Laboratorio di Basi di Dati

Turni T3 e T4

a.a. 2018/2019

Ruggero Pensa - Fabiana Vernero

In questa lezione

- SQL come DML:
  - Inserimenti, modifiche, cancellazioni
  - Common table expressions
- SQL come DDL:
  - Vincoli di integrità generici
  - Viste
  - Transazioni

Database di esempio

S

SNum	SName	Status	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

S=Supplier (fornitore)
P=Parts (parti)
QTY=quantity

P

PNum	PName	Color	Weight	City
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

SNum	PNum	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

nut=dado, bolt=bullone, screw=vite, cam=camma, cog=ruota dentata

Modifica dei dati in SQL

- Il DML (Data Manipulation Language) offre sia i comandi per l'interrogazione dei dati (**`select`**) sia i comandi per la modifica dei dati (inserimenti, aggiornamenti e cancellazioni).
- Si hanno a disposizione tre comandi:
  - **`insert`** (per gli inserimenti)
  - **`delete`** (per le cancellazioni)
  - **`update`** (per gli aggiornamenti)

Inserimento di dati - 1

- Per inserire nuove righe in una tabella si usa il comando **`insert`**.
  - Sintassi:

    `insert into` *Tabella*(*Attributo*$_1$, …, *Attributo*$_n$) **`values`** (*ValoreAttributo*$_1$, …, *ValoreAttributo*$_n$);

  - Inserisce nella tabella singole righe assegnando Attributo$_1$ = ValoreAttributo$_1$, ecc.

Inserimento di dati - 2

- Gli attributi omessi assumono il valore di default o **`null`**.
  - Se l'attributo non è nullable e non ha default, il DBMS segnala l'errore e annulla l'inserimento.
- Se si specificano i valori per tutte le colonne, la lista di attributi può essere omessa.

Inserimento di dati - 3

- **Esempio**: inserire la riga <'*S6*', '*Alice*', *40*, '*Turin*'> nella tabella S.

```
insert into S(SNum, SName, Status, City) values
('S6', 'Alice', 40, 'Turin');
```

oppure

```
insert into S values ('S6', 'Alice', 40, 'Turin');
```

oppure

```
insert into S(SNum, SName, City) values ('S6',
'Alice', 'Turin');
```

- Sono tutti validi (nell'ultimo caso **`Status`** sarà impostato a **`null`**.

Inserimento di dati - 4

- Per inserire più righe contemporaneamente in una tabella a partire da un'altra tabella si usa sempre **`insert`.**
  - Sintassi:

    ```
    insert into Tabella1(ListaAttributi)
      (select EspressioneAttributi from
    ListaTabelle2 where … );
    ```

  - `ListaAttributi` è opzionale e deve essere coerente con `EspressioneAttributi`.
  - La **`select`** può essere di qualsiasi tipo.
  - Gli attributi omessi prendono il valore di default o **`null`**.
    - Se un attributo non è nullable e non ha default, il DBMS segnala l'errore e annulla l'inserimento.

Inserimento di dati - 5

- **Esempio:** inserire dati in una tabella a partire da un'altra
  - Sono tutti validi:

```
insert into P_London(PNum, PName, Color, Weight)
  ( select PNum, PName, Color, Weight
    from P
    where City='London');

insert into P_Copia
  ( select * from P );

insert into P_CopiaTokyo
  ( select PNum, PName, Color, Weight, 'Tokyo'
    from P );

insert into P_Libbre
  ( select PNum, PName, Color, Weight/0.45, City
    from P );
```

- Le tabelle devono già esistere nel database: non vengono create automaticamente.

Cancellazione di dati - 1

- Per cancellare condizionatamente delle righe da una tabella si usa il comando **`delete`**.
  - Sintassi

    ```
    delete from Tabella where Condizione;
    ```

  - Cancella tutte le righe in **`Tabella`** per cui **`Condizione`** è vera.
  - **`Condizione`** può essere anche un predicato con una sottointerrogazione.

Cancellazione di dati - 2

- Per cancellare tutte le righe di una tabella:

  `delete from Tabella;`

  oppure

  `truncate table Tabella;`

  - La tabella viene svuotata, ma la sua struttura rimane.
- Per cancellare sia i dati, sia la struttura:

  `drop table Tabella;`

Cancellazione di dati - 3

- Esempi (cosa fanno?)
  - `delete from P where City='London';`
  - ```
    delete from S where SNum in
           (select SP.SNum
             from SP join P on SP.PNum = P.PNum
             where P.City = 'London');
    ```
  - `delete from P;`
  - `truncate table SP;`
    ```

Cancellazione di dati - 4

- Esempio di cancellazione con sottointerrogazione: cancellare le forniture dei fornitori di Londra.

```
delete from SP
where 'London' =
        (select City
         from S
         where S.SNum = SP.SNum);
```

Aggiornamento di dati - 1

- Per modificare il valore di uno o più attributi delle righe di una tabella che soddisfano una determinata condizione si usa il comando **update**.
  - Sintassi

```
update Tabella
set Attributo_1 = EspressioneNuovoValore_1,
…
set Attributo_n = EspressioneNuovoValore_n,
where Condizione;
```

Aggiornamento di dati - 2

- *EspressioneNuovoValore* può essere:
  - Un valore costante o un'espressione semplice.
  - Un'espressione calcolata con una sottointerrogazione.
  - **`null`** o **`default`**.

Aggiornamento di dati - 3

- Per modificare il valore di uno o più attributi di tutte le righe di una tabella, è sufficiente omettere la clausola **`where`**:

```
update Tabella
set Attributo1 = EspressioneNuovoValore1,
…
set Attributon = EspressioneNuovoValoren;
```

Aggiornamento di dati - 4

- **Esempio 1:** aumentare del 30% lo status di tutti i fornitori di Parigi.

```
update S
set Status = Status * 1.3
where City = 'Paris';
```

- **Esempio 2:** impostare lo Status dei fornitori uguale al numero di forniture.

```
update S
set Status =
        ( select count(*) from SP
          where SP.SNum = S.SNum );
```

Aggiornamento di dati - 5

- Esempio di aggiornamento con sottointerrogazione: impostare a zero la quantità fornita per tutti i fornitori di Londra.

```
update SP
set QTY = 0
where 'London' =
          (select City
          from S
          where S.SNum = SP.SNum);
```

Riepilogo DML

- `select … from … where … group by … having … order by …;`
- `insert into … values … ;`
- `delete from … where …;`
- `update … set … where …;`

Aspetti evoluti del DDL
Vincoli di integrità

Vincoli di integrità generici

- Abbiamo già visto alcuni vincoli predefiniti di SQL per garantire l'integrità intrarelazionale e interrelazionale dei database (ad es. not null, vincoli di integrità referenziale).
- SQL permette di definire vincoli ulteriori, rispetto a quelli predefiniti, che possono riguardare le *business rules* (regole aziendali) e garantire l'integrità del database dal punto di vista dell'applicazione.

Vincoli generici con `check` - 1

- Uso della clausola **`check`** nella **`create table`**:

```
create table NomeTabella (
            Attributo1 … check (Condizione1),
            …,
            Attributon … check (Condizionej),
            [constraint NomeVincolo1] check (Condizionej+1),
            …,
            [constraint NomeVincolom] check (Condizionen+m)
);
```

- Si può specificare:
  - Dopo una dichiarazione di attributo (se fa riferimento unicamente a quell'attributo).
  - Alla fine della **`create table`** (con la possibilità di assegnare un nome al vincolo).

Vincoli generici con `check` - 2

- **Esempio:** imporre che l'attributo `QTY` in `SP` non sia mai negativo (`check` *insieme alla dichiarazione dell'attributo*).

```
create table SP (
  SNum varchar(3),
  PNum varchar(3),
  QTY decimal(5) not null check (QTY >= 0),
  constraint SP_PK primary key(SNum, PNum),
  constraint SP_FK_S foreign key(SNum)
         references S(SNum) on delete cascade,
  constraint SP_FK_P foreign key(PNum)
         references P(PNum) on delete cascade
);
```

Vincoli generici con `check` - 3

- **Esempio:** imporre che l'attributo `QTY` in `SP` non sia mai negativo (`check` *alla fine di* `create`).

```
create table SP (
  SNum varchar(3),
  PNum varchar(3),
  QTY decimal(5) not null,
  constraint SP_PK primary key(SNum, PNum),
  constraint SP_FK_S foreign key(SNum)
          references S(SNum) on delete cascade,
  constraint SP_FK_P foreign key(PNum)
          references P(PNum) on delete cascade,
  check (QTY >= 0)
);
```

Vincoli generici con `check` - 4

- **Esempio:** imporre che l'attributo `QTY` in `SP` non sia mai negativo (*con assegnamento di un nome*).

```
create table SP (
  SNum varchar(3),
  PNum varchar(3),
  QTY decimal(5) not null,
  constraint SP_PK primary key(SNum, PNum),
  constraint SP_FK_S foreign key(SNum)
          references S(SNum) on delete cascade,
  constraint SP_FK_P foreign key(PNum)
          references P(PNum) on delete cascade,
  constraint SP_CHK_QTY check (QTY >= 0)
);
```

Vincoli con sottointerrogazione - 1

- **Esempio:** imporre che la quantità totale di prodotti forniti dai fornitori di Londra non superi 1000.

```
create table SP (
  SNum varchar(3),
  PNum varchar(3),
  QTY decimal(5) not null check ( 1000 >=
          ( select sum(QTY)
            from SP join S on SP.SNum = S.SNum
            where S.City = 'London' ) ),
  constraint SP_PK primary key(SNum, PNum),
  constraint SP_FK_S foreign key(SNum)
          references S(SNum) on delete cascade,
  constraint SP_FK_P foreign key(PNum)
          references P(PNum) on delete cascade
);
```

Vincoli con sottointerrogazione - 2

- In PostgreSQL e in Oracle i vincoli con sottointerrogazione non sono supportati
- Per aggirare il problema si può ricorrere ai **`trigger`**, che sono porzioni di codice eseguite quando si verifica un certo evento (ad esempio una insert o una update)
- Non vedremo i trigger.

Aspetti evoluti del DDL
Viste

Viste - 1

- È possibile aggiungere allo schema del database rappresentazioni diverse dello stesso insieme di dati definendo tabelle derivate da tabelle di base.
- Le tabelle derivate si chiamano viste e possono essere differenziate in:
  - **Relazioni virtuali** o semplicemente **viste**: tabelle definite per mezzo di query SQL. Non sono tabelle effettivamente memorizzate nello schema del DB ma possono venire utilizzate come se lo fossero.
    - La query SQL viene eseguita ogni volta che si fa riferimento alla vista.
  - **Viste materializzate**: tabelle, derivate da espressioni SQL, effettivamente memorizzate come tabelle nel DB e tenute automaticamente sincronizzate con le tabelle di base (non le vedremo).

Viste - 2

- Per creare una vista si utilizza il costrutto **`create view`**.
  - Sintassi più semplice

```
create view NomeVista as
select … from … where …;
```

  - Crea la vista **`NomeVista`** a partire dalla **`select`**
  - La vista viene aggiunta allo schema del database e può essere usata come se fosse una tabella che ha come attributi quelli definiti dalla **`select`**.
- Per cancellare una vista:

```
drop view NomeVista;
```

Viste - 3

- Sintassi con ridenominazione degli attributi:

```
create view NomeVista(ListaAttributi1) as
select (ListaAttributi2) from … where …;
```

  - Crea la vista **`NomeVista`** a partire dalla **`select`**.
  - Gli attributi della vista prenderanno il nome da **`ListaAttributi1`**.
  - **`ListaAttributi1`** e **`ListaAttributi2`** devono essere coerenti.

Viste - 4

- **Esempio:** creare una vista che contiene solo le forniture di almeno 300 pezzi.

```
create view BigSP as
  select *
  from SP
  where QTY>=300;

select * from BigSP;
```

SNum	PNum	QTY
S1	P1	300
S1	P3	400
S2	P1	300
S2	P2	400
S4	P4	300
S4	P5	400

Viste - 5

- **Esempio:** creare una vista delle forniture dei fornitori di Londra (rinominando gli attributi).

```
create view SPLondonView(SNumL,SNameL,PNumL,PNameL,QTYL) as
  select SP.SNum,S.SName,SP.PNum,P.PName,SP.QTY
  from SP join S on SP.SNum = S.SNum join P on SP.PNum=P.PNum
  where S.City = 'London';
```

```
select * from SPLondonView;
```

SNumL	SNameL	PNumL	PNameL	QTYL
S1	Smith	P1	Nut	300
S1	Smith	P2	Bolt	200
S1	Smith	P3	Screw	400
S1	Smith	P4	Screw	200
S1	Smith	P5	Cam	100
S1	Smith	P6	Cog	100
S4	Clark	P2	Bolt	200
S4	Clark	P4	Screw	300
S4	Clark	P5	Cam	400

Modifica di dati nelle viste - 1

- In alcuni casi è possibile modificare (inserire, aggiornare, cancellare) i dati contenuti in una vista; le modifiche si ripercuotono sulle tabelle di base.
- Affinché una vista sia modificabile è necessario che a ogni riga della vista corrisponda una, e una sola, riga di una sola tabella di base.
- **Esempio:** aggiornamento di una riga
  - ```
    update BigSP set QTY=350
    where SNum='S1' and PNum='P1';
    ```
  - ```
    update SPLondonView set QTYL=380
    where SNumL='S1' and PNumL='P1';
    ```

aggiornano le righe corrispondenti della tabella SP.

Modifica di dati nelle viste - 2

- Quali righe contiene, invece, la vista *BigSP* dopo l'esecuzione del seguente aggiornamento?

```
update BigSP
set QTY=100
where SNum='S1' and PNum='P1';
```

Modifica di dati nelle viste - 3

- Per fare in modo che modifiche a righe di una vista non interferiscano con le condizioni di definizione della vista stessa si usa:

```
create view … as
          select … from … where …
with check option;
```

Modifica di dati nelle viste - 4

- **Esempio:** creare una vista delle forniture di almeno 300 pezzi <u>con controllo delle modifiche.</u>

```
create view BigSP as
   select *
   from SP
   where QTY>=300
with check option;
```

- Ora la modifica in **`BigSP`** della riga `<'S1', 'P1', 300>` in `<'S1', 'P1', 100>` **non è più ammessa** perché la riga risultante non farebbe più parte della vista.

Modifica di dati nelle viste - 5

- È possibile creare viste a partire da altre viste.
- La clausola **`check option`**, in questo caso, può funzionare in due modi:
  - **`local`**: vengono annullate solo le modifiche che vìolano le condizioni della vista che si sta modificando:

    ```
    … with local check option;
    ```
  - **`cascaded`**: vengono annullate anche le modifiche che vìolano le condizioni delle viste da cui la vista è originata:

    ```
    … with cascaded check option;
    ```

Common table expressions

Clausola `with` - 1

- Definisce Common Table Expressions (CTE).
- Permette di definire una sorta di **tabella temporanea** che esiste per una sola query.

Clausola `with` - 2

- Ad esempio, selezioniamo i fornitori che forniscono almeno il 10% delle forniture totali:

```
WITH total_suppliers AS (
        SELECT SNum, SUM(qty) AS
total_supplied_parts
        FROM SP
        GROUP BY SNum )
SELECT SNum
FROM total_suppliers
WHERE total_supplied_parts > (
       SELECT SUM(total_supplied_parts)/10
       FROM total_suppliers );
```

Clausola `with` - 3

- La stessa interrogazione si può scrivere facendo uso di sottoquery nella clausola `from`:

```
SELECT SNum
FROM (
   SELECT SNum, SUM(qty) AS total_supplied_parts
   FROM SP
   GROUP BY Snum
) as total_suppliers
WHERE total_supplied_parts > (
   SELECT SUM(total_supplied_parts)/10
   FROM (
      SELECT SNum, SUM(qty) AS total_supplied_parts
      FROM SP
      GROUP BY Snum
   ) as total_suppliers
);
```

Clausola `with` - 4

- **Esempio:** elencare il nome del fornitore e la quantità massima da lui fornita (con clausola `with`):

```
with SMax as (
   select SNum, max(Qty) as MaxQty
   from SP
   group by Snum
)
select SName, MaxQty
from S left join SMax on S.SNum= SMax.SNum;
```

Clausola `with` - 5

- **Esempio:** elencare il nome del fornitore e la quantità massima da lui fornita (con sottoquery nella clausola `from`):

```
select SName, MaxQty
from S left join (
   select SNum, max(Qty) as MaxQty
   from SP
   group by SNum) SMax
    on S.SNum=SMax.SNum;
```

Clausola `with` - 6

- La parola chiave **`RECURSIVE`** permette di esprimere query non esprimibili altrimenti in SQL.
- Esempio, elenchiamo gli interi da 1 a 100:

```
WITH RECURSIVE t(n) AS (
    SELECT 1
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100 )
SELECT n FROM t;
```

- «**`SELECT 1`**» è il caso base, separato con «**`UNION ALL`**» dal termine ricorsivo.

Clausola `with` - 7

- **`WITH RECURSIVE`** è utile per dati che rappresentano strutture ad albero.
- Esempio, da tabella **`Parts(subpart, part, qty)`** ricaviamo tutte le sottoparti (dirette o indirette) di "Cog":

```
WITH RECURSIVE includedparts(subpart, part, qty) AS (
    SELECT subpart, part, qty FROM parts WHERE part='Cog'
  UNION ALL
    SELECT p.subpart, p.part, p.qty FROM includedparts pr, Parts p
    WHERE p.part = pr.subpart );

SELECT subpart, SUM(qty) as totalqty
FROM includedparts
GROUP BY subpart;
```

Creazione di tabelle da query

- Per creare una tabella a partire da una query su tabelle esistenti si usa

```
create table NomeTabella as
          select … from … where … ;
```

- Viene creata la tabella *NomeTabella* con le stesse colonne e righe che risultano dalla `select`
- È utile per copiare i dati delle tabelle ma non copia lo schema (vincoli, chiavi primarie, …); per copiare tutta la struttura, bisogna copiare l'SQL di creazione della tabella

Creazione di tabelle da query

- **Esempio 1:** creare una copia di S.

```
create table S_Copia as
        select * from S;
```

- **Esempio 2:** creare la tabella dei fornitori di Londra.

```
create table S_Londra as
        select * from S
        where City = 'London';
```

Gestione delle transazioni

- Per iniziare una transazione si usa il comando **`start transaction`** (o, in PostgreSQL **`begin`**;), per terminarla con successo il comando **`commit`**; e per farla abortire il comando **`rollback`**.
  - I comandi tra **`begin`** e **`commit`** (**`rollback`**) vengono eseguiti come una singola transazione.
  - PostgreSQL, se non trova un comando di inizio transazione, esegue un commit implicito dopo ogni istruzione SQL.