica

Istruzioni per

- inserimento (`insert`)
- cancellazione (`delete`)
- modifica dei valori degli attributi (`update`)

Tutte le istruzioni operano su un insieme di tuple

Inserimento

`insert into` *NomeTabella* [(*ListaAttributi*)]
`<` `values` (*ListaDiValori*) | *SelectSQL* >

- *ListaAttributi* attributi da inserire:
  - quelli non specificati vengono messi a null
- *ListaDiValori* valori da inserire
- *SelectSQL* query che ritorna i valori da inserire

La corrispondenza fra gli attributi e i valori da inserire è posizionale

Inserimento: esempi

STUDENTI(Matr,Cognome,Nome,CdL)

STUDENTIESTERNI(Codice,Cognome,Nome)

```
insert into Studenti
values ('123456','Rossi','Mario','SSRI')

insert into Studenti(Matr,Cognome,Nome)
select *
from StudentiEsterni
```

Cancellazione

`delete from` *NomeTabella* [`where` *Condizione*]

- *Condizione*
  - come nelle interrogazioni
- se si omette la clausola `where` vengono cancellate tutte le tuple della tabella
  - rimane la tabella vuota (diverso da `drop`!)
- se è presente un vincolo di integrità referenziale con politica `cascade` può provocare cancellazioni in altre tabelle

Cancellazione: esempio

STUDENTI(Matr,Cognome,Nome,CdL)
ESAMI(Matr,Cod-corso,Data,Voto)

- cancella lo studente con matricola 123456

```
delete from Studenti
where Matr='123456'
```

- cancella gli studenti che non hanno sostenuto esami

```
delete from Studenti
where Matr not in (select Matr
                   from Esami)
```

Aggiornamento

update *NomeTabella* set
Attributo = <*Espressione* |
SelectSQL|null|default>
{, *Attributo*=<*Espressione* |
SelectSQL|null|default>}
[where *Condizione*]

- *Condizione*
  - come nelle interrogazioni
- se si omette la clausola where vengono modificate tutte le tuple della tabella

Aggiornamento: esempio (1)

ESAMI(Matr,Cod-corso,Data,Voto)

- aumenta di un punto (se possibile) il voto di tutti gli studenti che hanno fatto più di tre esami

```
update Esami
set Voto = Voto+1
where Voto < 30 and Matr in (select Matr
                             from Esami
                             group by Matr
                             having count(*)>3)
```

Aggiornamento: esempio (2)

ESAMI(Matr,Cod-corso,Data,Voto)

- aumenta (se possibile) di un punto i voti di Basidati (codice ‘BD’), diminuisci (se possibile) di un punto i voti di Sistemi (codice ‘SO’)

```
update Esami
set Voto =
  case
  when (Cod-corso=‘BD’ and Voto<30) then
    Voto+1
  when (Cod-corso=‘SO’ and Voto>18) then
    Voto-1
  else Voto
  end
```

Vincoli di integrità generici

- associati a una tabella o attributo

  `check` (*Condizione*)

- associati alla base di dati

  `create assertion` *NomeAsserzione*
  `check` (*Condizione*)

Check: esempio

```
create table Corsi
(  Codice  char(6)       primary key,
           check (Codice like 'F%'),
   Titolo  varchar(30) not null unique,
   Cod-docente varchar(20)
   check (3 > (select count(*)
                 from Corsi as C
                 where Cod-docente = C.Cod-docente))
)
```

Definisce CORSI(Codice,Titolo,Cod-docente) con vincoli:

- il codice dei corsi deve cominciare per F
- ogni docente deve avere non più di due corsi

Asserzioni: esempio

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

deve esserci almeno un esame registrato per ogni studente

```
create assertion almeno-un-esame
check (0 = select count(*)
           from Studenti
            where Matr not in (select Matr
                               from Esami))
```

Controllo dei vincoli (1)

I vincoli definiti tramite clausola check o asserzione possono essere controllati in modo:

- immediato (`immediate`)
  - a seguito di ogni singola operazione
  - la loro violazione annulla l'operazione (rollback parziale)
- differito (`deferred`)
  - al termine delle transazioni
  - la loro violazione annulla l'intera transazione (rollback)

Controllo dei vincoli (2)

Tutti i vincoli predefiniti (`not null`, `unique`, `primary key`, `foreign key`) sono controllati in modo immediato

È possibile cambiare il tipo di controllo (default `immediate`) associato ai vincoli definiti tramite check o asserzione

`set constraints` *NomeVincolo*
[`immediate`|`deferred`]

Esempio

```
set constraints almeno-un-esame immediate
```

Viste

Tabelle virtuali definite come interrogazioni su altre tabelle (di base o virtuali)

`create view` *NomeVista* [(*ListaAttributi*)] `as` *SelectSQL*
[`with` [`local` | `cascaded`] `check option`]

- `check option`: controlla che modifiche sulla vista non eliminino tuple dalla vista (soddisfare predicato di selezione).
  Se la vista è definita in termini di altre viste:
  - `local`: controllo solo sulla vista
  - `cascaded`: controllo anche sulle viste su cui è definita (default)

Viste: esempio

STUDENTI(Matr,Cognome,Nome,CdL)

ESAMI(Matr,Cod-corso,Data,Voto)

```
create view Esami05 as
    select *
     from Esami
     where Data between 1-1-2005 and 31-12-2005

create view EsamiSSRI as
     select Esami.Matr,Cod-corso
     from Studenti join Esami
           on Studenti.Matr=Esami.Matr
     where CdL='SSRI'
```

Viste (su viste)

Si possono definire viste su altre viste

- unico vincolo: non mutua dipendenza

Esempio

Vista: ESAMI05(Matr,Cod-corso)

```
create view StudentiAttivi(Matr) as
    select Matr
    from Esami05
```

Viste e interrogazioni

Le viste sono utili per formulare interrogazioni che:

- combinano e nidificano diversi operatori aggregati
- fanno uso sofisticato dell'operatore di unione

Viste e interrogazioni: esempio (1)

STUDENTI(Matr,Cognome,Nome,CdL)

- trova il CdL con il maggior numero di studenti

```
select CdL
from Studenti
group by CdL
having count(*) >= all (select count(*)
                        from Studenti
                        group by CdL)
```

non è riconosciuta da tutti gli interpreti SQL

- non tutti gli interpreti permettono interrogazione nidificate nella clausola `having`

Viste e interrogazioni: esempio (2)

Soluzione con viste:

STUDENTI(Matr,Cognome,Nome,CdL)

- trova il CdL con il maggior numero di studenti

```
create view NumStudenti(CdL,numero) as
  select CdL,count(*)
  from Studenti
  group by CdL


select CdL
from NumStudenti
where numero = (select max(numero)
                from NumStudenti)
```

Viste e interrogazioni: esempio (3)

ESAMI(Matr,Cod-corso,Data,Voto)

- trova il numero medio di esami per ogni studente

Query scorretta!

```
select avg(count(*))
from Esami
group by Matr
```

non si possono combinare in cascata operatori aggregati

Viste e interrogazioni: esempio (4)

Soluzione con viste:

ESAMI(Matr,Cod-corso,Data,Voto)

- trova il numero medio di esami per ogni studente

```
create view NumEsami(Matr,numero) as
  select Matr,count(*)
  from Esami
  group by Matr


select avg(numero)
from NumEsami
```

Viste e aggiornamenti

Viste definite su più tabelle possono avere ambiguità negli aggiornamenti

- la maggior parte dei sistemi commerciali ammette modifiche solo a viste definite su una singola tabella

- qualche sistema ammette modifiche solo a viste definite su una tabella e che ne contengano la chiave primaria

Viste e aggiornamenti: esempio (1)

Aggiornamenti su una vista si propagano alla tabella su cui la vista è definita

Esempio

ESAMI(Matr,Cod-corso,Data,Voto)

```
create view Esami05 as
   select *
   from Esami
   where Data between 1-1-2005 and 31-12-2005


update Esami05
set Voto = Voto + 1
where Voto < 25
```

si propaga sulla tabella Esami

Viste e aggiornamenti: esempio (2)

CORSI(Codice,Titolo) ASSEGN(Codice-corso,Docente)

```
create view ListaDocCorsi(Docente,Titolo) as
   select Docente, Titolo
   from Corsi join Assegn on Codice=Codice-corso

update ListaDocCorsi
set Titolo = 'Basidati'
where Docente = 'Rossi'
```

Modifica ambigua!

- il corso ha cambiato titolo?
- Rossi ha cambiato corso?

Autorizzazioni in SQL (1)

SQL supporta una politica di tipo discrezionale (di tipo chiuso con ownership)

- politica discrezionale chiusa: solo le azioni esplicitamente permesse sono autorizzate
- ownership: il creatore di un oggetto ne è il proprietario e riceve sull'oggetto tutti i privilegi
  - per viste: ristretti a quelli che aveva sulle tabelle
- ogni elemento dello schema può essere protetto

Autorizzazioni in SQL (2)

- Utente predefinito `_system` rappresenta l'amministratore di sistema e ha pieno accesso a tutte le risorse

- Il proprietario di un oggetto può concedere privilegi sull'oggetto ad altri
  - se con `grant option` il ricevente potrà a sua volta garantire il privilegio ad altri

Autorizzazioni in SQL (3)

Ogni autorizzazione è caratterizzata da:

- l'oggetto al quale l'autorizzazione è riferita
- l'azione alla quale l'autorizzazione è riferita
- il soggetto al quale viene concesso il privilegio
  - utente, ruolo, procedura, …
- l'utente che ha concesso il privilegio
  - `_system` per le autorizzazioni date al proprietario dal sistema
- flag che indica se vale o meno la `grant option`

Azioni in SQL (1)

Azioni che possono comparire nelle autorizzazioni su tabelle:

- `select` (tabella, attributi)
- `insert` (tabella, attributi)
- `update` (tabella, attributi)
- `delete` (tabella)
- `trigger` (tabella)
- `references` (tabella, attributi)
- `all privileges` li riassume tutti

Esistono altri privilegi per i domini (es., `usage`) o le procedure (es., `execute`)

Azioni in SQL (2)

`references` (tabella, attributi)

- permette che venga fatto riferimento alla risorsa nell'ambito della definizione dello schema di una tabella
  - utile perché le politiche di reazione dei vincoli di integrità referenziale possono avere effetti sulle tabelle esterne (es., `no action` può impedire operazioni sulla tabella esterna)

Grant

- Il proprietario di un oggetto può concedere ad altri autorizzazioni per privilegi sull'oggetto

- Le autorizzazioni possono essere concesse con la `grant option`:
  - un utente che ha l'autorizzazione per un privilegio su una risorsa con `grant option` può concedere l'autorizzazione per il privilegio sulla risorsa (e la `grant option`) ad altri

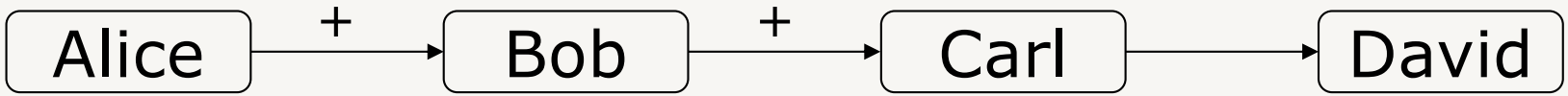
```
grant <ListaPriv | all privileges >
on Risorsa
to ListaRiceventi [with grant option]
```

Grant: esempio

Alice proprietario di STUDENTI

- Alice: `grant select on Studenti to Bob with grant option`
- Bob: `grant select on Studenti to Carl with grant option`
- Carl: `grant select on Studenti to David`

Grafo di autorizzazioni:



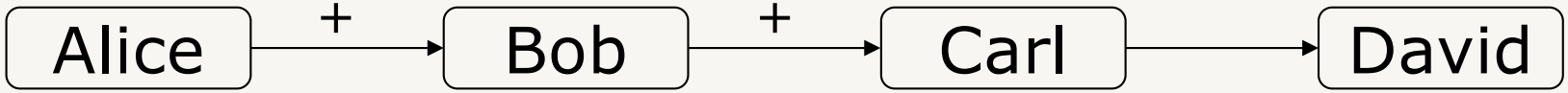
Revoke

- L'utente che ha concesso una autorizzazione può toglierla (`revoke`)
- Se l'utente a cui il privilegio è revocato aveva a sua volta concesso il privilegio ad altri e non ha altra autorizzazione per il privilegio con la `grant option`, le autorizzazioni da lui concesse rimarrebbero pendenti
- Due opzioni di revoca:
  - `restrict`: impedisce la revoca se ci sono autorizzazioni che rimarrebbero pendenti
  - `cascade`: propaga la revoca (cancellando ricorsivamente le autorizzazioni pendenti)

Revoke: esempio

```
revoke <ListaPriv | all privileges >
on Risorsa
from ListaRiceventi [restrict | cascade ]
```

Esempi



- Alice: `revoke select on Studenti from Bob restrict`

  non viene eseguita

- Alice: `revoke select on Studenti from Bob cascade`

  cancella anche le autorizzazioni di Carl e David

Autorizzazioni su viste

SQL non permette di restringere le autorizzazioni sulla base di condizioni sul valore dei dati (es., `select` ristretto alle tuple degli studenti di SSRI)

- Autorizzazioni ristrette ai dati sono possibili tramite viste:
  - si definisce la vista (che specifica le condizioni)
  - si concedono privilegi sulla vista
  - l'esercizio del privilegio sulla vista non implica (e non necessita) il privilegio sulle tabelle sottostanti

Autorizzazioni su viste: esempio

STUDENTI(Matr,Cognome,Nome,CdL)

```
create view Studentissri as
    select *
     from Studenti
     where CdL = 'SSRI'

grant select on Studentissri to Ellen
```

Ruoli

SQL-3 ha introdotto il concetto di ruolo per le autorizzazioni

- ruolo: rappresenta un insieme di privilegi; attivando il ruolo un utente è abilitato ad esercitare i privilegi associati al ruolo

Notare differenza con gruppi

- gruppo: insieme di utenti; come membro di un gruppo un utente ha tutti i privilegi del gruppo
  - non possono essere attivati/disattivati a piacere

Grant e revoke di ruoli

- Il creatore di un ruolo può concedere ad altri il ruolo (abilitandoli così ad attivare il ruolo)
- I ruoli possono essere concessi con `admin option`
  - equivale alla `grant option` delle autorizzazioni

```
grant ListaRuoli to ListaRiceventi
[with admin option]
```

- La revoca di ruoli può propagarsi ricorsivamente (come per le autorizzazioni)

```
revoke [admin option] ListaRuoli
from ListaRiceventi [restrict | cascade]
```

SQL– Esercizi

Interrogazione 1

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare la tariffa minima applicata dall'operatore Vodafone

```
select min(Tariffa)
from Abbonamento
where Operatore = 'Vodafone'
```

Interrogazione 2 (1)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare nome e cognome degli utenti che possiedono un cellulare Nokia ed hanno un abbonamento Tim

```
select Nome,Cognome
from Utente, Abbonamento as A, Cellulare as C
where CF = A.CFUtente and CF = C.CFUtente and
      Operatore = 'Tim' and Marca = 'Nokia'
```

Interrogazione 2 (2)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare nome e cognome degli utenti che possiedono un cellulare Nokia ed hanno un abbonamento Tim

```
select Nome,Cognome
from Utente join Abbonamento as A
        on CF = A.CFUtente
        join Cellulare as C on CF = C.CFUtente
where Operatore = 'Tim' and Marca = 'Nokia'
```

Interrogazione 2 (3)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare nome e cognome degli utenti che possiedono un cellulare Nokia ed hanno un abbonamento Tim

```
select Nome,Cognome
from Utente
where  CF in (Select CFutente from Abbonamento
              where Operatore = 'Tim’)
       and
       CF in (Select CFutente from Cellulare
              where Marca= ’Nokia’)
```

Interrogazione 3 (1)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare codice fiscale, cognome e nome degli utenti che possiedono più di tre numeri telefonici

```
select CF,Cognome,Nome
from Utente join Abbonamento on CF = CFUtente
group by CF,Cognome,Nome
having count(Numero)> 3
```

Interrogazione 3 (2)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare codice fiscale, cognome e nome degli utenti che possiedono più di tre numeri telefonici

```
select CF,Cognome,Nome
from Utente
Where CF in (Select CFUtente
             from Abbonamento
             group by CFUtente
             having count(Numero)> 3)
```

Interrogazione 4

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare, per ciascun operatore, il numero di abbonamenti e di persone servite

```
select Operatore,count(Numero),
                        count(distinct CFUtente)
from Abbonamento
group by Operatore
```

Interrogazione 5 (1)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare gli operatori che hanno l'esclusiva per almeno una città

```
select distinct A1.Operatore
from Abbonamento as A1 join Utente as U1
        on CFUtente = CF
where not exists
        (select *
         from Abbonamento as A2 join
              Utente as U2 on CFUtente = CF
         where U1.Città = U2.Città and
              A1.Operatore <> A2.Operatore)
```

Interrogazione 5 (2)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare gli operatori che hanno l'esclusiva per almeno una città

```
select distinct A1.Operatore
from Abbonamento as A1 join Utente as U1
        on CFUtente = CF
where A1.Città not in
        (select A2.Città
         from Abbonamento as A2 join
              Utente as U2 on CFUtente = CF
         where A1.Operatore <> A2.Operatore)
```

Interrogazione 5 (3)

CELLULARE (Codice, CFUtente, Modello, Marca, Colore)
ABBONAMENTO(Numero, CFUtente, Operatore,Tariffa)
UTENTE(CF, Nome, Cognome, Città)

- determinare gli operatori che hanno l'esclusiva per almeno una città

```
select distinct Operatore
from Abbonamento as A1 join Utente as U1
        on CFUtente = CF
where Città in
        (select Città
         from Abbonamento as A2 join
              Utente as U2 on CFUtente = CF
         group by Città
         having count(distinct Operatore)=1)
```

Interrogazione 6

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il codice delle radio per cui il numero totale di ore di programmi musicali è minore o uguale a 8 (si noti che la radio ritornate devono avere almeno un programma musicale)

```
select CodiceRadio
from Programma
where Tipo = 'musicale'
group by CodiceRadio
having sum(Durata) <= 8
```

Interrogazione 7 (1)

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il nome e la frequenza delle radio che trasmettono in un luogo dove non trasmette alcuna altra radio

```
select R1.Nome, R1.Frequenza
from Radio as R1
where R1.Luogo not in
        (select R2.Luogo
         from Radio as R2
         where R1.Codice <> R2.Codice)
```

Interrogazione 7 (2)

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il nome e la frequenza delle radio che trasmettono in un luogo dove non trasmette alcuna altra radio

```
select R1.Nome, R1.Frequenza
from Radio as R1
where not exists
        (select *
         from Radio as R2
         where R1.Luogo = R2.Luogo
              and R1.Codice <> R2.Codice)
```

Interrogazione 8 (1)

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il nome del conduttore che conduce esclusivamente programmi di tipo musicale

```
select distinct Conduttore
from Programma
where Tipo = 'musicale'
        except
select Conduttore
from Programma
where Tipo <> 'musicale'
```

Interrogazione 8 (2)

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il nome del conduttore che conduce esclusivamente programmi di tipo musicale

```
select distinct Conduttore
from Programma
where Tipo = 'musicale' and
   Conduttore not in
              (select Conduttore
               from Programma
               where Tipo <> 'musicale')
```

Interrogazione 8 (3)

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il nome del conduttore che conduce esclusivamente programmi di tipo musicale

```
select distinct Conduttore
from Programma as P1
where P1.Tipo = 'musicale' and not exists
                (select *
                from Programma as P2
                where P1.Conduttore = P2.Conduttore
                and P2.Tipo <> 'musicale')
```

Interrogazione 9

RADIO (Codice, Nome, Frequenza, Luogo)
PROGRAMMA (Codice, Nome, Conduttore, FasciaOraria, Durata, Tipo, CodiceRadio)

- determinare il nome delle radio che trasmettono il maggior numero di programmi

```
create view NumProgrammi(NomeRadio,NumProg) as
  select R.Nome,count(*)
  from Radio as R join Programma as P
       on R.Codice = P.CodiceRadio
  group by P.CodiceRadio, R.Nome

select NomeRadio
from NumProgrammi
where NumProg = (select max(NumProg)
                 from NumProgrammi)
```

VINCENZO CALABRÒ

LinkedIn vincenzocalabro

www.vincenzocalabro.it