

Basi di Dati
prof. Letizia Tanca

lucidi tratti dal libro
Atzeni-Ceri-Paraboschi-Torlone

AA 2006-07

SQL avanzato e altri aspetti del
DDL

Comandi di modifica e viste

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

- 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*)

Inserimento

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

```
insert into
Studiante(Mat, Nome, Città, CDip)
values ('456878', 'Giorgio Rossi',
       'Bologna', 'Logistica')
```
- Usando una query con *ListaAttributi*:

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

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 = 1-4-03
```

```
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 query nidificate nelle modifiche

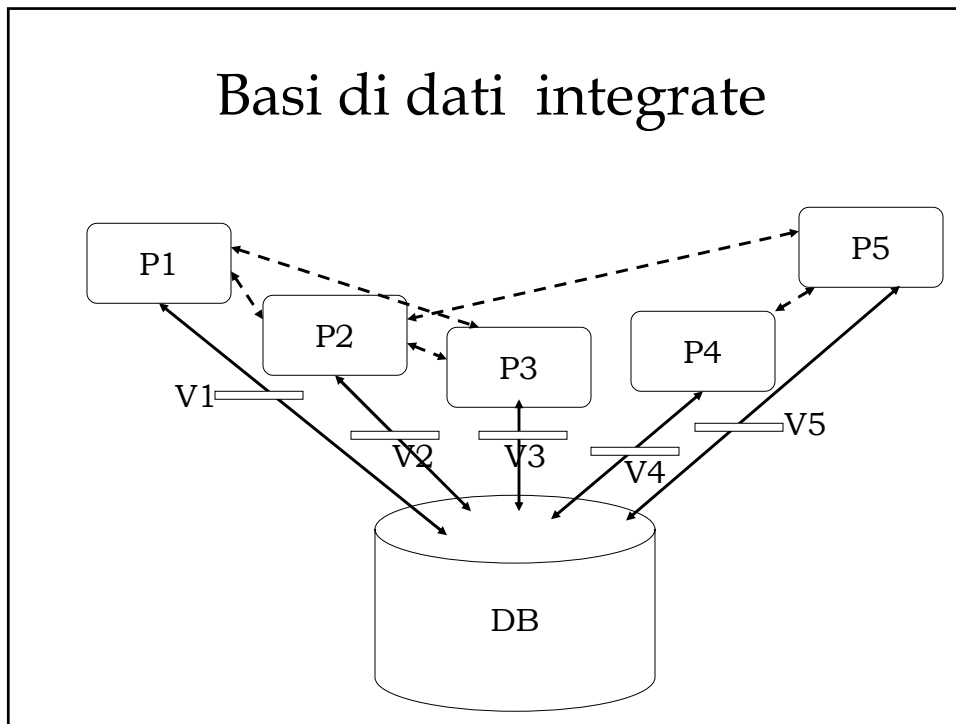
Assegnare a TotPezzi la somma delle quantità delle linee di un ordine

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

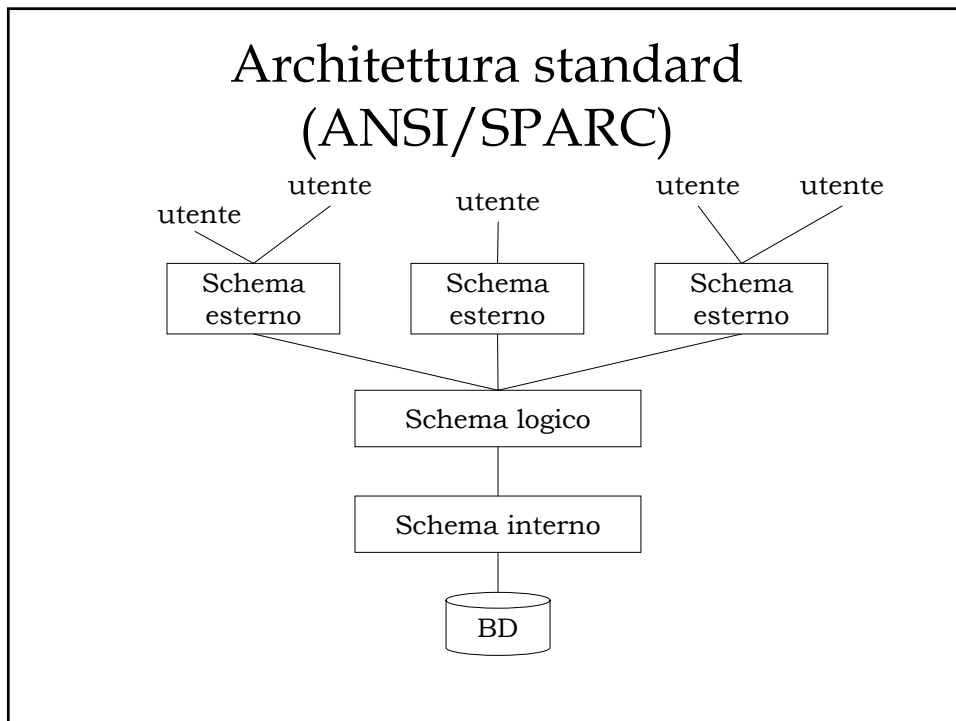
Viste

- Servono a offrire la "visione" di tabelle virtuali (schemi esterni)
- Si definiscono assegnando un nome a un'interrogazione
- Molto chiaro se si pensa ai predicati intensionali di Datalog

Basi di dati integrate



Architettura standard (ANSI/SPARC)



Architettura ANSI/SPARC: schemi

schema logico: descrizione dell'intera base di dati nel modello logico "principale" del DBMS

schema esterno: descrizione di parte della base di dati in un modello logico ("viste" parziali, derivate, anche in modelli diversi)

schema fisico: rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione.

Una vista

Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

Aule

Nome	Edificio	Piano
DS1	Ex-OMI	Terra
N3	Ex-OMI	Terra
G	Pincherle	Primo

CorsiSedi

Corso	Aula	Edificio	Piano
Sistemi	N3	Ex-OMI	Terra
Reti	N3	Ex-OMI	Terra
Controlli	G	Pincherle	Primo

Viste

Sintassi SQL:

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

Modifiche tramite le viste

- Vista:

```
create view OrdiniPrincipali as  
  select *  
  from Ordine  
  where Importo > 10000
```

- 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 > 10000
```

Check option

- La **check option** interviene quando viene aggiornato il contenuto di una vista, per verificare che la tupla inserita/modificata appartenga alla vista (cioè sia coerente con la sua definizione)
- 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 = 5000
  where CodOrd = '754'
```

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

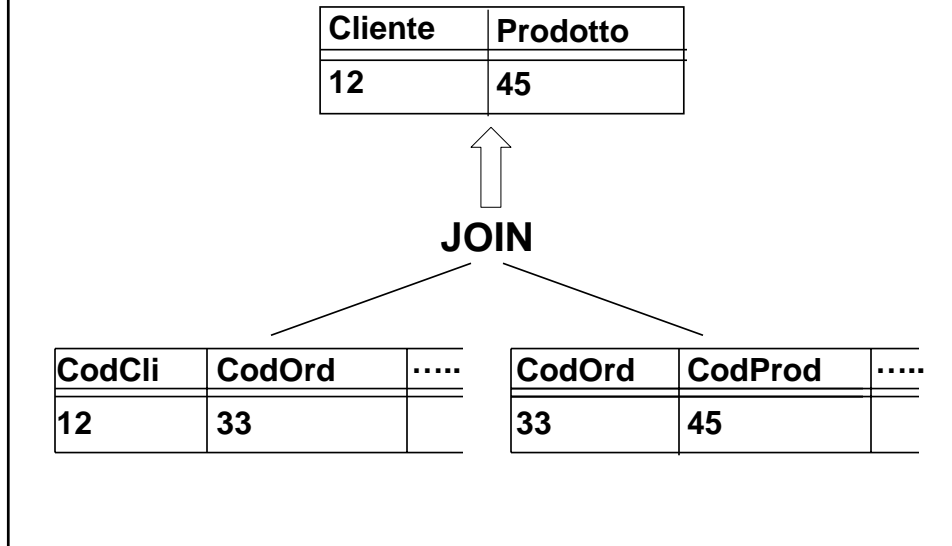
Viste

Le viste in SQL-2 possono contenere nella definizione altre viste precedentemente definite, ma non vi può essere mutua dipendenza (la ricorsione è stata introdotta in SQL:1999)

Interrogazioni su viste

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

Vista complessa (JOIN)



Composizione della vista con la query

- Query:

```
select Cliente
from CliProd
where Prodotto = '45'
```
- Composizione della vista con la query:

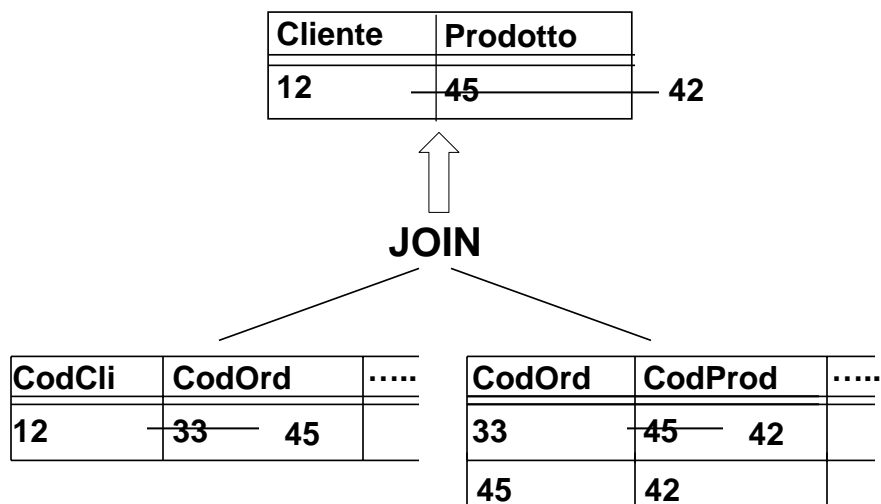
```
select CodCli
from Ordine join Dettaglio
on Ordine.CodOrd = Dettaglio.CodOrd
where CodProd = '45'
```

Modifiche su una vista complessa

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

```
update CliProd
  set Prodotto = '42'
  where Cliente = '12'
```
- Due interpretazioni alternative per la realizzazione sulle tabelle di base
 - il cliente ha cambiato l'ordine
 - il codice del prodotto è cambiato

Vista complessa (JOIN)



Aspetti avanzati del DDL

Aspetti evoluti del DDL

- Creazione di indici
- Autorizzazioni d'accesso
- Vincoli di integrità
- Procedure e regole attive

Creazione di indici

- Indici: meccanismi di accesso efficiente ai dati

```
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)

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., supponiamo di aggiungere la definizione di un attributo *Superiore* nello schema della tabella IMPIEGATO:

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

Esempio: gestione magazzino

Magazzino

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

Riordino

CodProd	Data	QtaOrd

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 interrelazionali 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))
```

Asserzioni: esempio

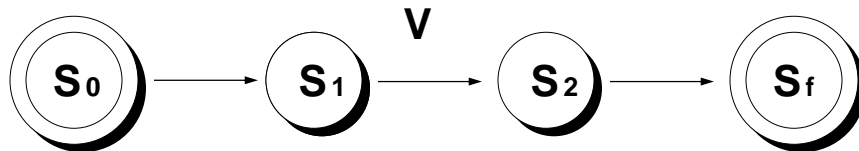
- Dato il seguente schema di Base di Dati:
ASTRONAUTA(MatricolaAst, Nome, Cognome, Base Brevetto)
MISSIONE(NumeroId, AnnoInizio, AnnoFine, Base Partenza, BaseArrivo)
PARTECIPA(MatricolaAst, NumeroMissione, Ruolo)
- Definire il vincolo che indica che ogni astronauta può conseguire il brevetto in una base spaziale che è base di partenza o di arrivo di qualche missione:

```
Create table astronauta( ....  
check (Base Brevetto IN (select BasePartenza  
from Missione  
union  
Select BaseArrivo  
from Missione)))
```

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



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
```

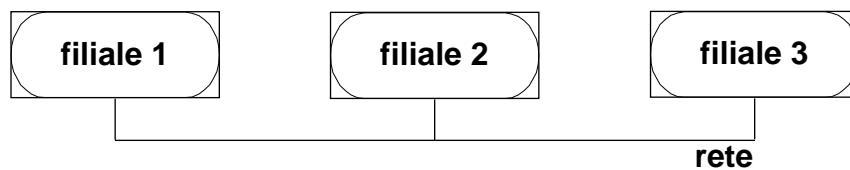
```
revoke update on Ordine from User1  
revoke select on Ordine from User3
```

Viste e autorizzazioni di accesso

Viste = unità di autorizzazione

- Consentono la gestione ottimale della privacy

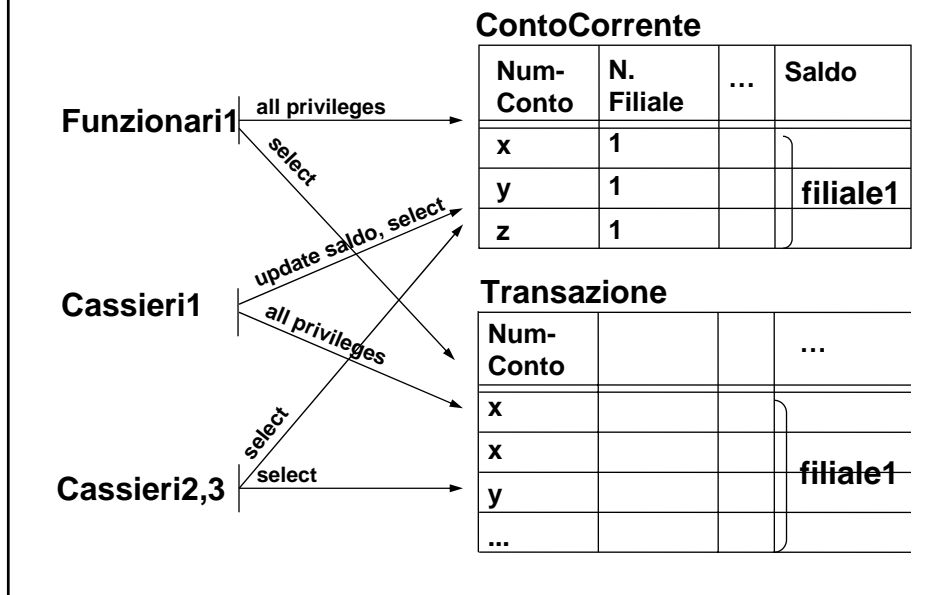
Esempio: gestione dei conti correnti



**ContoCorrente(NumConto, Filiale,
Cliente, CodFisc, DataApertura, Saldo)**

**Transazione(NumConto, Data, Progr,
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
```

Esercizi su vincoli da temi esame

Si consideri il seguente schema di base di dati di una casa editrice:

bozza (id., n°_capitolo, ISBN_libro,
c.f._correttore, data_invio, data_restit.)

capitolo(n°_capitolo, titolo, ISBN libro,
n°_pagine)

libro(ISBN, titolo, autore, n°_pagine,
data_prevista)

correttore(c.f., nome, n°_telefono)

Specificare in SQL il vincolo che valida che la data di restituzione di tutte le bozze del capitolo di un certo libro preceda la data prevista del libro.

```
CREATE ASSERTION ControlloDate  
CHECK (NOT EXIST (SELECT *  
FROM bozza,libro  
WHERE bozza.ISBN_libro=libro.ISBN  
AND  
data_restit>data_prevista))
```

Specificare in SQL il vincolo che valida che la data di restituzione di tutte le bozze del capitolo di un certo libro preceda la data prevista del libro.

```
CREATE ASSERTION ControlloDate  
CHECK (NOT EXIST (SELECT *  
FROM bozza,libro  
WHERE bozza.ISBN_libro=libro.ISBN  
AND  
data_restit>data_prevista))
```

```

CREATE ASSERTION ControlloDate
CHECK (0 = (SELECT COUNT (*)
FROM bozza,libro
WHERE bozza.ISBN_libro=libro.ISBN
AND
data_restit>data_prevista))

```

Nella definizione della tabella BOZZA

```

create table BOZZA
( id char(6) primary key,
  n_capitolo smallint,
  ISBN_libro char(10),
  c.f._correttore char(16),
  data_invio date not null,
  data_restit. date
  check data_restit.< SELECT data_prevista
FROM LIBRO
WHERE
ISBN_libro=LIBRO.ISBN,
  foreign key (ISBN_libro,n_capitolo)
  references CAPITOLO
  on delete cascade
  on update cascade)

```

Esercizi su vincoli da temi esame

Si consideri il seguente schema di base di dati che vuole tenere traccia delle rappresentazioni di un gruppo di compagnie teatrali:

COMPAGNIA (nome, cfdirettore, città_sede)

TEATRO (id_teatro, nome, città, numero di telefono)

CALENDARIO (id_teatro, giorno, titolo, compagnia,
ora_inizio, prezzo_biglietto)

SPETTACOLO (titolo, compagnia, genere, durata)

PERSONA (CF, Nome, Cognome, datadinascita,
cittànascita, telefono)

*La somma della durata delle
rappresentazioni previste per un teatro in un
giorno non superi le 6 ore.*

```
CREATE ASSERTION ControlloDurata  
CHECK ( 6 > ALL ( SELECT Durata  
FROM DurataRappresentazioni))
```

```
CREATE VIEW DurataRappresentazioni(Teatro, Giorno,  
Durata) AS  
SELECT id_teatro, giorno, SUM (Durata)  
FROM Calendario, Spettacolo  
WHERE Calendario.titolo=spettacolo.titolo AND  
Calendario.compagnia=spettacolo.compagnia  
GROUP BY id_teatro, giorno)
```

Esercizi su vincoli da temi esame

Si consideri il seguente schema di base di dati che vuole tenere traccia dei DVD noleggiati dai clienti di una videoteca:

CLIENTE (CodiceFiscale, Cognome, Nome, Residenza)

DVD (Matricola, TitoloFilm, Anno, Durata)

FILM (Titolo, Anno, CognomeRegista, NomeRegista, Genere)

NOLEGGIO (CodiceCliente, CodiceDvd, DataInizio, DataFine, Prezzo)

Per i prestiti di durata maggiore di 2 giorni il prezzo deve essere superiore a 6 euro

```
CREATE ASSERTION ControlloPrezzo
CHECK ( NOT EXISTS (SELECT *
                    FROM Noleggio
                    WHERE Prezzo < 6 AND
                    (DataFine-DataInizio) > 2))
```

oppure

```
CREATE ASSERTION ControlloPrezzo
CHECK ( 6 < ALL (SELECT Prezzo
                FROM Noleggio
                WHERE (DataFine-DataInizio) > 2))
```

Tabelle

- STUDENTE(Matricola, Nome, Cognome, Indirizzo, Città, CAP,M/F)
- INSEGNANTE(Matricola, Nome, Cognome, Città,Telefono, Stipendio)
- CORSO(Codice, Nome, Facoltà, NumeroCrediti)
- ESAME(CodiceCorso, MatricolaStudente, Voto)
- INSEGNAMENTO(CodiceCorso, MatricolaProfessore, NumeroStudenti)

Interrogazione 40

- Quali studenti hanno il voto medio maggiore del voto medio più basso? Elencare nome e voto medio degli studenti

```
CREATE VIEW VotiMedi(Matricola,VotoMedio) AS  
SELECT MatricolaStudente, AVG(Voto)  
FROM Esame  
GROUP BY MatricolaStudente
```

```
SELECT Matricola, Nome, Cognome, VotoMedio  
FROM Studente JOIN VotiMedi ON  
    Studente.Matricola=VotiMedi.Matricola  
WHERE VotoMedio > SELECT MIN(VotoMedio)  
                FROM VotiMedi
```

Interrogazione 45

- Quali studenti hanno qualche voto maggiore o uguale al voto medio più alto in assoluto? Elencare nome degli studenti e relativo voto

```
CREATE VIEW VotiMedi(Matricola,VotoMedio) AS  
SELECT MatricolaStudente, AVG(Voto)  
FROM Esame  
GROUP BY MatricolaStudente
```

```
SELECT Matricola, Nome, Cognome, Voto  
FROM Studente JOIN Esame ON  
    Studente.Matricola=Esame.MatricolaStudente  
WHERE Voto >= SELECT MAX(VotoMedio)  
                FROM VotiMedi
```

Esercizi su viste da temi esame

Si consideri il seguente schema di base di dati di una casa editrice:

bozza (id., n°_capitolo, ISBN_libro,
c.f._correttore, data_invio, data_restit.)

capitolo(n°_capitolo, titolo, ISBN libro,
n°_pagine)

libro(ISBN, titolo, autore, n°_pagine,
data_prevista)

correttore(c.f., nome, n°_telefono)

- *Trovare quale correttore ha corretto il maggior numero di pagine.*

```

CREATE VIEW NumPagine(CFCorrettore,N_Pagine) AS
SELECT c.f._correttore, SUM(n_pagine)
FROM bozza,capitolo
WHERE bozza.n_capitolo=capitolo.n_capitolo AND
      bozza.ISBN_libro=capitolo.ISBN_libro
GROUP BY c.f._correttore

SELECT CFCorrettore, Nome
FROM NumPagine JOIN CORRETTORE ON
      NumPagine.CFCorrettore=CORRETTORE.c.f
WHERE N_Pagine>= SELECT MAX(N_Pagine)
                  FROM NumPagine

```

Esercizi su viste da temi esame

Si consideri il seguente schema di base di dati che vuole tenere traccia dei DVD noleggiati dai clienti di una videoteca:

CLIENTE (CodiceFiscale, Cognome, Nome, Residenza)

DVD (Matricola, TitoloFilm, Anno, Durata)

FILM (Titolo, Anno, CognomeRegista, NomeRegista, Genere)

NOLEGGIO (CodiceCliente, CodiceDvd, DataInizio, DataFine, Prezzo)

- *Trovare i clienti che hanno effettuato il minor numero di noleggi del genere "Horror" nell'anno 2005.*

```
CREATE VIEW Noleggi (CFCliente, N_Noleggi) AS
SELECT CodiceCliente, COUNT(*)
FROM Noleggio, DVD, Film
WHERE Noleggio.CodiceDVD=DVD.Matricola AND
      Film.Titolo=DVD.Titolo AND Film.Anno=DVD.Anno AND
      DataInizio>=01/01/2005 AND DataInizio<=31/12/2005
      AND Genere="Horror"
GROUP BY CodiceCliente

SELECT Nome, Cognome
FROM Cliente JOIN Noleggi ON
      Cliente.CodiceFiscale=Noleggi.CFCliente
WHERE N_noleggi = (SELECT MIN(N_Noleggi)
                  FROM Noleggi)
```