Basi di dati

SQL

Docente: Stefano Paraboschi
parabosc@elet.polimi.it

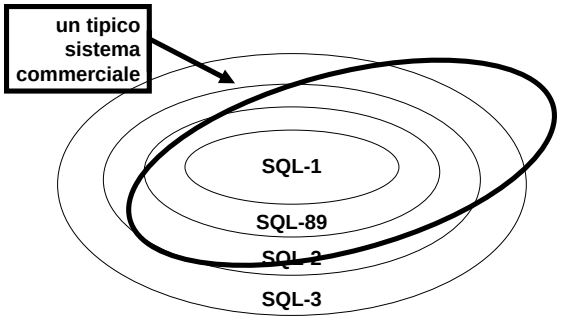
SQL

- Il nome sta per *Structured Query Language*
- Più che un linguaggio di query: si compone di una parte DDL e di una DML
  - DDL: definizione di domini, tabelle, indici, autorizzazioni, viste, vincoli, procedure, trigger
  - DML: linguaggio di query, linguaggio di modifica, comandi transazionali
- Storia:
  - Prima proposta: SEQUEL (IBM Research, 1974)
  - Prima implementazione commerciale in SQL/DS (IBM, 1981)

Standardizzazione di SQL

- La standardizzazione è stata cruciale per la diffusione di SQL
  - Dal 1983, standard de facto
  - Prima versione ufficiale nel 1986 (SQL-1), rivista nel 1989 (SQL-89)
  - Seconda versione nel 1992 (SQL-2 o SQL-92)
  - Terza versione nel 1999 (SQL-3 o SQL-99)
- In SQL-92 si distinguono tre livelli:
  - Entry SQL (più o meno equivalente a SQL-89)
  - Intermediate SQL
  - Full SQL
- La maggior parte dei sistemi è conforme al livello Entry e offre delle estensioni proprietarie per le funzioni avanzate

Potere espressivo di standard e sistemi commerciali



Domini

- I domini specificano i valori ammissibili per gli attributi
- Due categorie
  - Elementari (predefiniti dallo standard, elementary o built-in)
  - Definiti dall'utente (user-defined)

Domini elementari, 1

- Caratteri
  - Caratteri singoli o stringhe
  - Le stringhe possono avere lunghezza variabile
  - Possono usare una famiglia di caratteri (character set) diversa da quella di default (es., Latin, Greek, Cyrillic, etc.)
  - Sintassi:
    `character` [`varying`] [(*Lunghezza*)]
    [`character set` *FamigliaCaratteri*]
  - Si possono usare le alternative più compatte `char` e `varchar`, rispettivamente per `character` e `character varying`
  - Esempi:
    - `char(6)`
    - `varchar(50)`

Domini elementari, 2

- Bit
  - Valori booleani (vero/falso), singoli o in sequenza (la sequenza può essere di lunghezza variabile)
  - Sintassi:
    `bit` [`varying`] [(*Lunghezza*)]
- Domini numerici esatti
  - Valori esatti, interi o con una parte razionale
  - 4 alternative:
    `numeric` [(*Precisione* [, *Scala*])]
    `decimal` [(*Precisione* [, *Scala*])]
    `integer`
    `smallint`

Domini elementari, 3

- Domini numerici approssimati
  - Valori reali approssimati
  - Basati su una rappresentazione a virgola mobile
    `float` [(*Precisione*)]
    `double precision`
    `real`

Domini elementari, 4

- Istanti temporali
  `date`
  `time` [(*Precisione*)] [`with time zone`]
  `timestamp` [(*Precisione*)] [`with time zone`]
- Intervalli temporali
  `interval` *PrimaUnitàDiTempo* [`to` *UltimaUnitàDiTempo*]
  - Le unità di tempo sono divise in 2 gruppi:
    - year, month
    - day, hour, minute, second
  - Esempi:
    - `interval year to month`
    - `interval second`

Domini definiti dagli utenti

- Paragonabile alla definizione dei tipi nei linguaggi di programmazione
- Un dominio è caratterizzato da
  - nome
  - dominio elementare
  - valore di default
  - insieme di vincoli (constraint)
- Sintassi:
  `create domain` *NomeDominio* `as` *DominioElementare*
  [*ValoreDefault*] [*Constraints*]
- Esempio:
  `create domain Voto as smallint default null`

Valori di default per il dominio

- Definiscono il valore che deve assumere l'attributo quando non viene specificato un valore durante l'inserimento di una tupla
- Sintassi:
  `default` < *ValoreGenerico* | `user` | `null` >
- *ValoreGenerico* rappresenta un valore compatibile con il dominio, rappresentato come una costante o come un'espressione
- `user` è la login dell'utente che effettua il comando

Il valore "null"

null
è un valore polimorfico (che appartiene a tutti i domini) col significato di valore non noto

- il valore esiste in realtà ma è ignoto al database (es.: data di nascita)
- il valore è inapplicabile (es.: numero patente per minorenni)
- non si sa se il valore è inapplicabile o meno (es.: numero patente per un maggiorenne)

Vincoli intra-relazionali

- I vincoli sono condizioni che devono essere verificate da ogni istanza della base di dati
- I vincoli intra-relazionali coinvolgono una sola relazione
  - `not null` (su un solo attributo)
  - `unique`: permette la definizione di chiavi candidate; sintassi:
    - per un solo attributo:
      `unique,` dopo il dominio
    - per diversi attributi:
      `unique(` *Attributo* {, *Attributo* })
  - `primary key`: definisce la chiave primaria (una volta per ogni tabella; implica *not null*); sintassi come per `unique`
  - `check`: descritto più avanti

Definizione dei domini applicativi

```
create domain PrezzoQuotidiani
as integer
     default 1700
     not null

create domain OreLezione
as smallint
     default 40
```


Esempi di vincoli intra-relazionali

- Ogni coppia di attributi Nome e Cognome identifica univocamente ogni tupla

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

- Si noti la differenza con la seguente definizione (più restrittiva):

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


Definizione di schemi

- Uno *schema* è una collezione di oggetti:
  - domini, tabelle, indici, asserzioni, viste, privilegi
- Uno schema ha un nome e un proprietario

- Sintassi:
  `create schema` [*NomeSchema*]
  [[`authorization`] *Autorizzazione*]
  { *DefinizioneElementoSchema* }

Definizione di tabelle

- Una tabella SQL consiste di:
  - un insieme ordinato di attributi
  - un insieme di vincoli (eventualmente vuoto)
- Comando `create table`
  - definisce lo schema di una relazione, creandone un'istanza vuota
- Sintassi:
  `create table` *NomeTabella*
  (
  NomeAttributo Dominio [*ValoreDiDefault*] [*Constraints*]
  {, *NomeAttributo Dominio* [*ValoreDiDefault*] [*Constraints*] }
  [*AltriConstraints*]
)

Esempio di **`create table`** (1)

```
create table Studente
  (  Matr      character(6) primary key,
     Nome      varchar(30) not null,
     Città     varchar(20),
     CDip      char(3) )
```

Esempio di `create table` (2)

```
create table Esame
 (  Matr       char(6),
    CodCorso   char(6),
    Data       date  not null,
    Voto       smallint not null,
    primary key(Matr,CodCorso) )

 create table Corso
 (  CodCorso char(6) primary key,
    Titolo   varchar(30) not null,
    Docente  varchar(20) )
```


Integrita' referenziale

- **Esprime un legame gerarchico (padre-figlio) fra tabelle**
- **Alcuni attributi della tabella figlio sono definiti FOREIGN KEY**
- **I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre**

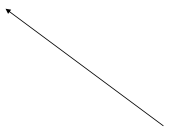
Vincoli inter-relazionali

- I vincoli possono tenere conto di diverse relazioni
  - `check`: descritto più avanti
  - `references` e `foreign key` permettono la definizione di vincoli di integrità referenziale; sintassi:
    - per un solo attributo
      `references` dopo il dominio
    - per diversi attributi
      `foreign key` (*Attributo* {, *Attributo* })
      `references` …
  - Si possono associare anche delle politiche di reazione alle violazioni dei vincoli di integrità referenziale

Esempio: studente - esame

Studente

Matr	
123	
415	
702	



Esame

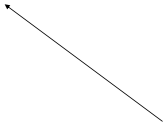
Matr	
123	
123	
702	

Il problema degli orfani

Studente

Matr	
~~123~~	
415	
702	

orfani:
tuple che restano prive di padre a causa di cancellazioni
e modifiche della tabella padre



Esame

Matr	
~~123~~	
~~123~~	
702	

Gestione degli orfani

- Le reazioni operano sulla tabella interna, in seguito a modifiche alla tabella esterna
- Le violazioni possono essere introdotte (1) da aggiornamenti (update) dell'attributo cui si fa riferimento oppure (2) da cancellazioni di tuple
- Reazioni previste:
  - `cascade`: propaga la modifica
  - `set null`: annulla l'attributo che fa riferimento
  - `set default`: assegna il valore di default all'attributo
  - `no action`: impedisce che la modifica possa avvenire
- La reazione può dipendere dall'evento; sintassi:
  `on` < `delete` | `update` >
  < `cascade` | `set null` | `set default` | `no action` >

Gestione degli orfani: cancellazione

Cosa succede degli esami se si cancella uno studente?

- **`cascade`**
  si cancellano anche gli esami dello studente
- **`set null`**
  si pone a null la matricola dei relativi esami
- **`set default`**
  si pone al valore di default la matricola dei relativi esami
- **`no action`**
  si impedisce la cancellazione dello studente

Gestione degli orfani: modifica

Cosa succede degli esami se si modifica la matricola di uno studente?

- **`cascade`**
  si modifica la matricola degli esami dello studente
- **`set null`**
  si pone a null la matricola dei relativi esami
- **`set default`**
  si pone al valore di default la matricola dei relativi esami
- **`no action`**
  si impedisce la modifica della matricola dello studente

Definizione: nella tabella figlio

```
create table Esame
   ( ....
     ....
     foreign key Matr
       references Studente
        on delete cascade
        on update cascade )
```


E' lecito essere figli di più padri

```
create table Esame
( ....
    primary key(Matr,CodCorso)
    foreign key Matr
      references Studente
        on delete cascade
        on update cascade
    foreign key CodCorso
      references Corso
        on delete no action
        on update no action )
```


Una istanza scorretta

Matr	Nome	Città	CDip
123			
415			
702			

Esame

Matr	Cod Corso	Data	Voto
123	1	7-9-97	30
123	2	8-1-98	28
~~123~~	~~2~~	~~1-8-97~~	~~28~~
702	2	7-9-97	20
~~702~~	~~1~~	~~NULL~~	~~NULL~~
~~714~~	~~1~~	~~7-9-97~~	~~28~~

viola la chiave

viola il NULL

viola la integrità referenziale

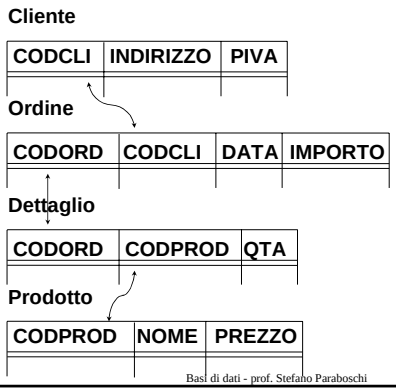
Una istanza corretta

Matr	Nome	Città	CDip
123			
415			
702			

Esame

Matr	Cod Corso	Data	Voto
123	1	7-9-97	30
123	2	8-1-98	28
702	2	7-9-97	20

Esempio: gestione ordini



Definizione della tabella Cliente

```
create table Cliente
 ( CodCli    char(6)  primary key,
   Indirizzo char(50),
   PIva      char(12) unique )
```


Definizione della tabella Ordine

```
create table Ordine
(  CodOrd   char(6)  primary key,
   CodCli   char(6)  not null
                     default='999999',
   Data     date,
   Importo  integer,
    foreign key CodCli
      references Cliente
         on delete set default
         on update set default)
```


Definizione della tabella Dettaglio

```
create table Dettaglio
(  CodOrd  char(6),
   CodProd char(6),
   Qta     smallint,
    primary key(CodOrd,CodProd)
    foreign key CodOrd
      references Ordine
        on delete cascade
        on update cascade
    foreign key CodProd
      references Prodotto
        on delete no action
        on update no action)
```


Definizione della tabella Prodotto

```
create table Prodotto
 ( CodProd   char(6)  primary key,
   Nome      char(20),
   Prezzo    smallint )
```


Modifiche degli schemi

- Necessarie per garantire l'evoluzione della base di dati a fronte di nuove esigenze
- Ci sono due comandi SQL appositi:
  - `alter` (`alter domain` ..., `alter table` …) modifica oggetti persistenti
  - `drop`
    `drop` < `schema` | `domain` | `table` | `view` | `assertion` > *NomeComponente* [`restrict` | `cascade`]
    cancella oggetti dallo schema

Modifica degli oggetti DDL

- **`alter`**
  - Si applica su domini e tabelle

```
es.: alter table Ordine
     add column NumFatt char(6)

es.: alter table Ordine
     alter column Importo
     add default 0

es.: alter table Ordine
     drop column Data
```


Cancellazione degli oggetti DDL

- **`drop`**
  - si applica su domini, tabelle, indici, view, asserzioni, procedure, trigger

```
es.: drop table Ordine
es.: drop index DataIx
```

- Opzioni `restrict` e `cascade`
  - **`restrict`**: impedisce `drop` se gli oggetti comprendono istanze
  - **`cascade`**: applica `drop` agli oggetti collegati

Cataloghi relazionali

- Il catalogo contiene il dizionario dei dati (data dictionary), ovvero la descrizione della struttura dei dati contenuti nel database
- È basato su una struttura relazionale
  - riflessività
- Lo standard SQL-2 organizza il catalogo su due livelli
  - **`Definition_Schema`** (composto da tabelle, non vincolante)
  - **`Information_Schema`** (composto da viste, vincolante)

Interrogazioni in SQL

SQL come linguaggio di interrogazione

- Le interrogazioni SQL sono dichiarative
  - l'utente specifica quale informazione è di suo interesse, ma non come estrarla dai dati
- Le interrogazioni vengono tradotte dall'ottimizzatore (query optimizer) nel linguaggio procedurale interno al DBMS
- Il programmatore si focalizza sulla leggibilità, non sull'efficienza
- È l'aspetto più qualificante delle basi di dati relazionali

Interrogazioni SQL

- Le interrogazioni SQL hanno una struttura `select-from-where`
- Sintassi:
  `select` *AttrEspr* [[`as`] *Alias*] {, *AttrEspr* [[`as`] *Alias*] }
  `from` *Tabella* [[`as`] *Alias*] {, *Tabella* [[`as`] *Alias*] }
  [`where` *Condizione*]
- Le tre parti della query sono chiamate:
  - clausola `select` / target list
  - clausola `from`
  - clausola `where`
- La query effettua il prodotto cartesiano delle tabelle nella clausola `from`, considera solo le righe che soddisfano la condizione nella clausola `where` e per ogni riga valuta le espressioni nella target list

Interpretazione algebrica delle query SQL

- La query generica:

```
select T_1.Attributo_11, …, T_h.Attributo_hm
from Tabella_1 T_1, …, Tabella_n T_n
where Condizione
```

- corrisponde all'interrogazione in algebra relazionale:

$$\pi_{T\_1.Attributo\_11,\ldots,T\_h.Attributo\_hm}(\sigma_{Condizione}(Tabella\_1 \times \ldots \times Tabella\_n))$$

Esempio: gestione degli esami universitari

Studente

MATR	NOME	CITTA'	CDIP
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf
702	Antonio	Roma	Log

Esame

MATR	COD-CORSO	DATA	VOTO
123	1	7-9-97	30
123	2	8-1-98	28
702	2	7-9-97	20

Corso

COD-CORSO	TITOLO	DOCENTE
1	matematica	Barozzi
2	informatica	Meo

Interrogazione semplice

```
select  *
from Studente
```

MATR	NOME	CITTA'	CDIP
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf
702	Antonio	Roma	Log

Interrogazione semplice

Studente

Matr	Nome	Città	CDip

```
select Nome
from Studente
where CDip = 'Log'
```

interpretazione algebrica (a meno dei duplicati)

$\Pi_{Nome}\ \sigma_{CDip='Log'}$ **Studente**

Sintassi nella clausola `select`

```
select  *
select  Nome, Città
select distinct  Città
select  Città as LuogoDiResidenza
select ReddditoCatastale * 0.05
         as TassaIci
select  sum(Salario)
```


Sintassi della clausola `from`

```
from  Studente
from  Studente as X
from  Studente, Esame
from  Studente join Esame
      on Studente.Matr=Esame.Matr
```

Sintassi della clausola `where`

- Espressione booleana di predicati semplici (come in algebra)
- Alcuni predicati aggiuntivi:
  - **between:**
    Data between 1-1-90 and 31-12-99
  - **like:**
    CDip like 'Lo%'
    Targa like 'MI_777_8%'

Congiunzione di predicati

- Estrarre gli studenti di informatica originari di Bologna:

```
select *
from Studente
where CDip = 'Inf' and
      Città = 'Bologna'
```

- Risultato:

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf

Disgiunzione di predicati

- Estrarre gli studenti originari di Bologna o di Torino:

```
select *
from Studente
where Città = 'Bologna' or
      Città = 'Torino'
```

- Risultato:

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf

Espressioni booleane

- Estrarre gli studenti originari di Roma che frequentano il corso in Informatica o in Logistica:

```
select *
from Studente
where Città = 'Roma' and
      (CDip = 'Inf' or
       CDip = 'Log')
```

- Risultato:

Matr	Nome	Città	CDip
702	Antonio	Roma	Log

Operatore `like`

- Estrarre gli studenti con un nome che ha una 'a' in seconda posizione e finiscono per 'o':

```
select *
from Studente
where Nome like '_a%o'
```

- Risultato:

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf

Duplicati

- In algebra relazionale e nel calcolo, i risultati delle interrogazioni non contengono elementi duplicati
- In SQL, le tabelle prodotte dalle interrogazioni possono contenere più righe identiche tra loro
- I duplicati possono essere rimossi usando la parola chiave `distinct`

Duplicati

```
select
distinct CDip
from Studente
```

CDip
Inf
Log

```
select CDip
from Studente
```

CDip
Inf
Inf
Log

Gestione dei valori nulli

- I valori nulli rappresentano tre diverse situazioni:
  - un valore non è applicabile
  - un valore è applicabile ma sconosciuto
  - non si sa se il valore è applicabile o meno
- SQL-89 usa una logica a due valori
  - un confronto con *null* restituisce FALSE
- SQL-2 usa una logica a tre valori
  - un confronto con *null* restituisce UNKNOWN
- Per fare una verifica sui valori nulli:
  Attributo **`is`** [**`not`**] **`null`**

Predicati e valori nulli

- **logica a tre valori (V,F,U)**

V and U = U
V or U = V

F and U = F
F or U = U

U and U = U
U or U = U
not U = U

- **P =**
  `(Città is not null) and (CDip like 'Inf%')`

Città	CDip
Milano	Inf
Milano	NULL
NULL	Inf
Milano	Log

P	TUPLA SELEZ
V	si
U	no
F	no
F	no

Interrogazioni sui valori nulli

```
select *
from Studente
where Città  is  [not]  null
```

se `Città` ha valore *null*
`(Città = 'Milano')` ha valore Unknown

Interrogazioni sui valori nulli

```
select *
from Studente
where Cdip = 'Inf' or
      Cdip <> 'Inf'
```

è equivalente a:

```
select *
from Studente
where Cdip is not null
```


Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di "Logistica" che hanno preso almeno un 30

```
select Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
  and CDip like 'Lo%' and Voto = 30
```

NOME
Carlo

Join in SQL-2

- SQL-2 ha introdotto una sintassi alternativa per i join, rappresentadoli esplicitamente nella clausola `from`:

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

- *TipoJoin* può essere `inner`, `right` [`outer`], `left` [`outer`] oppure `full` [`outer`], consentendo la rappresentazione dei join esterni
- La parola chiave `natural` può precedere *TipoJoin* (però è implementato di rado)

Join di due tabelle in SQL-2

```
select Nome
from Studente, Esame
where  Studente.Matr = Esame.Matr
     and CDip like 'Lo%' and Voto = 30
```

```
select Nome
from Studente join Esame
     on Studente.Matr = Esame.Matr
where CDip like 'Lo%'and Voto = 30
```


Database d'esempio: guidatori e automobili

DRIVER

FirstName	Surname	DriverID
Mary	Brown	VR 2030020Y
Charles	White	PZ 1012436B
Marco	Neri	AP 4544442R

AUTOMOBILE

CarRegNo	Make	Model	DriverID
ABC 123	BMW	323	VR 2030020Y
DEF 456	BMW	Z3	VR 2030020Y
GHI 789	Lancia	Delta	PZ 1012436B
BBB 421	BMW	316	MI 2020030U

Left join

- Estrarre i guidatori con le loro macchine, includendo i guidatori senza macchine:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver left join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Risultato:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL

Full join

- Estrarre tutti i guidatori e tutte le automobili, mostrando le possibili relazioni tra di loro:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver full join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Risultato:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	NULL	BBB 421	BMW	316

Interrogazione semplice con tre tabelle

- Estrarre il nome degli studenti di "Matematica" che hanno preso almeno un 30

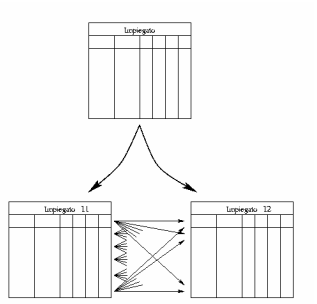
```
select Nome
from Studente, Esame, Corso
where  Studente.Matr = Esame.Matr
  and Corso.CodCorso = Esame.CodCorso
  and Titolo like 'Mat%' and Voto = 30
```

$\Pi_{Nome}\ \sigma_{(Titolo\ like\ 'Mat\%') \wedge (Voto=30)}$ (Studente $\triangleright\triangleleft$ Esame $\triangleright\triangleleft$ Corso)

Variabili in SQL

- Gli alias di tabella possono essere interpretati come variabili, con valore pari a un'intera tabella
- L'uso delle variabili corrisponde all'operatore di ridenominazione ρ dell'algebra relazionale

Variabili in SQL



Interrogazione semplice con variabili relazionali

Chi sono i dipendenti di Giorgio?

Impiegato

Matr	Nome	DataAss	Salario	MatrMgr
1	Piero	1-1-95	3 M	2
2	Giorgio	1-1-97	2,5 M	null
3	Giovanni	1-7-96	2 M	2

Chi sono i dipendenti di Giorgio?

```
select  X.Nome, X.MatrMgr, Y.Matr, Y.Nome
from Impiegato as X, Impiegato as Y
where X.MatrMgr = Y.Matr
  and Y.Nome = 'Giorgio'
```

X.Nome	X.MatrMgr	Y.Matr	Y.Nome
Piero	2	2	Giorgio
Giovanni	2	2	Giorgio

Interrogazioni complesse

Classificazione delle interrogazioni complesse

- Query con ordinamento
- Query con aggregazioni
- Query con raggruppamento
- Query binarie
- Query nidificate

Esempio: gestione ordini

Cliente

CODCLI	INDIRIZZO	P-IVA

Ordine

CODORD	CODCLI	DATA	IMPORTO

Dettaglio

CODORD	CODPROD	QTA

Prodotto

CODPROD	NOME	PREZZO

Istanza di ordine

Ordine

CODORD	CODCLI	DATA	IMPORTO
1	3	1-6-97	50.000.000
2	4	3-8-97	8.000.000
3	3	1-9-97	5.500.000
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
6	3	3-9-97	27.000.000

Ordinamento

- La clausola `order by`, che compare in coda all'interrogazione, ordina le righe del risultato
- Sintassi:

  `order by` *AttributoOrdinamento* [`asc` | `desc`]
  {, *AttributoOrdinamento* [`asc` | `desc`] }

- Le condizioni di ordinamento vengono valutate in ordine
  - a pari valore del primo attributo, si considera l'ordinamento sul secondo, e così via

Query con ordinamento

```
select  *
from Ordine
where Importo > 100.000
order by Data
```

CODORD	CODCLI	DATA	IMPORTO
1	3	1-6-97	50.000.000
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
2	4	3-8-97	8.000.000
3	3	1-9-97	1.500.000
6	3	3-9-97	5.500.000

`order by CodCli`

CODORD	CODCLI	DATA	IMPORTO
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
1	3	1-6-97	50.000.000
6	3	3-9-97	5.500.000
3	3	1-9-97	1.500.000
2	4	3-8-97	27.000.000

`order by CodCli asc, Data desc`

CODORD	CODCLI	DATA	IMPORTO
5	1	1-8-97	1.500.000
4	1	1-7-97	12.000.000
6	3	3-9-97	5.500.000
3	3	1-9-97	1.500.000
1	3	1-6-97	50.000.000
2	4	3-8-97	27.000.000

Funzioni aggregate

- Le interrogazioni con funzioni aggregate non possono essere rappresentate in algebra relazionale
- Il risultato di una query con funzioni aggregate dipende dalla valutazione del contenuto di un insieme di righe
- SQL-2 offre cinque operatori aggregati:
  - `count` cardinalità
  - `sum` sommatoria
  - `max` massimo
  - `min` minimo
  - `avg` media

Operatore `count`

- `count` restituisce il numero di righe o valori distinti; sintassi:
  `count(< * | [ distinct | all ] ListaAttributi >)`
- Estrarre il numero di ordini:
```
select count(*)
from Ordine
```
- Estrarre il numero di valori distinti dell'attributo CodCli per tutte le righe di Ordine:
```
select count(distinct CodCli)
from Ordine
```
- Estrarre il numero di righe di Ordine che posseggono un valore non nullo per l'attributo CodCli:
```
select count(all CodCli)
from Ordine
```


`sum`, `max`, `min`, `avg`

- Sintassi:
  < `sum` | `max` | `min` | `avg` > ([`distinct` | `all`] *AttrEspr*)
- L'opzione `distinct` considera una sola volta ciascun valore
  - utile solo per le funzioni `sum` e `avg`
- L'opzione `all` considera tutti i valori diversi da *null*

Query con massimo

- **Estrarre l'importo massimo degli ordini**

```
select max(Importo) as MaxImp
from Ordine
```

MaxImp
50.000.000

Query con sommatoria

- **Estrarre la somma degli importi degli ordini relativi al cliente numero 1**

```
select sum(Importo) as SommaImp
from Ordine
where CodCliente = 1
```

SommaImp
13.500.000

Funzioni aggregate con join

- Estrarre l'ordine massimo tra quelli contenenti il prodotto con codice 'ABC' :
```
select max(Importo) as MaxImportoABC
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and
      CodProd = 'ABC'
```

Funzioni aggregate e target list

- Query scorretta:

```
select Data, max(Importo)
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and
      CodProd = 'ABC'
```

- La data di quale ordine? La target list deve essere omogenea

Funzioni aggregate e target list

- Estrarre il massimo e il minimo importo degli ordini:

```
select max(Importo) as MaxImp,
       min(Importo) as MinImp
from Ordine
```

MaxImp	MinImp
50.000.000	1.500.000

Query con raggruppamento

- Nelle interrogazioni si possono applicare gli operatori aggregati a sottoinsiemi di righe
- Si aggiungono le clausole
  - **`group by`** (raggruppamento)
  - **`having`** (selezione dei gruppi)

```
select …
from …
where …
group by …
having …
```


Query con raggruppamento

- Estrarre la somma degli importi degli ordini successivi al 10-6-97 per quei clienti che hanno emesso almeno 2 ordini

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*)  >= 2
```


Passo 1: Valutazione `where`

CodOrd	CodCli	Data	Importo
2	4	3-8-97	8.000.000
3	3	1-9-97	5.500.000
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
6	3	3-9-97	27.000.000

Passo 2 : Raggruppamento

- si valuta la clausola **`group by`**

CodOrd	CodCli	Data	Importo
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
3	3	1-9-97	1.500.000
6	3	3-9-97	5.500.000
2	4	3-8-97	8.000.000

Passo 3 : Calcolo degli aggregati

- si calcolano **`sum(Importo)`** e **`count(*)`** per ciascun gruppo

CodCli	sum (Importo)	count (*)
1	13.500.000	2
3	32.500.000	2
4	5.000.000	1

Passo 4 : Estrazione dei gruppi

- si valuta il predicato **`count(*) >= 2`**

CodCli	sum (Importo)	count (*)
1	13.500.000	2
3	32.500.000	2
~~4~~	~~5.000.000~~	~~1~~

Passo 5 : Produzione del risultato

CodCli	sum (Importo)
1	13.500.000
3	32.500.000

Query con group by e target list

- Query scorretta:

```
select Importo
from Ordine
group by CodCli
```

- Query scorretta:

```
select O.CodCli, count(*), C.Città
from Ordine O join Cliente C
     on (O.CodCli = C.CodCli)
group by O.CodCli
```

- Query corretta:

```
select O.CodCli, count(*), C.Città
from Ordine O join Cliente C
     on (O.CodCli = C.CodCli)
group by O.CodCli, C.Città
```


`where` o **`having`**?

- Soltanto i predicati che richiedono la valutazione di funzioni aggregate dovrebbero comparire nell'argomento della clausola `having`
- Estrarre i dipartimenti in cui lo stipendio medio degli impiegati che lavorano nell'ufficio 20 è maggiore di 25:

```
select Dipart
from Impiegato
where Ufficio = '20'
group by Dipart
having avg(Stipendio) > 25
```


Query con raggruppamento e ordinamento

- È possibile ordinare il risultato delle query con raggruppamento

```
select  …..
from …..
[ where ….. ]
group by …..
[ having ….. ]
order by ….
```

Raggruppamento e ordinamento

- Estrarre la somma degli importi degli ordini successivi al 10-6-97 per quei clienti che hanno emesso almeno 2 ordini, in ordine decrescente di somma di importo

```
select  CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
order by sum(Importo) desc
```


Risultato dopo la clausola di ordinamento

CodCli	sum (Importo)
3	32.500.000
1	13.500.000

Doppio raggruppamento

- Estrarre la somma delle quantità dei dettagli degli ordini emessi da ciascun cliente per ciascun prodotto, purché la somma superi 50

```
select CodCli, CodProd, sum(Qta)
from Ordine as O, Dettaglio as D
where  O.CodOrd = D.CodOrd
group by CodCli, CodProd
having sum(Qta) > 50
```


Situazione dopo il join e il raggruppamento

Ordine		Dettaglio			
CodCli	Ordine. CodOrd	Dettaglio. CodOrd	CodProd	Qta	
1	3	3	1	30	gruppo 1,1
1	4	4	1	20	
1	3	3	2	30	gruppo 1,2
1	5	5	2	10	
2	3	3	1	60	gruppo 2,1
3	1	1	1	40	
3	2	2	1	30	gruppo 3,1
3	6	6	1	25	

Estrazione del risultato

- si valuta la funzione aggregata **`sum(Qta)`** e il predicato **`having`**

CodCli	CodProd	sum(Qta)
~~1~~	~~1~~	~~50~~
~~1~~	~~2~~	~~40~~
2	1	60
3	1	95

Query binarie (set queries)

• Costruite concatenando due query SQL tramite operatori insiemistici
• Sintassi:
SelectSQL **{** < **`union`** | **`intersect`** | **`except`** > **[** **`all`** **]** ***SelectSQL*** **}**

- **`union`** unione
- **`intersect`** intersezione
- **`except`** (**`minus`**) differenza

• Si eliminano i duplicati, a meno che non venga usata l'opzione **`all`**

Unione

- Estrarre i codici degli ordini i cui importi superano 500.000 lire oppure presenti in qualche dettaglio con quantità superiore a 1000

```
select CodOrd
from Ordine
where Importo > 500.000
      union
select CodOrd
from Dettaglio
where Qta > 1000
```


Notazione posizionale

```
select Padre
from Paternita
  union
select Madre
from Maternita
```

- Quali nomi per gli attributi del risultato?
  - Nessuno
  - Quelli del primo operando
  - ...

Notazione posizionale

```
select Padre, Figlio
from Paternita
union
select Figlio, Madre
from Maternita
```

```
select Padre, Figlio
from Paternita
union
select Madre, Figlio
from Maternita
```

- Sono interrogazioni diverse

Differenza

- Estrarre i codici degli ordini i cui importi superano 500.000 lire ma non presenti in nessun dettaglio con quantità superiore a 1000

```
select CodOrd
from Ordine
where Importo > 500.000
      except
select CodOrd
from Dettaglio
where Qta > 1000
```

- Può essere rappresentata usando una query nidificata

Differenza

- Estrarre i codici degli ordini che presentano *n* >=1 linee d'ordine con quantità maggiore di 10 e non presentano un numero *m* >= *n* di linee d'ordine con quantità superiore a 1000

```
select CodOrd
from Dettaglio
where Qta > 10
      except all
select CodOrd
from Dettaglio
where Qta > 1000
```


Intersezione

- Estrarre i codici degli ordini i cui importi superano 500.000 lire e che sono presenti in qualche dettaglio con quantità superiore a 1000

```
select CodOrd
from Ordine
where Importo > 500.000
      intersect
select CodOrd
from Dettaglio
where Qta > 1000
```

- Anche in questo caso la query può essere rappresentata usando una query nidificata

Query nidificate

- Nella clausola **`where`** possono comparire predicati che:
  - confrontano un attributo (o un'espressione sugli attributi) con il risultato di una query SQL; sintassi:

    ScalarValue Operator < **`any`** | **`all`** > *SelectSQL*

    - **`any`**: il predicato è vero se almeno una riga restituita dalla query *SelectSQL* soddisfa il confronto
    - **`all`**: il predicato è vero se tutte le righe restituite dalla query *SelectSQL* soddisfano il confronto
    - *Operator*: uno qualsiasi tra =, <>, <, <=, >, >=

Query nidificate

  - usano il quantificatore esistenziale sul risultato di una query SQL; sintassi:

    `exists` *SelectStar*

    - il predicato è vero se la query *SelectStar* restituisce un risultato non vuoto (sempre **`select *`** perché è irrilevante la proiezione)

- La query che appare nella clausola **`where`** è chiamata query nidificata

Query nidificate semplici

- Estrarre gli ordini di prodotti con un prezzo superiore a 100.000

```
select CodOrd
from Dettaglio
where CodProd = any (select CodProd
                     from Prodotto
                     where Prezzo > 100.000)
```

- Equivalente a (senza query nidificata, a meno di duplicati)

```
select CodOrd
from Dettaglio D, Prodotto P
where D.CodProd = P.CodProd
  and Prezzo > 100.000
```


Query nidificate semplici

- Estrarre i prodotti ordinati assieme al prodotto avente codice 'ABC'
  - senza query nidificate:

```
select D1.CodProd
from Dettaglio D1, Dettaglio D2
where D1.CodOrd = D2.CodOrd and
      D2.CodProd = 'ABC'
```

  - con una query nidificata:

```
select CodProd
from Dettaglio
where CodOrd = any
        (select CodOrd
         from Dettaglio
         where CodProd = 'ABC')
```


Negazione con query nidificate

- Estrarre gli ordini che non contengono il prodotto 'ABC':

```
select distinct CodOrd
from Ordine
where CodOrd <> all (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

- In alternativa:

```
select CodOrd
from Ordine
   except
select CodOrd
from Dettaglio
where CodProd = 'ABC'
```


Operatori **`in`** e **`not in`**

- L'operatore `in` è equivalente a `= any`

```
select CodProd
from Dettaglio
where CodOrd in
        (select CodOrd
         from Dettaglio
         where CodProd = 'ABC')
```

- L'operatore `not in` è equivalente a `<> all`

```
select distinct CodOrd
from Ordine
where CodOrd not in (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

Query nidificate

- Estrarre nome e indirizzo dei clienti che hanno emesso qualche ordine di importo superiore a 10.000.000

```
select Nome, Indirizzo
from Cliente
where CodCli in
      select CodCli
      from Ordine
      where Importo > 10000000
```


Query nidificate a più livelli

- Estrarre nome e indirizzo dei clienti che hanno emesso qualche ordine che comprende il prodotto “Pneumatico”

```
select distinct Nome, Indirizzo
from Cliente
where CodCli in
       select CodCli
       from Ordine
       where CodOrd in
              select CodOrd
              from Dettaglio
              where CodProd in
                    select CodProd
                    from Prodotto
                    where Nome = 'Pneumatico'
```


La query equivalente

• La query precedente equivale a:

```
select distinct C.Nome, Indirizzo
from Cliente as C, Ordine as O,
     Dettaglio as D, Prodotto as P
where C.CodCli = O.CodCli
  and O.CodOrd = D.CodOrd
  and D.CodProd = P.CodProd
  and P.Nome = 'Pneumatico'
```


max e **min** con query nidificate

- Gli operatori aggregati `max` e `min` possono essere espressi tramite query nidificate
- Estrarre l’ordine con il massimo importo
  - usando `max`:

```
select CodOrd
from Ordine
where Importo in (select max(Importo)
                  from Ordine)
```

  - con una query nidificata:

```
select CodOrd
from Ordine
where Importo >= all (select Importo
                      from Ordine)
```


Uso di **any** e **all**

```
select CodOrd
from Ordine
where Importo > any
        select Importo
        from Ordine
```

```
select CodOrd
from Ordine
where Importo >= all
        select Importo
        from Ordine
```

COD-ORD	IMPORTO	ANY	ALL
1	50.000	F	F
2	300.000	V	V
3	90.000	V	F

Query nidificate complesse

- La query nidificata può usare variabili della query esterna
  - Interpretazione: la query nidificata viene valutata per ogni tupla della query esterna
- Estrarre tutti i clienti che hanno emesso più di un ordine nella stessa giornata:

```
select CodCli
from Ordine O
where exists (select *
              from Ordine O1
              where O1.CodCli = O.CodCli
                and O1.Data = O.Data
                and O1.CodOrd <> O.CodOrd)
```

Query nidificate complesse

- Estrarre tutte le persone che [non] hanno degli omonimi:

```
select *
from Persona P
where [not] exists
        (select *
         from Persona P1
         where P1.Nome = P.Nome
           and P1.Cognome = P.Cognome
           and P1.CodFisc <> P.CodFisc)
```


Costruttore di tupla

- Il confronto con la query nidificata può coinvolgere più di un attributo
- Gli attributi devono essere racchiusi da un paio di parentesi tonde (costruttore di tupla)
- La query precedente può essere espressa così:

```
select *
from Persona P
where (Nome,Cognome) [not] in
        (select Nome, Cognome
         from Persona P1
         where P1.CodFisc <> P.CodFisc)
```


Commenti sulle query nidificate

- L'uso di query nidificate può produrre query 'meno dichiarative', ma spesso si migliora la leggibilità
- La prima versione di SQL prevedeva solo la forma nidificata (o strutturata) con una sola relazione nella clausola **`from`**, il che è insoddisfacente
- Le sottointerrogazioni non possono contenere operatori insiemistici ("l'unione si fa solo al livello esterno"); la limitazione non è significativa, ed è superata da alcuni sistemi

Commenti sulle query nidificate

- Query complesse, che fanno uso di variabili, possono diventare molto difficili da comprendere
- L'uso delle variabili deve rispettare le regole di visibilità
  - una variabile può essere usata solamente all'interno della query dove viene definita o all'interno di una query che è ricorsivamente nidificata nella query dove è definita
  - se un nome di variabile è omesso, si assume il riferimento alla variabile più vicina

Visibilità delle variabili

- Query scorretta:

```
select *
from Cliente
where CodCli in
          (select CodCli
           from Ordine O1
           where CodOrd = 'AZ1020')
   or CodCli in
          (select CodCli
           from Ordine O2
           where O2.Data = O1.Data)
```

- La query è scorretta poiché la variabile `O1` non è visibile nella seconda query nidificata

Esercizi

- Riprendere le basi di dati per la gestione del personale e degli ordini ed esprimere in SQL le interrogazioni :
  - quali impiegati lavorano in un progetto in cui non lavora il loro manager?
  - quanti ordini ha emesso Paolo?
  - quante candele sono state ordinate il 5/7/97?
  - calcolare per ciascun cliente la somma degli importi di tutti gli ordini
  - estrarre l'ordine di importo più alto

Comandi di modifica in SQL

- Istruzioni per
  - inserimento (**insert**)
  - cancellazione (**delete**)
  - modifica dei valori degli attributi (**update**)
- Tutte le istruzioni possono operare su un insieme di tuple (set-oriented)
- Il comando può contenere una condizione, nella quale è possibile fare accesso a tabelle esterne

Inserimento

- Sintassi:
  insert into *NomeTabella* [(*ListaAttributi*)]
  < **values** (*ListaDiValori*) | *SelectSQL*>
- Usando **values**:

```
insert into Studente
values ('456878', 'Giorgio Rossi',
        'Bologna', 'Logistica')
```

- Usando una query:

```
insert into Bolognesi
  select *
  from Studente
  where Città = 'Bologna'
```


Inserimento

- Usando **values** con *ListaAttributi*:

```
insert into Studente(Matr,Nome,Città,CDip)
values ('456878', 'Giorgio Rossi',
        'Bologna', 'Logistica')
```

- Usando una query con *ListaAttributi*:

```
insert into Bolognesi(Matr,Nome,Città,CDip)
  select Matr, Nome, Città, CDip
  from Studente
  where Città = 'Bologna'
```


Inserimento

- L'ordine degli attributi e dei valori è significativo (notazione posizionale, il primo valore viene associato al primo attributo, e così via)
- Se la *ListaAttributi* viene omessa, si considerano tutti gli attributi della relazione, nell'ordine in cui compaiono nella definizione della tabella
- Se la *ListaAttributi* non contiene tutti gli attributi della relazione, agli attributi rimanenti viene assegnato il valore di default (se definito, altrimenti il valore *null*)

Cancellazioni

- Sintassi:
  delete from *NomeTabella* [**where** *Condizione*]
- Cancellare lo studente con matricola 678678:

```
delete from Studente
  where Matr = '678678'
```

- Cancellare gli studenti che non hanno sostenuto esami:

```
delete from Studente
  where Matr not in (select Matr
                     from Esame)
```


Cancellazioni

- L'istruzione **delete** cancella dalla tabella tutte le tuple che soddisfano la condizione
- Il comando può provocare delle cancellazioni in altre tabelle, se è presente un vincolo d'integrità referenziale con politica **cascade**
- Se si omette la clausola **where**, il comando **delete** cancella tutte le tuple
- Per cancellare tutte le tuple da STUDENTE (mantenendo lo schema della tabella):

```
delete from Studente
```

- Per cancellare completamente la tabella STUDENTE (contenuto e schema):

```
drop table Studente cascade
```

Modifiche

- Sintassi:

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

- Esempi:

```
update Esame
set Voto = 30
where Data = '1-4-97'

update Esame
set Voto = Voto + 1
where Matr = '787989'
```


Modifiche

- Poiché il linguaggio è set-oriented, è molto importante l'ordine dei comandi

```
update Impiegato
  set Stipendio = Stipendio * 1.1
  where Stipendio <= 30
update Impiegato
  set Stipendio = Stipendio * 1.15
  where Stipendio > 30
```

- Se i comandi sono scritti in questo ordine, alcuni impiegati possono ottenere un aumento doppio

Uso di `in` nelle modifiche

• Aumentare di L. 5000 l'importo di tutti gli ordini che comprendono il prodotto 456

```
update Ordine
  set Importo = Importo + 5000
  where CodOrd in
    select CodOrd
    from Dettaglio
    where CodProd = '456'
```


Uso di query nidificate nelle modifiche

• Assegnare a TotPezzi la somma delle quantità delle linee di un ordine

```
update Ordine O
  set TotPezzi =
    (select sum(Qta)
     from Dettaglio D
     where D.CodOrd = O.CodOrd)
```


Aspetti evoluti del DDL

- Creazione di indici
- Gestione di viste
- Autorizzazioni d'accesso
- Vincoli di integrità
- Procedure

Creazione di indici

• Indici: meccanismi di accesso efficiente ai dati (spiegati poi)

```
create index
es.: create index DataIx
       on Ordine(Data)

create unique index
es.: create unique index OrdKey
       on Ordine(CodOrd)
```

Qualità dei dati

- Qualità dei dati:
  - correttezza, completezza, attualità
- In molte applicazioni reali i dati sono di scarsa qualità (5% - 40% di dati scorretti)
- Per aumentare la qualità dei dati:
  - Regole di integrità
  - Manipolazione dei dati tramite programmi predefiniti (procedure e trigger)

Esempio: gestione magazzino

Magazzino

CodProd	QtaDisp	QtaRiord
1	150	100
3	130	80
4	170	50
5	500	150

Riordino

CodProd	Data	QtaOrd

Vincoli di integrità generici

- Predicati che devono essere veri se valutati su istanze corrette (legali) della base di dati
- Espressi in due modi:
  - negli schemi delle tabelle
  - come asserzioni separate

Vincoli d'integrità generici

- La clausola **check** può essere usata per esprimere vincoli arbitrari nella definizione dello schema
- Sintassi:
  check (*Condizione*)
- *Condizione* è ciò che può apparire in una clausola `where` (comprese le query nidificate)
- Es, la definizione di un attributo *Superiore* nello schema della tabella IMPIEGATO:

```
Superiore character(6)
check (Matr like "1%" or
       Dipart = (select I.Dipart
                 from Impiegato I
                 where I.Matr = Superiore))
```


Esempio: definizione di Magazzino

```
create table Magazzino as
  ( CodProd    char(2) primary key,
    QtaDisp    integer not null
               check(QtaDisp >= 0),
    QtaRiord   integer not null
               check(QtaRiord > 10))
```


Asserzioni

- Le asserzioni permettono la definizione di vincoli al di fuori della definizione delle tabelle
- Utili in molte situazioni (es., per esprimere vincoli inter-relazionali di tipo generico)
- Una asserzione associa un nome a una clausola `check`; sintassi:
  create assertion *NomeAsserzione* **check** (*Condizione*)
- Es., la tabella IMPIEGATO deve contenere almeno una tupla:

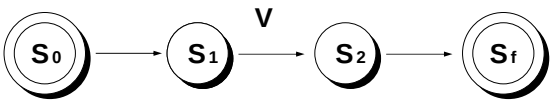
```
create assertion SempreUnImpiegato
  check (1 <= (select count(*)
               from Impiegato))
```

Significato dei vincoli

La verifica dei vincoli può essere:

a immediate (immediata):
la loro violazione annulla l'ultima modifica

b deferred (differita):
la loro violazione annulla l'intera applicazione



Modifica dinamica del significato dei vincoli

• Ogni vincolo è definito di un tipo (normalmente "immediate")
• L'applicazione può modificare il tipo iniziale dei vincoli:
- **`set constraints immediate`**
- **`set constraints deferred`**
• Tutti i vincoli vengono comunque verificati, prima o poi

Viste

- Offrono la "visione" di tabelle virtuali (schemi esterni)
- Classificate in:
  – semplici (selezione e proiezione su una sola tabella)
  – complesse
- Sintassi:
  `create view` *NomeVista* [(*ListaAttributi*)] **`as`** *SelectSQL*
  [**`with`** [**`local`** | **`cascaded`**] **`check option`**]

Esempio di vista semplice

- Ordini di importo superiore a 10.000.000

```
create view OrdiniPrincipali as
       select *
       from Ordine
       where Importo > 10000000
```

Ordine

1	**3**	**1-6-96**	**50.000.000**
4	**1**	**1-7-97**	**12.000.000**
6	**3**	**3-9-97**	**27.000.000**

VISTA : ordini principali

Viste semplici in cascata

```
create view ImpiegatoAmmin
   (Matr,Nome,Cognome,Stipendio) as
select Matr, Nome, Cognome, Stipendio
from Impiegato
where Dipart = 'Amministrazione'
  and Stipendio > 10

create view ImpiegatoAmminJunior as
select *
from ImpiegatoAmmin
where Stipendio < 50
with check option
```


Viste

- Le viste in SQL possono contenere nella definizione altre viste precedentemente definite, ma non vi può essere mutua dipendenza (ricorsione introdotta in SQL-1999)
- Le viste possono essere usate per formulare query complesse
  – Le viste decompongono il problema e producono una soluzione più leggibile
- Le viste sono talvolta necessarie per esprimere alcune query:
  – query che combinano e nidificano diversi operatori aggregati
  – query che fanno un uso sofisticato dell'operatore di unione

Ricorsione in SQL-1999

```
with recursive Raggiungibile (Orig,Dest,Costo) as
( select Orig, Dest, Costo
  from Volo where Orig = 'Milano'
  union
  select V.Orig, R.Dest, V.Costo+R.Costo
  from Volo V join Raggiungibile R
       on V.Dest = R.Orig )

select Orig, Dest, min(Costo)
from Raggiungibile
where Dest = 'San Francisco'
```


Viste e query

- Estrarre il cliente che ha generato il massimo fatturato (senza usare le viste):

```
select CodCli
from Ordine
group by CodCli
having sum(Importo) >= all
          (select sum(Importo)
           from Ordine
           group by CodCli)
```

- Questa soluzione può non essere riconosciuta da tutti i sistemi SQL

Viste e query

- Estrarre il cliente che ha generato il massimo fatturato (usando le viste):

```
create view CliFatt(CodCli,FattTotale) as
select CodCli, sum(Importo)
from Ordine
group by CodCli

select CodCli
from CliFatt
where FattTotale = (select max(FattTotale)
                    from CliFatt)
```


Viste e query

- Estrarre il numero medio di ordini per cliente:
  - Soluzione scorretta (SQL non permette di applicare gli operatori aggregati in cascata):

```
select avg(count(*))
from Ordine
group by CodCli
```

  - Soluzione corretta (usando una vista):

```
create view CliOrd(CodCli,NumOrdini) as
select CodCli, count(*)
from Ordine
group by CodCli

select avg(NumOrdini)
from CliOrd
```


Uso della vista per query

- Vista:

```
create view OrdiniPrincipali as
    select *
    from Ordine
    where Importo > 10000000
```

- Query:

```
select CodCli
from OrdiniPrincipali
```

- Composizione della vista con la query:

```
select CodCli
from Ordine
where Importo > 10000000
```


Modifiche tramite le viste

- Vista:

```
create view OrdiniPrincipali as
  select *
  from Ordine
  where Importo > 10000000
```

- Modifica:

```
update OrdiniPrincipali
  set Importo = Importo * 1.05
  where CodCli = '45'
```

- Composizione della vista con la modifica:

```
update Ordine
  set Importo = Importo * 1.05
  where CodCli = '45'
    and Importo > 10000000
```

Check option

- La **check option** interviene quando viene aggiornato il contenuto di una vista, per verificare che la tupla inserita/modificata appartenga alla vista
- Se l'opzione è **local**, il controllo viene fatto solo rispetto alla vista su cui viene invocato il comando
- Se l'opzione è **cascaded**, il controllo viene fatto su tutte le viste coinvolte
- Es.:

```
create view OrdiniPrinc70 as
  select *
  from OrdiniPrincipali
  where CodCli = 70
  with local check option
```


Check option

- **update OrdiniPrinc70**
  set CodCli = 71
  where CodOrd = '754'

  viene rifiutato con check option **local** e **cascaded**

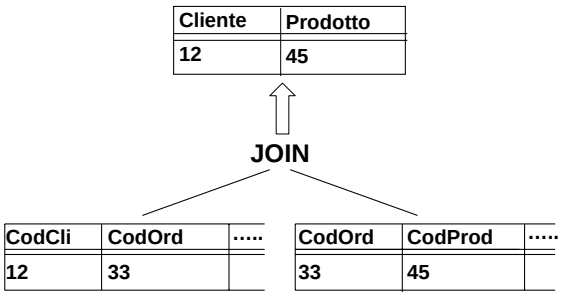
- **update OrdiniPrinc70**
  set Importo = 5000000
  where CodOrd = '754'

  viene accettato dalla **local**, rifiutato dalla **cascaded**

Esempio di vista complessa

```
create view CliProd(Cliente,Prodotto) as
   select CodCli, CodProd
   from Ordine join Dettaglio
     on Ordine.CodOrd = Dettaglio.CodOrd
```


Vista complessa (JOIN)



Interrogazione sulla vista complessa

- Query:

```
select Cliente
from CliProd
where Prodotto = '45'
```

- Composizione della vista con la query:

```
select CodCli
from Ordine join Dettaglio
  on Ordine.CodOrd = Dettaglio.CodOrd
where CodProd = '45'
```

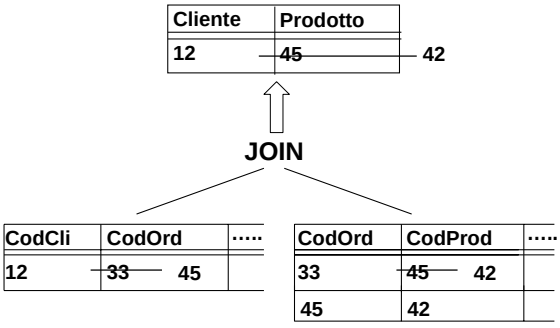

Modifiche sulla vista complessa

- Non è possibile modificare le tabelle di base tramite la vista perché l'interpretazione è ambigua
- Es.:

```
update CliProd
       set Prodotto = '43'
       where Cliente = '12'
```

- Due alternative per la realizzazione sulle tabelle di base
  - il cliente ha cambiato l'ordine
  - il codice del prodotto è cambiato

Vista complessa (JOIN)



Controllo dell'accesso

- Privatezza: protezione selettiva della base di dati in modo da garantire l'accesso solo agli utenti autorizzati
- Meccanismi per identificare l'utente (tramite *parola chiave* o *password*):
  - Quando si collega al sistema informatico
  - Quando accede al DBMS
- Utenti individuali e gruppi di utenti

Autorizzazioni

- Ogni componente dello schema può essere protetto (tabelle, attributi, viste, domini, etc.)
- Il proprietario di una risorsa (il creatore) assegna privilegi (autorizzazioni) agli altri utenti
- Un utente predefinito **`_system`** rappresenta l'amministratore di sistema e ha pieno accesso a tutte le risorse
- Un privilegio è caratterizzato da:
  - la risorsa
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene consentita sulla risorsa
  - la possibilità di passare il privilegio ad altri utenti

Tipi di privilegi

- SQL offre 6 tipi di privilegi
  - **`insert`**: per inserire un nuovo oggetto nella risorsa
  - **`update`**: per modificare il contenuto della risorsa
  - **`delete`**: per rimuovere un oggetto dalla risorsa
  - **`select`**: per accedere al contenuto della risorsa in una query
  - **`references`**: per costruire un vincolo di integrità referenziale che coinvolge la risorsa (può limitare la modificabilità della risorsa)
  - **`usage`**: per usare la risorsa in una definizione di schema (es., un dominio)
- **`all privileges`** li riassume tutti

`grant` e `revoke`

- Per concedere un privilegio a un utente:
  `grant` < *Privilegi* | **`all privileges`** > **`on`** *Risorsa* **`to`** *Utenti* [**`with grant option`**]
  - **`grant option`** specifica se deve essere garantita la possibilità di propagare il privilegio ad altri utenti
- Per revocare un privilegio:
  `revoke` *Privilegi* **`on`** *Risorsa* **`from`** *Utenti* [**`restrict`** | **`cascade`**]

Esempi

```
grant all privileges on Ordine to User1
grant update(Importo) on Ordine to User2
grant select on Ordine to User2, User3

revoke update on Ordine from User1
revoke select on Ordine from User3
```

Esempio di uso, grant option

1 Database administrator

```
grant all privileges on Ordine to User1
  with grant option
```

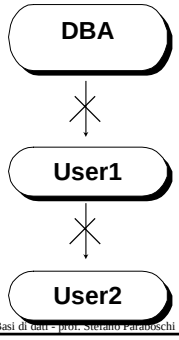
2 User1

```
grant select on Ordine to User2
  with grant option
```

3 User2

```
grant select on Ordine to User3
```


Revoca di un privilegio con cascata



Revoca di un privilegio con cascata

1 Database administrator

```
grant select on Ordine to User1
    with grant option
```

2 User1

```
grant select on Ordine to User2
```

3 Database administrator

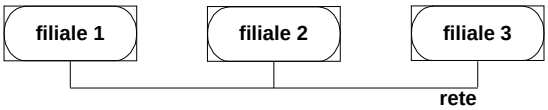
```
revoke select on Ordine from User1 cascade
```


Viste e autorizzazioni di accesso

Viste = unità di autorizzazione

- Consentono la gestione ottimale della privatezza

Esempio: gestione dei conti correnti

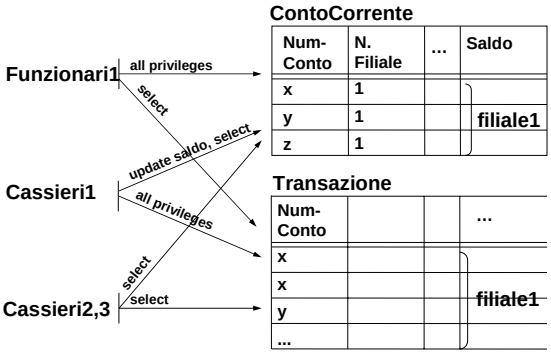


Banca

ContoCorrente(<u>NumConto</u>, Filiale, Cliente, CodFisc, DataApertura, Saldo)

Transazione(<u>NumConto, Data, Progr,</u> Causale, Ammontare)

Requisiti di accesso



Viste relative alla prima filiale

```
create view Conto1 as
( select *
  from ContoCorrente
  where Filiale = 1)

create view Transazione1 as
( select *
  from Transazione
  where NumConto in
        ( select NumConto
          from Conto1 ) )
```


Autorizzazioni relative ai dati della prima filiale

```
grant all privileges on Conto1
        to Funzionari1
grant update(Saldo) on Conto1
        to Cassieri1
grant select on Conto1
        to Cassieri1, Cassieri2, Cassieri3
grant select on Transazione1
        to Funzionari1
grant all privileges on Transazione1
        to Cassieri1
grant select on Transazione1
        to Cassieri2, Cassieri3
```


Funzioni SQL

- Costrutti definiti in SQL-2 che estendono la sintassi delle espressioni nella target list
  - `coalesce`
  - `nullif`
  - `case`
- Vengono valutati per ogni riga estratta dalla query
- Di una certa importanza nella pratica d'uso del linguaggio

`coalesce`

- Sintassi
  - `coalesce` (*list*)
- Descrizione
  - restituisce il primo valore non nullo nella lista
- Esempio:

```
select Nome,
       coalesce(DataNasc,'Non disponibile')
from Persona
```


`nullif`

- Sintassi
  - `nullif` (*input*,*value*)
- Descrizione
  - restituisce il valore nullo se l'espressione *input* ha come valore *value*, altrimenti restituisce *input*
- Esempio:

```
select Nome, nullif(DataNasc,'Ignota')
from Persona
```


`case`

- Sintassi
  - `case when` *Condizione* `then` *Espr*
    { `when` *Condizione* `then` *Espr* }
    `else` *Espr* `end`
- Descrizione
  - restituisce l'espressione associata alla prima condizione soddisfatta
- Esempio:

```
select Nome, case when Voto <= 10 then 'Grav. Ins.'
                  when Voto <= 17 then 'Ins.'
                  else to_char(Voto) end
from Esame
```

Altre funzioni in SQL

- Funzioni matematiche:
  - `abs(), degrees(), exp(), ln(), log(), radians(), round(), sqrt(), trunc(), float(), integer(), acos(), asin(), atan(), cos(), cot(), sin(), tan()`
- Funzioni su stringhe
  - `char_length(), lower(), upper(), substring(), trim(), …`

Altre funzioni in SQL

- Funzioni su dati temporali
  - `now, today` (variabili predefinite)
  - `abstime(), age(), date_part(), date_trunc(), to_char()`

  per `date_part()`e `date_trunc()` la porzione può essere
  - `year, month, day, hour, minute, second, decade, century, millennium, millisecond, microsecond, dow, week, epoch`

Altre funzioni in SQL

- Normalmente i sistemi estendono ulteriormente questo insieme di funzioni
  - Funzioni per la formattazione dell'output
  - Funzioni geometriche
  - Funzioni di accesso ad alcuni servizi del sistema operativo

SQL Embedded

- Le applicazioni spesso richiedono di "rinchiudere" (embed) comandi SQL all'interno delle istruzioni di un linguaggio di programmazione procedurale (C, COBOL, etc.)
- I programmi con SQL embedded usano un precompilatore per gestire i comandi SQL
- I comandi SQL embedded sono preceduti da '$' o 'EXEC SQL'
- Le variabili del programma possono essere usate come parametri nei comandi SQL (precedute da ':')
- I comandi `select` che producono una sola tupla e i comandi di aggiornamento, possono essere usati facilmente
- L'ambiente SQL offre una variabile predefinita **`sqlcode`** che descrive l'esito dei comandi SQL (vale zero se il comando SQL è stato eseguito con successo)

Cursori

- Problema fondamentale: conflitto d'impedenza (impedance mismatch)
  - i linguaggi di programmazione tradizionali gestiscono i record uno alla volta (tuple-oriented)
  - SQL gestisce insiemi di tuple (set-oriented)
- I cursori risolvono questo problema
- Un cursore:
  - accede al risultato di una query in modo set-oriented
  - restituisce le tuple al programma una alla volta
- Sintassi per la definizione dei cursori:
  `declare` *NomeCursore* [**`scroll`**] **`cursor for`** *SelectSQL* [**`for`** < **`read only`** | **`update`** [**`of`** *Attributo* {, *Attributo*}]>]

Operazioni sui cursori

- Per eseguire la query associata a un cursore:
  `open` *NomeCursore*
- Per estrarre una tupla dal risultato della query:
  `fetch` [*Posizione* **`from`**] *NomeCursore* **`into`** *ListaDiFetch*
- Per disallocare il cursore, scartando il risultato della query:
  `close` *NomeCursore*
- Per accedere alla tupla corrente (quando un cursore accede a una tabella, nell'ambito di un comando di modifica) :
  `current of` *NomeCursore* (nella clausola `where`)

Esempio di SQL embedded

```
void MostraStipendiDipart(char NomeDip[])
{
  char Nome[20], Cognome[20];
  long int Stipendio;

$ declare DipImp cursor for
    select Nome, Cognome, Stipendio
    from Impiegato
    where Dipart = :NomeDip;
$ open DipImp;
$ fetch DipImp into :Nome, :Cognome, :Stipendio;
  printf("Dipartimento %s\n",NomeDip);
  while (sqlcode == 0)
  {
    printf("Nome: %s %s ",Nome,Cognome);
    printf("Stipendio: %d\n",Stipendio);
$   fetch DipImp into :Nome, :Cognome, :Stipendio;
  }
$ close DipImp;
}
```


SQL Dinamico

- SQL dinamico serve quando le applicazioni non possono sapere al momento della compilazione quale sarà il comando SQL da eseguire
- Problema principale: gestire il trasferimento di parametri tra il programma e l'ambiente SQL
- Per l'esecuzione diretta:
  `execute immediate` *ComandoSQL*
- Se l'esecuzione è preceduta da un'analisi del comando:
  `prepare` *NomeComando* **`from`** *ComandoSQL*
  - seguito da:
    `execute` *NomeComando* [**`into`** *TargetList*]
    [**`using`** *ListaParametri*]

ODBC/JDBC

- Verranno descritti più avanti
- Sono soluzioni per SQL dinamico in contesto OO

ODBC per linux:

```
Connection* conn;
conn = DriverManager::getConnection("prova","test","c42");
Statement* sqlIstr=conn->createStatement();
ResultSet* risSet=sqlIstr->executeQuery
          ("select Nome,Cognome from Persona");
ODBCXX_STRING NomeC,CognomeC;
while (risSet->next())
{ NomeC = risSet->getString(1);
  CognomeC = risSet->getString(2); }
```


Procedure

- Moduli di programma che svolgono una specifica attività di manipolazione dei dati
- SQL-2 permette la definizione di procedure (anche note come *stored procedures*), ma solo in forma molto limitata
- La maggior parte dei sistemi offrono delle estensioni che permettono di scrivere procedure complesse (es., Oracle PL/SQL)
- Due momenti:
  - dichiarazione (DDL)
  - invocazione (DML)
- Con architettura client-server sono:
  - invocate dai client
  - memorizzate ed eseguite presso i server

Esempio : prelievo dal magazzino

Magazzino

CodProd	QtaDisp	QtaRiord
1	150	100
3	130	80
4	170	50
5	500	150

Riordino

CodProd	Data	QtaOrd

Specifica

- L'utente indica un prelievo dando il codice del prodotto e la quantità da prelevare
- Se la quantità disponibile in magazzino non è sufficiente la procedura si arresta con una eccezione
- Viene eseguito il prelievo, modificando la quantità disponibile in magazzino
- Se la quantità disponibile in magazzino è inferiore alla quantità di riordino si predispone un nuovo ordine d'acquisto

Interfaccia

```
procedure Prelievo
    ( Prod   integer,
      Quant  integer )
```

Invocazione

```
Prelievo(4,150)
```

Stato iniziale nella base di dati

CodProd	QtaDisp	QtaRiord
4	170	50

Realizzazione della procedura

1. Dichiarazione variabili
2. Lettura dello stato
3. Se la quantità disponibile è insufficiente: eccezione
4. Aggiornamento dello stato
5. Se la nuova quantità disponibile è inferiore alla quantità di riordino: emissione di un ordine

Procedura

```
procedure Prelievo (Prod integer, Quant integer) is
begin
 Q1, Q2 integer;
 X exception;
 select QtaDisp, QtaRiord into Q1, Q2
   from Magazzino
   where CodProd = Prod;
 if Q1 < Quant then raise(X);
 update Magazzino
   set QtaDisp = QtaDisp - Quant
   where CodProd = Prod;
 if Q1 - Quant < Q2 then
  insert into Riordino
    values(Prod,sysdate,Q2)
end;
```


Esempio di invocazione

Prelievo(4,150)

Prod=4, **Quant**=150

```
select QtaDisp, QtaRiord into Q1, Q2
    from Magazzino
    where CodProd = Prod;
```

CodProd	QtaDisp	QtaRiord
4	170	50

Q1 = 170, **Q2** = 50

Invocazione (continua)

if Q1 < Quant then raise(X) non scatta

```
update Magazzino
set QtaDisp = QtaDisp - Quant
where CodProd = Prod
```

CodProd	QtaDisp	QtaRiord
4	20	50

Q1 - Quant < Q2 è vero:

```
  insert into Riordino
     values(Prod, sysdate, Q2)
```

CodProd	Data	QtaRiord
4	10-10-97	50

Problemi del progetto di procedure

- Decomposizione modulare delle applicazioni
- Aumento di:
  - efficienza
  - controllo
  - riuso
- Aumenta la responsabilità dell'amministratore della base di dati (rispetto al programmatore applicativo)
- Si sposta "conoscenza" dalle applicazioni allo schema della base di dati (indipendenza di conoscenza)