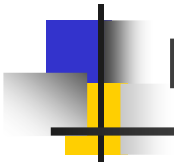
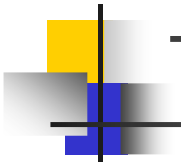


Basi di dati attive

Basi di dati attive

Una base di dati attiva

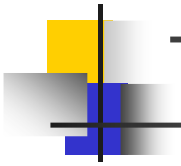
- dispone di un sottosistema integrato per definire e gestire regole di produzione
- reagisce autonomamente a eventi eseguendo azioni
- dispongono di un processore di regole (rule engine)
- integra nello schema parte della business logic
  - altrimenti, tale conoscenza dovrebbe essere implementata da programmi applicativi
  - il vantaggio è l'indipendenza della conoscenza (che si aggiunge all'indipendenza logica e fisica)



Trigger

I trigger

- definiscono regole in SQL
- Adottano lo schema ECA (Event Condition Action)
  - **ON *evento* IF *condizione* THEN *azione***
    1. se si verifica l'evento, si valutata la cndizione
    2. se la condizione è soddisfatta l'azione viene eseguita
- gli eventi corrispondono alle primitive DML di SQL: insert, update, delete
- la condizione è un'espressione SQL
- l'azione è una sequenza di primitive SQL eventualmente arricchite da un linguaggio di programmazione



Trigger in SQL-99

- I trigger sono stati definiti formalmente da SQL-99 (SQL3)
- I DBMS commerciali usano versioni proprietarie dei trigger nate prima di SQL-99

Ogni trigger è caratterizzato da

- nome
- nome della tabella che viene monitorata
- modo di esecuzione (BEFORE o AFTER)
- l'evento monitorato (INSERT, DELETE o UPDATE)
- granularità
- nomi e alias per transition values e transition tables
- l'azione
- il timestamp di creazione

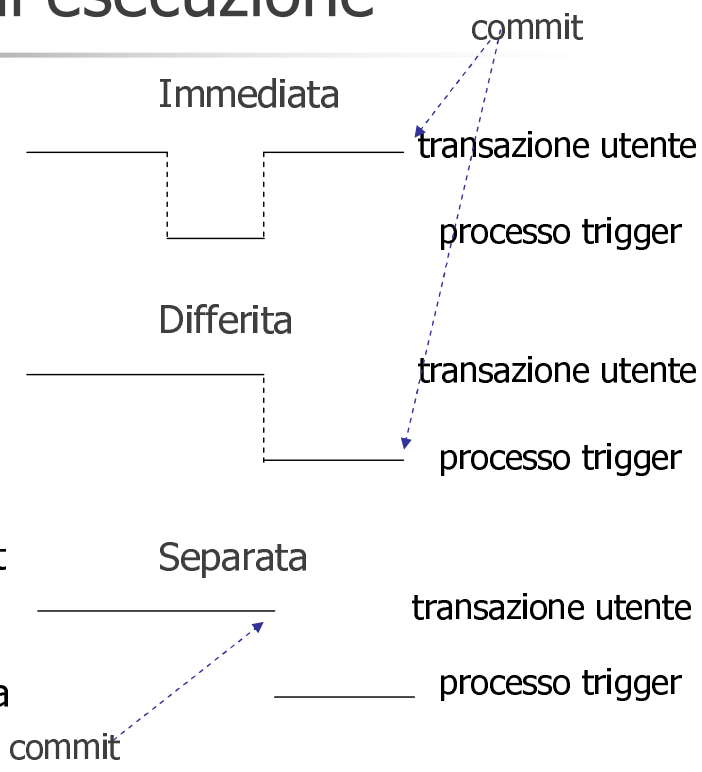
Sintassi trigger SQL-99

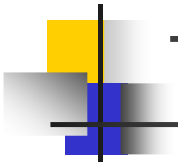
create trigger *NomeTrigger*
{**before** | **after**}
{ **insert** | **delete** | **update** [**of** *Colonne*] } **on** *Tabella*
[**referencing** {[**old table** [**as**] *AliasTabellaOld*]
[**new table** [**as**] *AliasTabellaNew*] } |
{[**old** [**row**] [**as**] *NomeTuplaOld*]
[**new** [**row**] [**as**] *NomeTuplaNew*] }]
[**for each** { **row** | **statement** }]
[**when** *Condizione*]
Comandi SQL

Trigger: modo di esecuzione

Modo di esecuzione

- Un trigger può essere eseguito in modalità diverse che dipendono dall'implementazione (dal DBMS)
  - immediata, cioè prima del commit dell'evento (usata da SQL-99)
  - differita, al momento del commit
  - separata, dopo il commit
- Nell'esecuzione immediata e differita un rollback nel trigger causa un rollback dell'evento

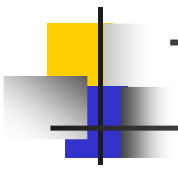




Trigger: modo di esecuzione

Un trigger eseguito in modalità immediata può essere eseguito

- prima dell'evento (opzione before)
- dopo l'evento (opzione after)
- alcune implementazioni permettono anche "instead of" che indica che il trigger deve essere eseguito invece del comando:
  - tali implementazioni sono poche perché la semantica che ne risulta può essere poco intutiva



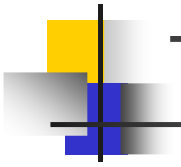
Trigger: granularità e condizione

Granularità (row, statement)

- definisce quante volte viene attivato
  - row-level: attivazione per ogni ennupla coinvolta nell'operazione
  - statement-level: una sola attivazione per ogni istruzione SQL, con riferimento a tutte le ennuple coinvolte ("set-oriented")

Condizione

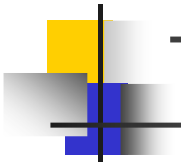
- la condizione è una espressione logica che restringe il campo di applicazione della regola



Trigger: azione

L'azione

- contiene un singolo comando o una sequenza di comandi
- in SQL-99, ORACLE e MSSQL SERVER mettono a disposizione un linguaggio proprio per scrivere la sequenza di comandi
- Nei trigger before, l'azione può contenere definizione di dati, selezioni di dati … ma non può modificare lo stato della base di dati
- Nei trigger after, l'azione può contenere anche operazioni di manipolazione dei dati (INSERT, DELETE, UPDATE)

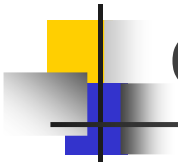


Trigger: alias

Alias delle tabelle o tuple che contengono i valori prima e dopo la modifica

- il trigger può accedere a tabelle (tuple) che contengono copie dei dati prima e dopo la modifica
- in SQL-99 è possibile dare un nuovo nome alle tabelle (tuple) che contengono i nuovi e i vecchi dati
- le tabelle (tuple) accessibili dipendono dal tipo di trigger

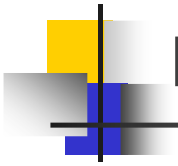
	OLD ROW	NEW ROW	OLD TABLE	NEW TABLE
before statement	-	-	-	-
before row	delete, update	insert, update	-	-
after statement	-	-	delete, update	insert, update
after row	delete, update	insert, update	delete, update	insert, update



Osservazione

In intelligenza artificiale si usano spesso regole con condizioni ma senza eventi, perché nei trigger si usano anche gli eventi?

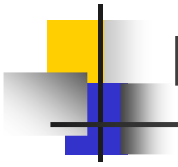
- l'evento è valutabile facilmente e a basso costo
- questo vantaggio è importante nella basi di dati dove spesso si opera su una grande quantità di dati
- inoltre, posso specificare azioni diverse per eventi diversi e stessa condizione



Esempio: before

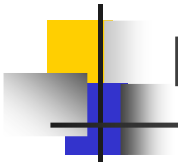
```
create trigger LimitaAumenti
    before update of salario  on Impiegati
    for each  row
    when New.salario > 1.2*Old.salario
    set New.salario=1-2*Old.salario
```

tabelle che di default contengono i dati prima e dopo le modifche



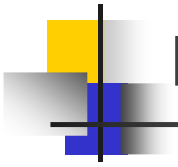
Esempio: after

```
create trigger LimitaAumenti
    after update of salario  on Impiegati
    for each  row
    when New.salario > 1.2*Old.salario
          update Impiegati
          set Impiegati.salario=1.2 * Impiegati.salario
          where Impiegati.id=New.id
```



Esempio: granularità statement

```
create trigger LimitaAumenti
    after update of salario  on Impiegati
    for each  statement
    insert into  LogSalari
          (Current_date,
          select AVG(salario) from Impiegati)
```



Processo di esecuzione dei trigger

I passi del processo di esecuzione dei trigger sono

1. Il nucleo del DBMS individua un evento
2. Viene istanziato il corpo delle regole attivate
3. Si sceglie un ordine di esecuzione delle regole
4. Si verifica la condizione delle regole
5. Si eseguono le regole:
   - se durante l'esecuzione si verificano nuovi eventi si va a 2

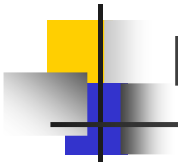
Ordine di esecuzione dei trigger e vincoli in SQL-99

Ordine di esecuzione trigger di tipo diverso e vincoli

1. trigger BEFORE STATEMENT
2. per ogni tupla oggetto del comando
   3. trigger BEFORE ROW
   4. comando e verifica dei vincoli di integrità per righe
   5. trigger AFTER ROW
6. verifica dei vincoli che richiedono di aver completato il comando
7. trigger AFTER STATEMENT

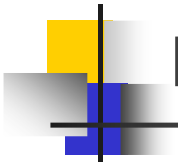
Ordine di esecuzione trigger dello stesso tipo

- se esiste più di un trigger dello stesso tipo, si usa l'ordine di creazione



Proprietà di insieme di trigger

- L'attivazione di un trigger può determinare in cascata l'attivazione di altri trigger
- L'attivazione di un trigger può causare un rollback che, in cascata e a seconda dell'implementazione del DBMS, può determinare il rollback delle modifiche dell'evento e di altre regole
- Mentre il comportamento di ogni singola regola è facile da comprendere, il comportamento di un insieme di regole può essere più difficile da capire



Proprietà di insiemi di trigger

- Le proprietà più interessanti sono la terminazione, la confluenza e l'osservabilità deterministica

Terminazione

- insieme di regole termina se nessun evento può determinare un'attivazione ciclica delle regole

Confluenza

- un insieme di regole è confluente se per ogni evento l'insieme di regole scatenate porta allo stesso stato finale indipendentemente dall'ordine di attivazione

Determinismo delle osservazioni

- insieme di regole garantisce il determinismo delle osservazioni se per ogni evento l'esecuzione delle regole è confluente e le azioni visibili delle regole sono sempre le stesse e sono svolte nello stesso ordine

Proprietà di insiemi di trigger

In teoria

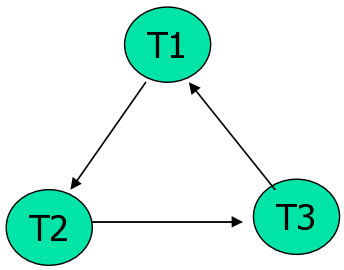
- La terminazione è essenziale, mentre confluenza e determinismo delle osservazioni sono desiderabili, ma non necessarie
- Nessuna delle proprietà può essere verificata automaticamente

In pratica, i DBMS

- definiscono dei limiti per il numero di trigger o la profondità della ricorsione che può essere attivata da un evento
- definiscono un ordine con cui i trigger relativi ad un evento sono attivati: ad esempio, tempo di creazione, nome

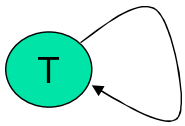
Osservazione

- Per visualizzare l'interazione fra trigger si può definire il grafo di attivazione dove
  - ogni nodo corrisponde ad una regola
  - gli archi indicano le dipendenze
- Si osservi che la presenza di cicli nel grado non significa necessariamente che le regole non terminano



Esempio di regola con grafo ciclico

```
create trigger LimitaAumenti
    after update of salario  on Impiegati
    for each  row
    update Impiegati
    set Impiegati.salario=0.9 * Impiegati.salari
    where (select AVG(salario) from Impiegati)>100
```



Termina

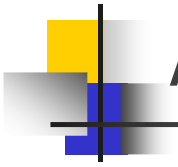
Non termina

```
create trigger LimitaAumenti
    after update of salario  on Impiegati
    for each  row
    update Impiegati
    set Impiegati.salario=1.1 * Impiegati.salari
    where (select AVG(salario) from Impiegati)>100
```

Estensioni

I trigger possono avere diverse (rare) estensioni

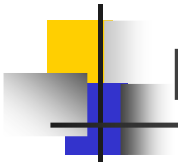
- eventi temporali (anche periodici) o "definiti dall'utente"
- combinazioni booleane di eventi
- definizione di priorità
- regole a gruppi, attivabili e disattivabili
- regole associate anche a interrogazioni
- regole associate a comandi di creazione e modifica dello schema, assegnazione di diritti, creazione e modifica di procedure, ...



Applicazioni dei trigger

I trigger possono essere usati per

- regole esterne che implementano parti di applicazioni
- regole interne ai DBMS
  - replicazione, integrità referenziale, alerting, sicurezza, statistiche, viste materializzate, …
  - si osservi che tali regole possono essere a sua volta suddivise in
    - regole del nucleo del DBMS
    - regole che espandono le funzionalità del DBMS
  - le regole interne sono tipicamente generate automaticamenre e possono essere o non essere visibili all'utente



Trigger in Oracle, DB2, MSS SQL e postgress

- PostgreSQL implementa un sottoinsieme dei trigger SQL-99 e la sintassi si attiene ampiamente allo standard
- DB2 differisce in pochi punti rispetto allo standard: fra coloro che hanno definito SQL-99 c'erano diversi progettisti IBM
- La sintassi di dei trigger in Oracle molto simile a SQL-99 con alcune differenze e diverse funzioni in più
- In MSSQL server la sintassi è completamente diversa

Sintassi trigger MSSQL server

create trigger *trigger_name* **on** { *table* | *view* }
{ for | after | instead of }
{ [**insert**] [,] [**update**] [,] [**delete**] }
as
[**if update (** *column* **)**
[{ **and** | **or** } **update (** *column* **)**]]
sql_statement

Trigger in MSSQL server

Particolarità di MSSQL server

- MSSQL server adotta l'esecuzione immediata e permette di scegliere fra la modalità **After** e quella "**instead of**" (before non è disponibile)
- La condizione **when** è sostituita da un controllo sulla modifica di colonne
- La clausola **reference** non esiste: le tabelle **inserted** e **deleted** sono disponibili di default
- Non è possibile scegliere la granularità: i trigger vengono eseguiti con granularità statement
- L'azione può essere scritta con il linguaggio TRANSACT-SQL

Esercizio 1

- Creare una tabella studenti con attributi: nome e id
- Creare una tabella esami con attributi; studente, voto, lode, corso e id
- Creare un trigger che implementa il vincolo per cui non ammesso un voto con la lode diverso dal 30 e il voto deve essere compreso fra 1 e 30

Transact-SQL

Variabili

- le variabili sono iniziano con il simbolo @
- devono essere dichiarate con il comando

  DECLARE @nome_variabile tipo

  il tipo può essere uno qualsiasi di quelli usabili in SQL eccetto **text**, **ntext** o **image**
- possono essere assegnate e aggiornate usando il comando SET

```
declare @contatore int
set @contatore=1
set @contatore=@contatore +1
```

Transact-SQL

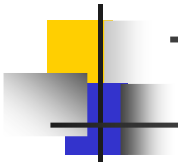
Controllo di flusso

- per controllare il flusso sono disponibili le istruzioni IF, FOR e WHILE
  - **if** (condition) **then** [statements]
  - [**else** [elsestatements]]
  - **while** (expression)
  - { statements | **break** | **continue** }

Blocchi

- insieme di istruzioni sono racchiusi fra **begin** e **end**

```
declare @b int
set @b=10
while (@b>0)
        begin
        print @b
        set @b=@b-1
        end
```



Transact-SQL

Errori e rollback

- l'istruzione raiserror permette di segnalare al sistema che siamo in presenza di un errore

  raiserror('messaggio',livello,stato)

- il livello dell'errore è una valore fra 1 e 25. Se l'errore è maggiore di 10, il batch corrispondente viene fermato e si fa il rollback della transazione corrispondente
- lo stato è un valore fra 1 e 128 e serve definire in che stato del codice in esecuzione si è verificato l'errore

Soluzione esercizio 1: vincolo intrarelazionale

```
create trigger trigger1 on  esami
after update, insert
as
begin
    if (exists (select * from inserted
          where voto<>30 and lode='true' ))
                begin
                raiserror('Vincolo sulla lode non rispettato',11,1)
                end
end
```

Esercizio 2

- Implementare il seguente vincolo nella tabella esami

  studente **int references** studenti(id) **on delete cascade**

- Si ricordi che il vincolo impone dei vincoli sia sui valori della tabella studenti che in quella esami

Soluzione esercizio 2: vincolo interrelazionale

```
create trigger trigger2 on  studenti
after delete
begin
    delete from esami
        where esami.studente in (select id from deleted)
end
```

Soluzione esercizio 2

```
create trigger trigger3 on  esami
after update, insert
as
begin
    if (exists (select * from inserted
        where inserted.studente not in (select id from studenti)))
                begin
                raiserror('Vincolo referenziale non rispettato',11,1)
                end
end
```

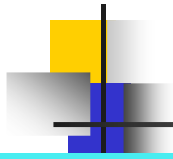
Esercizio 3: creare un log

- definire dei trigger che si attivano in corrispondenza delle modifiche della tabella studenti e registrano in un'altra tabella: timestamp, tipo di operazione, id,before image e after image del campo nome
- si assuma che l'id non sia modificato

time	operazione	id	before	after
1-1-08…	insert	2		puto
1-1-08..	update	3	puto	pluto
…	…		..	…

Soluzione esercizio 3

```
create trigger trigger4 on  studenti
after delete
as
begin
    insert into log(time,azione,id,before,after)
        select current_timestamp,'delete',deleted.id,deleted.nome,null
        from deleted
end
```

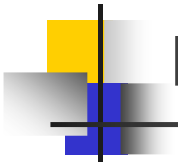


Soluzione esercizio 3

```
create trigger trigger6 on  studenti
after update
as
begin
     insert into log(time,azione,id,before,after)
    select current_timestamp,'update',deleted.id,deleted.nome,inserted.nome
          from inserted, deleted
          where inserted.id = deleted.id
end
```

Soluzione esercizio 3

```
create trigger trigger4 on  studenti
after delete
as
begin
      insert into log(time,azione,id,before,after)
            select current_timestamp,'delete',deleted.id,deleted.nome,null
            from deleted
end
```



Esercizio 4

- Inserire nella tabella studenti un campo media e definire dei trigger che aggiornano automaticamente il campo quando si modificano gli esami
- si tenga presente che se un trigger viene eseguite in modalità **after**, al tempo di esecuzione la tabella su cui opera appare già modificata

Soluzione esercizio 4

```
create trigger trigger7 on  studenti
after delete, insert, update
as
begin
 update studenti set media=(select avg(voto) from esami
                                    where studenti.id=esami.studente)
    where studenti.id in (select  deleted.studente from deleted)
            or studenti.id in (select  inserted.studente from inserted)
end
```

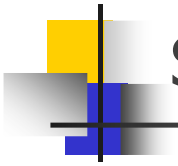
Stored procedure

Stored procedure (functions)

- sono procedure (funzioni) che vengono memorizzate ed eseguite direttamente dal DBMS
- sono scritte in linguaggi proprietari oppure in linguaggi tradizionali con SQL embedded

Vantaggi (rispetto all'implementazione del codice sul client)

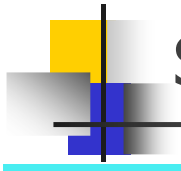
- se la procedura serve a diverse applicazioni, si riduce la duplicazione
- si riduce le comunicazioni fra i client
- si incrementa la capacità di rappresentazione delle viste
- si possono memorizzare sul server le procedure di manutenzione, …
- il DBMS può precompilare le procedure preparando il piano di accesso per ciascuna interrogazione contenuta



Stored procedure

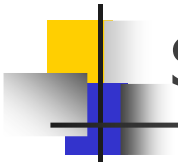
Svantaggi

- si appesantisce il server
- le procedure non sono portabili perchè ogni server ha il suo linguaggio



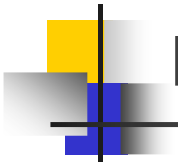
Sintassi procedure TRANSACT-SQL

```
create procedure procedure_name
   { { @parameter data_type }
        [ = default ] [ output ] [readonly]     }
as
sql_statement
-----------------------------------------------------------------------------------
execute   { [ @return_status = ]      procedure_name
        [ [ @parameter = ]       { value |
                                   @variable [ output]|
                                   [ default]}]
```



Stored procedure: osservazioni

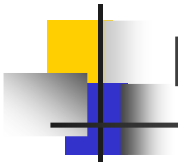
- le procedure possono restituire una o più tabelle temporanee che possono essere accedute da client e da altre procedure
- si osservi che sebbene esistano dei parametri di output, questi non possono contenere tabelle: le tabelle restituite dalle procedure sono quelle che corrispondono alle select della procedura …. vedremo come si accede a questi dati



Esempio

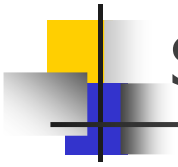
Una procedura per cercare gli esami di uno studente

```
create procedure procedura1
    @nome varchar(10)
as
begin
    select corso,voto from esami, studenti
    where studenti.id=esami.studente and studenti.nome = @nome
end
-----------
exec procedura1 @nome='pippo'
--- oppure    procedura1 'pippo'
```



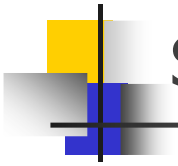
Esercizio 5

- Creare due procedure:
  - la prima dovrebbe prendere in ingresso il nome di uno studente e restituire, usando i parametri di uscita, il massimo e il minimo voto dello studente
  - la seconda dovrebbe visualizzare il nome dei corsi dove uno studente dato ha preso il minimo e il massimo voto



Soluzione esercizio 5

```
create procedure procedura2
    @nome varchar(10),
    @max int output,
    @min int output
as
begin
    select max(voto) from esami, studenti
    where studenti.id=esami.studente and studenti.nome = @nome
end
```



Soluzione esercizio 5

```
create procedure procedura3
as
begin
    declare @max int, @min int
    exec procedura2 'pippo', @max  output, @min  output
end
```

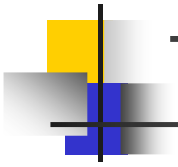
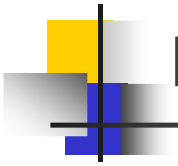


Tabelle temporanee

- TRANSACT-SQL permette la definizione di tabelle temporanee
- le tabelle temporanee possono essere usate attraverso delle variabili di tipo tabella

```
declare @t   table(
    corso varchar(20),
    voto int)

insert into @t
exec procedura1 @nome='pippo`

select max(voto) from @t
```



Esercizio 6

- Creare una procedura che visualizza i corsi che hanno la media massima e minima usando le tabelle temporanee
- si osservi che questa funzionalità non sarebbe implementabile usando una singola interrogazione SQL: serve usare delle tabelle temporanee, una vista oppure delle procedure

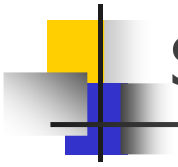
Soluzione esercizio 6

```
create procedure procedura4
begin
    declare @t   table(
            corso varchar(20),
            media float)

    insert into @t
            select corso, avg(voto)
            from esami group by corso

    select corso from @t
    where media= (select max(media) from @t)

    select corso from @t
    where media= (select min(media) from @t)
end
```

Cursori

I cursori

- sono variabili con cui si può scorrere le righe restituite da una interrogazione: seguono la stessa idea che si ritrova, ad esempio, nei resultset di Java
- l'uso dei cursori si ritrova nei linguaggi che estendono SQL e nelle varie versioni di embedded SQL
- il DBMS ottimizza gli accessi ai dati tramite cursori:
  - ad esempio, invece di recuperare per intero il risultato di una interrogazione, può accederci alle righe man mano che queste su rendono necessarie
  - questa strategia è particolarmente utile se i risultati contengono moli enormi di dati



Sintassi dei cursori

- i cursori, in una versione semplificata, erano presenti in SQL-92
- la sintassi di TRANSACT-SQL permette di specificare varie opzioni con cui si può ottimizzare l'efficienza dei cursori

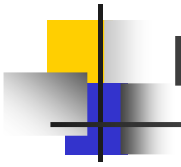
```
declare cursor_name cursor
   [local|global]
   [forward_only | scroll ][ static | keyset | dynamic | fast_forward ]
   [ read_only | scroll_locks| optimistic ]
   for select_statement
   [for update]
```

Sintassi dei cursori

- Diverse istruzione interagiscono con i cursori

open (apre il cursore ed esegue l'interrogazione), fetch (accede ad una riga), update (modifica una riga), close (chiude il cursore), deallocate (dealloca le strutture dati usate dal cursore)

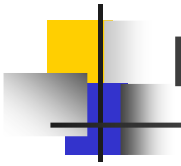
```
open cursor_name
close cursor_name
deallocate cursor_name
fetch   [ [ next | prior | first | last
          | absolute { n | @nvar } | relative { n | @nvar}]
          from] cursor_name [ into { @variable_name}]
update table_name   set       { column_name = expression}
          from table_name  current of  cursor_name }
```



Esempio

```
create procedure procedura4
begin

declare a cursor local for
    select studente from esami

declare @lode bit, @voto int

open a

fetch  a into @voto, @lode
```

```
while @@fetch_status=0
    begin
    if (@lode='true' and @voto<>30)

        update esami set lode='false'
            where current of a

    fetch  a into @voto, @lode
    end
```



Esercizio 7

- Creare una procedura che controlla se c'è una violazione del vincolo referenziale fra studenti.id e esami.studente usando i cursori per scorrere la tabella esami
- Nel caso di una violazione, si registri un evento nel log degli eventi indicando l'id dello studente, il corso e il voto
- Per scrivere sul log si usi

```
xp_logevent error_number, 'message'
```

  il numero di errore deve essere maggiore o uguale a 50000
- per concatenare le stringhe si usi "+", per convertire in varchar si usi

```
cast(@variabile as varchar)
```

Soluzione esercizio 7

```
create procedure procedura5
as
begin

    declare cur cursor local for
          select studente, corso, voto
          from esami

    declare @studente int,
          @corso varchar(10),
          @voto int,
          @mess varchar(50)

    open cur

    fetch  cur    into @studente, @corso, @voto
```

Soluzione esercizio 7

```
 while @@fetch_status=0 begin
   if (not exists(select * from studenti
         where studenti.id=@studente)) begin
               set @mess='violazione vincolo, studente:`
                     +cast(@studente as varchar)+ ',corso: '+@corso
                     +', voto:'+ cast(@voto as varchar)
               exec xp_logevent 50000, @mess
               end
   fetch  cur    into @studente, @corso, @voto
   end
close cur
end
```