

# SQL

## (Structured Query Language)

**Eugenio Di Sciascio**

# SQL come DDL e DML

- SQL non è solo un linguaggio di interrogazione
- Linguaggio di **definizione e manipolazione dati**
- **DDL** (*Data Definition Language*): consente di definire lo schema: tabelle, vincoli, domini, viste, ecc.
- **DML** (*Data Manipulation Language*): permette la modifica e l'interrogazione dell'istanza di una base di dati

# SQL come standard

- Varie versioni con successivi miglioramenti:
  - SQL-1 o SQL-89
  - SQL-2 o SQL-92 (entry, intermediate e full)
  - SQL-3 o SQL-99

# Definizione dei domini elementari

- 6 gruppi di domini elementari. Da essi possono costruirsi nuovi domini.
- **Carattere**: consente di definire singoli caratteri oppure stringhe

```
character [varying] [(nro_char)] [character set  
nome_set]
```

Es.(con uso della forma compatta):

Codice\_fiscale **char**(13),

prodotto\_greco **varchar**(100) **character set** Greek

## Definizione dei domini elementari (2)

- **Bit**: introdotto in SQL2 assume valori 0 e 1

**bit** [varying] [ (nro\_bit) ]

Es.(con uso della forma compatta):

Sequenza **bit(5)**

Codice **varbit(16)**

# Definizione dei domini elementari (3)

- **Tipi numerici esatti:** consentono la rappresentazione di numeri interi o in virgola fissa

- `integer`
- `smallint`
- `decimal [ (precision [ , scale ] ) ]`
- `numeric [ (precision [ , scale ] ) ]`

**precision:** n.ro di cifre significative

**scale:** n.ro cifre dopo la virgola

# Definizione dei domini elementari (4)

Es.:

`dato_vendita numeric(6,2)`

consente di rappresentare i valori tra  $-9999,99$  e  $+9999,99$ .

**Differenze: integer e smallint** non consentono di controllare la *precision* (dettata dal sistema di calcolo), ma *precision* di integer  $\geq$  di quella di smallint. Tra **numeric e decimal**: la *precision* di numeric è un valore esatto, il minimo per decimal.

# Definizione dei domini elementari (5)

- **Tipi numerici approssimati:** consentono la rappresentazione di numeri in virgola mobile
  - `Float [ (precision) ]`
  - `Real`
  - `Double precision`

Es. 1.6E12

A float può essere assegnata esplicitamente una precision, i.e. il n.ro di cifre della mantissa. Per gli altri due la precision dipende dal sistema di calcolo, ma la precision di `double precision`  $\geq$  `real` (normalmente doppia).

# Definizione dei domini elementari (6)

- **Data e ora**: consentono di rappresentare istanti di tempo.
  - **Date**
  - **Time** [ (precision) ] [with time zone]
  - **Timestamp** [ (precision) ] [with time zone]

Sono strutturati: **date:year-month-day (yyyy:mm:dd)**;  
**time:hour-min-sec (HH:MM:SS)**;

Default di precision: 0 (s) per time, 6 per timestamp ( $\mu$ s)

Time zone fa riferimento all'ora di Greenwich

# Definizione dei domini elementari (7)

- **Time intervals**: consentono di rappresentare intervalli di tempo
- **Interval** FirstTimeUnit [to LastTimeUnit]
- Specifica un intervallo di tempo cioè un valore relativo utilizzabile per incrementare/decrementare un valore di `date`, `time` o `timestamp`.

Es.

durata **interval year(5) to month**

permette di rappresentare intervalli temporali fino a 99.999 anni e 11 mesi

durata **interval day(4) to second (6)**

permette di rappresentare intervalli temporali fino a 9.999 giorni, 23 ore, 59 minuti e 59,999999 secondi, con una precisione al milionesimo di sec.

# Definizione di nuovi domini

- E' possibile definire nuovi domini a partire da quelli elementari, alternativamente è possibile dichiarare il dominio ed usarlo (peggiora la leggibilità e la modificabilità)
- `Create domain DomainName as DataType [DefaultValue][Constraint]`

Es.:

```
create domain copie_vendute as smallint default 0
```

# Specifica dei valori di default

- **Default:** valore assunto da un attributo in assenza di specificazione
- `Default <Generic|user|null>`
- *Generic* è un valore scelto (purché nel dominio), *user* è l'ID dell'utente che inserisce l'update, *null* è il default generico.

# Definizione dello schema

- E' possibile definire lo schema di una base di dati. Lo schema sarà costituito da un insieme di *domini, tabelle, indici, asserzioni, viste e autorizzazioni*.
- `Create schema [SchemaName]`  
`[[authorization] Authorizedname]`  
`{DefiningElements}`
- Nota: i DefiningElements possono essere definiti successivamente

Es.:

```
create schema azienda authorization antonio
```

# Definizione delle tabelle

- Una tabella viene definita specificando una collezione ordinata di attributi e un insieme di vincoli
- `Create table RelationName`  
(AttributeName Domain [DefaultValue][constraints]  
{, AttributeName Domain [DefaultValue][constraints]}  
FurtherConstraints)

Es.:

```
create table Dipartimento  
  
  (IdDip integer primary key,  
  nome varchar(30) not null,  
  indirizzo varchar(50))
```

# Vincoli di integrità

- Descrivono le condizioni che ciascuna istanza di una relazione deve sempre soddisfare
- Distinguiamo:
  - vincoli **intra-relazionali** che operano su una relazione
  - **inter-relazionali** in cui il predicato opera su più di una relazione

# Vincoli intra-relazionali

Proprietà che devono essere verificate da ogni istanza del DB

- **Vincoli di dominio**: Valgono ovviamente i vincoli di dominio; i dati inseriti devono essere sempre del tipo corretto
- **not null**: indica che il valore null non è ammesso per l'attributo che quindi va sempre specificato a meno che non sia previsto un default diverso da null

Es.:

```
nome char(30) not null default pippo
```

# Vincoli intra-relazionali (2)

- **Primary key**: specifica la chiave primaria che è *necessariamente* unica; i valori degli attributi costituenti non possono, ovviamente, essere null.
- **Primary key**(AttributeName{, AttributeName })

Es.:    Nome varchar(30) ,  
          Cognome varchar(30) ,  
          Dipart varchar(15),  
          Stipendio numeric(9) default 0,  
          **primary key** (Cognome, Nome)

# Vincoli intra-relazionali (3)

- **unique**: impone l'unicità degli attributi cui il vincolo è applicato. In pratica definisce una chiave candidata, ma non scelta come primaria.
- **Unique(AttributeName {, AttributeName })**

Es.: nome varchar(30) not null,  
cognome varchar(30) not null,  
CF char(13) primary key,  
matricola int **unique**

# Vincoli intra-relazionali (4)

Es.: nome char(30) not null,  
cognome char(30) not null,  
**unique**(nome,cognome)

N.B. la seguente dichiarazione non è equivalente a quella del primo esempio, perché?

nome char(30) not null **unique**,  
cognome char(30) not null **unique**

# Vincoli intra-relazionali (5)

- Reazione alla violazione di vincoli **intra-relazionali**: il sistema che rilevi violazioni di vincoli reagirà rifiutando il comando di aggiornamento e segnalando la violazione all'utente

# Vincoli inter-relazionali

- Stabiliscono i *vincoli di integrità referenziale*.
- vincolo che crea un *legame* tra i valori di un attributo della tabella corrente R (**interna**) e i valori di un attributo di un'altra tabella S (**esterna**).

```
foreign key (AttributeName{,AttributeName}) references  
TableName(AttributeName{,AttributeName})
```

- I campi devono essere una chiave (primaria, possibilmente) di S;
- Nelle tuple di R i valori dei campi devono corrispondere a valori in qualche tupla di S o essere null.

N.B. La corrispondenza tra attributi avviene in base all'ordinamento espresso nella dichiarazione.

# Vincoli inter-relazionali (2)

Es.:

**Create table** lavora\_su

(id\_imp int,

pnum int,

ore\_lav decimal(3,1),

**primary key** (id\_imp,pnum)

**foreign key**(id\_imp) **references** impiegato(imp\_num),

**foreign key**(pnum) **references** progetto(id\_prog)

)

# Vincoli inter-relazionali (3)

- A seguito di violazioni nella tabella interna R la reazione è quella standard: l'azione viene rifiutata (*no action*).
- SQL consente però di definire tipi diversi di reazione a violazione di vincoli inter-relazionali quali cancellazione di una tupla riferita o modifica di una primary key riferita.
- A seguito di una azione nella tabella S esterna la reazione avviene nell'ambito della tabella R interna.

NOTA: L'idea base è fornire un modo per adeguare una tabella che sia in associazione con un'altra a seguito di modifiche in quest'ultima.

Ad es., con riferimento al ben noto schema Azienda: cosa accade quando venga cancellata una tupla in Dipartimento alle tuple di Impiegati che si riferiscono ad essa?

# Vincoli inter-relazionali (4)

- Posso introdurre ulteriori specifiche di reazione che riducono la forza del vincolo
- La specificazione di reazione va esplicitata subito dopo il vincolo di integrità:
  - Aggiornamento `on update`
  - Cancellazione `on delete`

# Vincoli inter-relazionali (5)

- Su un aggiornamento (*on update*) o cancellazione (*on delete*) posso specificare le azioni da intraprendere:
  - **cascade**
  - **set null**
  - **set default**
  - **no action**

# Vincoli inter-relazionali (6)

- Esplicitando le reazioni per la richiesta di cancellazione:
  - `on delete {cascade| set null| set default| no action }`

**Cascade:** tutte le tuple della tabella R interna corrispondenti alla tupla cancellata in S vengono cancellate

**Set null:** il valore cancellato nella tabella S viene sostituito con il valore null nell'attributo della tabella R

**Set default:** il valore cancellato nella tabella S viene sostituito con un valore di default nell'attributo di R

**No action:** la cancellazione viene inibita

# Vincoli inter-relazionali (7)

- Esplicitando le reazioni per la richiesta di modifica:
- `on update (cascade | set null | set default | no action )`

**Cascade:** il nuovo valore dell'attributo della tabella esterna S viene riportato su tutte le corrispondenti righe della tabella interna

**Set null:** il valore modificato nella tabella S viene sostituito con il valore null nell'attributo di R

**Set default:** il valore modificato nella tabella S viene sostituito con un valore di default nell'attributo di R

**No action:** la modifica viene inibita

**N.B.:** Cosa succede se non specifico “on delete” o “on update”?

# Esempio

Create table Impiegato

(Nome varchar(20) not null,

Cognome varchar(30) not null,

CF char(16) unique,

ID\_imp smallint primary key,

**Dnum** smallint not null default 0,

Capo smallint **references** Impiegato(ID\_imp) **on delete set null on update cascade,**

Constraint dip\_app

**Foreign key** (Dnum) **references** Dipartimento(ID\_dip) **on delete set default on update cascade);**

**Constraint:** Dichiarazione esplicita di vincolo, consente di cancellare o modificare il vincolo identificato

Create table lavora\_su

(Imp smallint not null,

Prog integer not null,

Ore\_lav decimal(3,1) not null,

Primary key (Imp,Prog),

**Foreign key** (Imp) **references** Impiegato(ID\_imp),

**Foreign key** (Prog) **references** Progetto(ID\_prog))

# Ulteriori specificazioni di vincoli

- La clausola **check**: `check` (Condition)
- Condition corrisponde alle condizioni che possono essere specificate;
- La condizione deve essere sempre verificata per mantenere l'integrità del DB.

```
Es.:  
Create table esami_sup  
(  
....  
Voto integer,  
...  
Check(voto >= 18 and  
voto <= 30)  
)
```

# Modifica degli schemi relazionali

- Le modifiche possono consistere in alterazioni e cancellazioni di schemi e domini.
- Per **modifiche** si usa il comando **Alter**

```
Alter domain DomainName <  
    set default DefaultValue|  
    drop default |  
    add constraint ConstraintDefinition |  
    drop constraint ConstraintName>
```

# Modifica degli schemi relazionali (2)

```
Alter table TableName <  
    alter column ColumnName <  
        set default DefaultValue|  
        drop default)>|  
    add constraint ConstraintDefinition|  
    drop constraint ConstraintName|  
    add column ColumnName |  
    drop column ColumnName >
```

**Es. Alter table** *Impiegato* **add column** *telefono* *char(20)*  
**Alter table** *Dipartimento* **drop column** *città* *varchar(20)*

# Modifica degli schemi relazionali (3)

- Per cancellazioni si usa il comando **Drop**

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

Es.: **Drop table *Progetto* cascade**

# Modifica degli schemi relazionali (4)

**Restrict** specifica che il comando non deve essere eseguito in presenza di oggetti non vuoti

- Per uno **schema**: richiede che lo schema sia vuoto
- Per una **tabella**: richiede che sia vuota e non abbia vincoli esterni
- Per un **dominio** che esso non sia presente in alcuna tabella

Restrict è l'opzione di default.

# Modifica degli schemi relazionali (5)

**Cascade** supera le limitazioni precedenti ed esegue una cancellazione in cascata (va quindi usato con maggiore giudizio!!), tutti gli oggetti specificati devono essere rimossi

- Per uno **schema** effettua una cancellazione completa;
- Per un **dominio** cancella la definizione, ma gli attributi rimangono definiti secondo il dominio elementare di origine.
- Per una **tabella** tutte le righe vengono perse e se la tabella compariva in qualche definizione di tabella o vista, anche queste vengono rimosse
- Per una **vista** elimina tutte le tabelle che compaiono nella definizione.

Es. DoppioNome char(50) → Drop domain DoppioNome cascade

Tutti gli attributi definiti su quel dominio assumeranno direttamente il dominio char(50)

# Controllo degli accessi e sicurezza

- La sicurezza dei dati è uno dei principali problemi da affrontare nella gestione di un DBMS.
- SQL fornisce vari meccanismi di protezione dei dati e degli accessi agli stessi.
- Ogni utente è identificato univocamente (account di S.O. o indipendente, proprio del DBMS).
- Il DBA è il gestore del sistema e ha un account privilegiato; egli ha in genere le responsabilità di concedere/revocare privilegi sia a livello di account che di dati.

# Controllo degli accessi e sicurezza (2)

- Due classi di meccanismi di sicurezza: discrezionali e obbligatori.

- **Discrezionali**: concedere e revocare privilegi, tipicamente la capacità di accedere a specifiche risorse.

- **Obbligatori**: implementare meccanismi multilivello mediante classificazione degli utenti e dei dati (es. Bell-LaPadula).

# Controllo degli accessi .. (3)

- Al creatore di una risorsa (tipicamente il DBA) il sistema fornisce automaticamente tutti i privilegi relativi ad essa. Questi può concederli ad altri.
- Vengono forniti due comandi per concedere o revocare diritti, **grant** e **revoke**. I privilegi di creazione e distruzione non sono cedibili.

```
Grant Privileges on Resource to User_id [with grant option]
```

```
Revoke Privileges on Resource from User_id [restrict|cascade]
```

# Controllo degli accessi

La Grant option permette di estendere in cascata i privilegi ottenuti ad altri utenti

Grant update on Impiegato to Giacomo

Grant select on Impiegato to Antonio **with grant option**

Grant **all privileges** on Dipartimento to Pietro, Paolo

Funziona come una wild card, estende tutti i privilegi disponibili: insert, update, delete, select, ecc.

# Controllo degli accessi .. (5)

Revoke update on Impiegato from Giacomo **cascade**

Cascade propaga la revoca dei privilegi, e tutti gli elementi del DB costruiti sfruttando tali privilegi vengono rimossi.  
Restrict (default) inibisce la revoca se i privilegi erano stati estesi (grant option)

Revoke **grant option** on Impiegato from Antonio

# Controllo degli accessi e sicurezza (6)

- Il meccanismo di gestione degli accessi **Bell-Lapadula** è basato sulla classificazione di soggetti (*utenti*) ed oggetti (*dati*).
- Si stabiliscono *livelli di sicurezza* relativi a soggetti e oggetti, che una volta implementati vanno comunque rispettati (non sono a discrezione dell'amministratore).

– **Unclassified**

– **Classified**

– **Secret**

– **Top secret**



# Controllo degli accessi e sicurezza (7)

- Tale meccanismo di gestione degli accessi si basa su due proprietà:

- **Simple property**

- Un soggetto (utente) può leggere oggetti purchè abbiano una clearance *maggiore o uguale* di quella dell'oggetto:

$$\text{clearance (s)} \geq \text{clearance (o)}$$

N.B.: un soggetto  $s$  di tipo unclassified quali oggetti  $o$  può leggere?

un soggetto  $s$  di tipo top secret quali tipi di documenti può leggere?

# Controllo degli accessi e sicurezza (8)

## – Star property

- Posso accedere in scrittura ad oggetti (documenti) la cui clearance sia *maggiore uguale* di quella del soggetto (utente):
  - $\text{clearance (o)} \geq \text{clearance (s)}$

N.B: Ciò è fatto per evitare la propagazione di informazioni a livello più basso: un soggetto top secret non deve poter scrivere oggetti di tipo unclassified, perché?

# Ulteriori specificazioni di vincoli (2)

- **Assertzioni (SQL2):** consentono di definire vincoli al di fuori della definizione di tabelle o domini. Utili in particolare per definire vincoli su più tabelle

```
Create assertion AssertionName check(condition)
```

*Es.: Devono esserci almeno 2 dipartimenti*  
create assertion almenodue  
check(2 <=  
(select count(\*)  
From Dipartimento))

*Es.: Non devono esserci più di 10 tra dipartimenti ed istituti*  
create assertion nontroppi  
check(10 >=  
(select count(\*)  
From Dipartimento) +  
(select count(\*)  
From Istituto))

# Definizione di viste

- Vista o relazione derivata: tabella virtuale il cui contenuto è definito a partire da altre tabelle nello schema. In pratica è una relazione non costituita da tuple, ma da una definizione.
- Si definiscono associando un nome ed una lista di attributi al risultato di una query. In quest'ultima possono apparire altre viste, ma non ricorsive.

```
Create view ViewName [ (AttributeList) ] as  
selectSQL [with[local | cascaded] check option]
```

```
Createview  
info_dip(nome_dip,no_imp,salario_tot  
As select nome_dip, count(*), sum(salario)  
From impiegato I, dipartimento D  
Where I.Id_dip=D.Id_dip  
Group by D.nome_dip;
```

Gli attributi nella lista devono essere in corrispondenza 1 a 1 con le colonne prodotte dalla query, oppure la vista li eredita dalla query.

```
Createview sforzo_progettuale  
as  
Select nome, cognome,  
prog_name, ore_lav  
From impiegato I, progetto P,  
lavora_su L  
Where I.id_imp=P.Id_imp and  
P.Id_prog=L.id_prog
```

# Query utilizzando viste

- Le viste possono essere utilizzate per formulare query complesse e non esprimibili con altri costrutti, tipicamente quando si vogliono utilizzare operatori aggregati in cascata

Es.: Determinare il dipartimento che spende di più in stipendi:

```
Create view budget_dipart(dip, stip_tot) as  
Select Id_dip, sum(stipendio)  
From Impiegato  
Group by Id_dip;
```

```
Select Id_dip  
From budget_dipart  
Where stip_tot=(select max(stip_tot)  
From budget_dipart)
```

# Aggiornamenti sulle viste

L'aggiornamento della viste può essere problematico. Esso comporta in pratica l'aggiornamento delle tabelle sottostanti la vista, può essere ambiguo o addirittura impossibile.

Ha senso modificare il valore di `stip_tot` in una tupla della vista precedente?

E quali righe della relazione di partenza impatterebbe la modifica??

SQL92 consente l'update solo per viste determinate a partire da tabelle singole senza funzioni aggregate. Cioè quando ogni tupla della vista mappa una tupla della relazione di partenza.

L'opzione `with check option` è necessaria quando si preveda l'aggiornamento di una vista, indicando che un update deve far sì che le tuple risultanti appartengano ancora alla vista (non violino i predicati di selezione).

Per viste ottenute da altre viste, `Local` e `cascaded` specificano, rispettivamente, se il controllo vada effettuato solo al livello della vista presente o debba propagarsi. Il default è `cascaded`.

# L'implementazione delle viste

- Due strategie base:
- Query modification - La vista è creata effettuando una query sulla base di dati.
  - Problemi: inefficienza (bisogna eseguire le query al volo.)
- View Materialization - Viene creata una tabella temporanea con i dati disponibili al momento della creazione.
  - Problemi: è necessario sviluppare una strategia di aggiornamento a seguito di modifiche nel DB sottostante per tenere la vista aggiornata

# Viste Ricorsive

- SQL-3 (1999) a differenza delle versioni precedenti supporta le viste ricorsive utilizzando la clausola `with recursive`.
- Utile quando si vogliono risolvere interrogazioni del tipo “Trova tutti i superiori di un impiegato”, non risolubile altrimenti poiché non è noto a priori il numero di join richiesti.

# Creazione di Indici

- La sintassi per gli indici non è standardizzata. Una sintassi “tipica” per crearne uno è:

```
create [unique] index IndexName on  
TableName (Attributes_list)
```

Per eliminare un indice: drop Index *IndexName*

*Es: create index CittadinanzaIDX on Impiegato(città-residenza)*

# Funzioni di utilità

- SQL mette a disposizione varie funzioni di utilità, alcune sono disponibili con sintassi diverse in dipendenza del produttore, alcune di esse sono non standard.
- Gestione tempo: `current_date`, `current_time`, `current_timestamp`, `extract`, `age`...più molte altre

**Es.:** `SELECT CURRENT_DATE, EXTRACT( YEAR FROM  
Impiegato.data_nascita ) FROM Impiegato`

- Manipolazione stringhe e tipi: `CHARACTER_LENGTH`, `POSITION`, `SUBSTRING`, `CAST`.
- **Es.:** `Cast (data_nascita as char(20))`

# Funzioni di utilità

- Funzioni condizionali:
- **Case: espressioni condizionali**

```
case when condizione then espressione  
{ when condizione then espressione }  
else Espressione  
End
```

## **Esempio**

```
Update impiegato  
Set stipendio =  
Case  
When ID_Dip=1 then stipendio *1.2  
When ID_Dip=2 then stipendio *1.15  
Else stipendio  
end
```

# Funzioni di utilità

- Coalesce e nullif (comportamento duale) la prima sostituisce a null un valore predefinito, la seconda a un valore predefinito null.

Es:

```
Select città, coalesce(temperatura, 'non pervenuta')  
From dati_meteo
```

```
Select città, nullif (temperatura, 'non pervenuta')  
From dati_meteo
```

- **Nota:** molte altre funzioni disponibili...