Basi di Dati- VIII

Corso di Laurea in Informatica

Anno Accademico 2013/2014

Paolo Baldan

baldan@math.unipd.it

http://www.math.unipd.it/~baldan

SQL per le applicazioni

Programma di massima

- Transazioni
- SQL per le applicazioni
  - PL/SQL
  - Call level interface: ODBC e JDBC
  - Embedded SQL
- Accesso da WEB
  - HTML - cenni
  - PHP - basics
  - form, sessioni, autenticazione

Transazioni

Transazioni

- Un DBMS permette l'accesso "contemporaneo" di vari utenti e di vari processi (o thread)
- Necessità di impedire interazioni indesiderate
- Concetto di transazione

Transazioni: proprietà ACID

Transazione: sequenza di istruzioni collegate che devono essere trattate come una unità indivisibile con proprietà di

Transazioni: proprietà ACID

6

Transazione: sequenza di istruzioni collegate che devono essere trattate come una unità indivisibile con proprietà di

- Atomicità: la transazione ha successo (committed) e produce un effetto visibile, oppure fallisce (aborted) e non ha alcun effetto sulla base di dati

Transazioni: proprietà ACID

6

Transazione: sequenza di istruzioni collegate che devono essere trattate come una unità indivisibile con proprietà di

- Atomicità: la transazione ha successo (committed) e produce un effetto visibile, oppure fallisce (aborted) e non ha alcun effetto sulla base di dati
- Consistenza: la transazione porta la BD da uno stato consistente ad un altro stato consistente (e.g., che soddisfa i vincoli di integrità)

Transazioni: proprietà ACID

Transazione: sequenza di istruzioni collegate che devono essere trattate come una unità indivisibile con proprietà di

- Atomicità: la transazione ha successo (committed) e produce un effetto visibile, oppure fallisce (aborted) e non ha alcun effetto sulla base di dati
- Consistenza: la transazione porta la BD da uno stato consistente ad un altro stato consistente (e.g., che soddisfa i vincoli di integrità)
- Isolamento: le transazioni non interferiscono tra di loro (sono serializzabili)

Transazioni: proprietà ACID

6

Transazione: sequenza di istruzioni collegate che devono essere trattate come una unità indivisibile con proprietà di

- Atomicità: la transazione ha successo (committed) e produce un effetto visibile, oppure fallisce (aborted) e non ha alcun effetto sulla base di dati
- Consistenza: la transazione porta la BD da uno stato consistente ad un altro stato consistente (e.g., che soddisfa i vincoli di integrità)
- Isolamento: le transazioni non interferiscono tra di loro (sono serializzabili)
- Durabilità: l'effetto di una transazione committed è permanente (indipendentemente da interazioni con altre transazioni, fallimenti di sistema, ...)

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Transazioni

- Le proprietà ACID non sono facili da ottenere in un ambiente multi-utente, multi-processore, distribuito.
- Esempio: Data la tabella Conti dei conti bancari

Id	Saldo
01	10000
02	15000
03	2000

- Un trasferimento dal conto 01 al conto 03

```
SELECT Saldo INTO @trasf FROM Conti WHERE Id="01";
UPDATE Saldo=0 WHERE Id="01";
SELECT Saldo INTO @old FROM Conti WHERE Id="03";
UPDATE Conti SET Saldo=@old+@trasf WHERE Id="03”;
```

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Transazioni

- Potrebbe non produrre il risultato desiderato
  - se fallisce a metà o
  - altre query modificano il DB contemporaneamente.
- Es. Nel frattempo potrebbe essere fatto un altro prelievo sul conto con Id="01" ....
- Come si fa?

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Come si fa?

- **Meccanismi di locking**
  - prima di leggere / modificare un dato una transazione lo blocca (in lettura o lettura/scrittura)
    - record
    - table
  - se una transazione T2 tenta di acquisire un lock su di un dato già bloccato da T1, viene messa in attesa
- **Comandi**
  - **lock/unlock** (su tabelle e/o blocchi di righe, gestito dall'utente)
  - **start transaction/commit/rollback** (ci si affida al dbms)

Transazioni con LOCK

- Sintassi

```
LOCK TABLES tabelle { READ | WRITE }
```

- Il meccanismo di lock garantisce accesso esclusivo ad una tabella
  - **WRITE**: solo il thread che acquisisce il lock può leggere o scrivere sulla tabella (gli altri non possono né leggere né scrivere) [eXclusive lock]
  - **READ**: il thread che acquisisce il lock, e così tutti gli altri, possono solo leggere la tabella [Shared lock]

Transazioni con LOCK

- Esempio:

```
LOCK TABLES Conti WRITE
  SELECT Saldo INTO @trasf FROM Conti WHERE Id="01";
  UPDATE Saldo=0 WHERE Id="01";
  SELECT Saldo INTO @old FROM Conti WHERE Id="03";
  UPDATE Conti SET Saldo=@old+@trasf WHERE Id="03";
UNLOCK TABLES
```

- Gestibile anche a livello di record

```
SELECT * FROM ... LOCK IN SHARE MODE
SELECT * FROM ... FOR UPDATE
```

- Lock espliciti utili per velocizzare applicazioni che usano meccanismi transazionali solo raramente. Altrimenti è meglio affidarsi al DBMS ...

Transazioni gestite dal DBMS

- **autocommit on**
  - stato di default
  - ogni comando è una transazione
  - impossibile fare rollback se si raggiunge uno stato indesiderato
- **autocommit off**
  - programmazione esplicita delle transazioni
- MySQL:

```
SET autocommit = 0/1;
```

Transazioni gestite dal DBMS

- è possibile specificare (es. per MySQL InnoDB) che una sequenza di query forma una transazione

```
START TRANSACTION;
   SELECT Saldo INTO @saldo FROM Conti WHERE Id="01";
   UPDATE Saldo=0 WHERE Id="01";
   SELECT Saldo INTO @old FROM Conti WHERE Id="03";
   UPDATE Conti SET Saldo=@old+@trasf WHERE Id="03”;
COMMIT;
```

- Gestisce il lock a livello di ennuple.

Transazioni gestite dal DBMS

- Una transazione inizia con
  - **START TRANSACTION**
  - implicitamente con la prima istruzione su tabelle (SELECT, UPDATE, INSERT)

Transazioni gestite dal DBMS

14

- Una transazione inizia con
  - **START TRANSACTION**
  - implicitamente con la prima istruzione su tabelle (SELECT, UPDATE, INSERT)

- termina con
  - **COMMIT** (successo esplicito)
  - terminazione normale del programma (successo)
  - **ROLLBACK** (fallimento esplicito)
  - terminazione con errore (fallimento)

Transazioni: livelli di isolamento

- Si può specificare il livello di isolamento per non ridurre troppo la concorrenza

```
SET TRANSACTION ISOLATION LEVEL
        { READ UNCOMMITTED |
          READ COMMITTED |
          REPEATABLE READ |
          SERIALIZABLE }
```

- Livello crescente di isolamento

Read Uncommitted

- letture e scritture senza lock
- cambiamenti prodotti da transazioni non committed sono visibili ad altre transazioni
- dirty read

```
T1
y:=x;
```

```
T2
x:=x+1;
rollback
```


Read Committed

- Lock su ennuple
  - in lettura sono rilasciati subito
  - in scrittura sono rilasciati alla fine della transazione
- Cambiamenti prodotti da transazioni committed sono visibili ad altre transazioni (due read ripetute possono dare risultati diversi!)

```
T1
y:=x;
z:=x;
```

```
T2
x:=x+1;
```


Repeatable read

- lock in lettura e scrittura per l'intera transazione

Repeatable read

- lock in lettura e scrittura per l'intera transazione
- se all'interno di una singola transazione si effettuano più select sulla singola riga il risultato è garantito essere sempre lo stesso

```
SELECT Saldo FROM Conti WHERE Id=11;
```

Repeatable read

- lock in lettura e scrittura per l'intera transazione
- se all'interno di una singola transazione si effettuano più select sulla singola riga il risultato è garantito essere sempre lo stesso

```
SELECT Saldo FROM Conti WHERE Id=11;
```

- però se si selezionano più righe

```
SELECT avg(Saldo) FROM Conti WHERE Id>11;
```

la seconda interrogazione potrebbe selezionare più record (inseriti da una transazione committed) -> valore diverso!

Repeatable read

18

- lock in lettura e scrittura per l'intera transazione
- se all'interno di una singola transazione si effettuano più select sulla singola riga il risultato è garantito essere sempre lo stesso

```
SELECT Saldo FROM Conti WHERE Id=11;
```

- però se si selezionano più righe

```
SELECT avg(Saldo) FROM Conti WHERE Id>11;
```

la seconda interrogazione potrebbe selezionare più record (inseriti da una transazione committed) -> valore diverso!

- vari workaround (es. index-gap locking per InnoDB)

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

Serializable

- Le transazioni producono un effetto che si potrebbe avere eseguendole in modo sequenziale, senza alcun interleaving
- T1 || T2 si comporta come T1 ; T2 oppure come T2 ; T1
- Maggiore è l'isolamento più l'esecuzione risulta lenta (sia per le attese legate ai lock, sia per la necessita di prevedere l'interazione tra transazioni).

Transazioni: dimensioni

- Devono essere piccole più possibile
  - per ridurre la dimensione dei blocchi "logici"
  - aumentare la concorrenza
- problemi nello spezzare in parti le transazioni .. occorrono più controlli!
  - Esempio: se realizzo come due transazioni separate
    - verifica della presenza del prodotto
    - vendita
  - prima della vendita devo verificare alcuni aspetti (quantità di prodotto disponibile)

Transazioni: deadlock e ripetizione

- Quando c'è concorrenza ... ci sono **deadlock**!
  - I DBMS verificano automaticaticamente la presenza di deadlock (e timeout)
  - con uccisione delle transazioni per risolvere i deadlock
    (le transazioni vengono terminate una alla volta iniziando dalle più piccole)
- In applicazioni che operano con transazioni
  - verificare lo stato della transazione quando termina
    (codice di errore)
  - ripetizione esplicita nel caso la transazione sia stata terminata per deadlock

SQL e linguaggi di programmazione

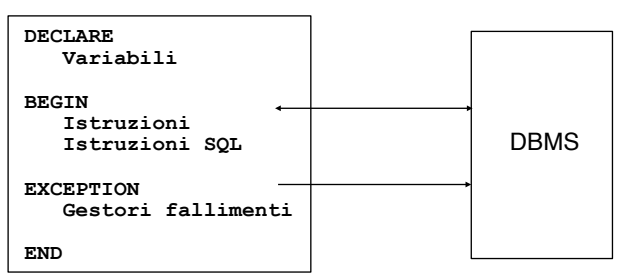
Friday, May 23, 2014

SQL e linguaggi di programmazione

- Inizialmente: SQL pensato per l'uso interattivo da parte di utenti non esperti
- Insufficiente:
  - difficile da usare
  - necessità di scrivere applicazioni complesse
    - applicazioni gestionali
    - analisi complesse
    - applicazioni web
- Tendenza attuale: SQL come componente dedicato alla gestione dei dati + linguaggio di programmazione

Uso di SQL da programmi: problemi

- Come collegarsi alla BD ?
- Come utilizzare gli operatori SQL ?
- Come trattare il risultato di un comando SQL (relazioni) ?
- Come scambiare informazioni sull'esito delle operazioni ?



Uso di SQL da programmi: soluzioni

- **Linguaggio integrato (dati e DML)**
  - linguaggio ad hoc (4GL, es. Oracle PL/SQL)
  - controllo statico/compilazione delle query

Uso di SQL da programmi: soluzioni

25

- **Linguaggio integrato (dati e DML)**
  - linguaggio ad hoc (4GL, es. Oracle PL/SQL)
  - controllo statico/compilazione delle query

- **Linguaggio convenzionale + API**
  - funzioni di libreria per l'uso di SQL (ODBC, JDBC)
  - query passate come stringhe a funzioni (controllo dinamico)
  - ORM (object-relational mapping)

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Uso di SQL da programmi: soluzioni

25

- **Linguaggio integrato (dati e DML)**
  - linguaggio ad hoc (4GL, es. Oracle PL/SQL)
  - controllo statico/compilazione delle query
- **Linguaggio convenzionale + API**
  - funzioni di libreria per l'uso di SQL (ODBC, JDBC)
  - query passate come stringhe a funzioni (controllo dinamico)
  - ORM (object-relational mapping)

- **Linguaggio che ospita l'SQL**
  - costrutti per marcare i comandi SQL
  - un preprocessore lo traduce in linguaggio convenzionale + API

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Linguaggio integrato

PL/SQL

Friday, May 23, 2014

PL/SQL

- Linguaggio offerto dal DBMS, che estende SQL
  - variabili scalari (e record)
    - con i tipi dei domini relazionali
  - assegnamento
  - costrutti di controllo
    - for, while, ...
  - cursori
    - per operare in modo iterativo sulle ennuple restituite da una select
  - eccezioni
  - procedure, moduli, funzioni, trigger

Aspetti procedurali

- Codifica nello schema di informazione procedurale
  - Stored procedure
    - funzioni e procedure
    - eseguite dal DBMS su richiesta delle applicazioni
  - Trigger
    - procedure memorizzate
    - eseguite automaticamente dal DBMS al verificarsi di dati eventi

Stored Procedures: Perché?

- Permettono di condividere il codice tra applicazioni, con una gestione centralizzata
- Garantiscono una semantica prefissata per le operazioni sulla BD che implementano
- Permettono di implementare vincoli di integrità semantici, non esprimibili nel modello dei dati
- Riducono il traffico di rete
- Migliorano sicurezza e astrazione

Funzioni

- Sintassi molto variabile (qui MySQL)
- Es.:

```
CREATE FUNCTION contaStudenti () RETURNS INT
BEGIN
  DECLARE numStudenti INT;
  SELECT COUNT(*) INTO numStudenti FROM Studenti;
  RETURN numStudenti;
END
```

Funzioni

- Una volta definite sono richiamabili come normali funzioni predefinite.
  - `SELECT contaStudenti();`
  - `SELECT ... WHERE contaStudenti() > ...`

Funzioni

31

- Una volta definite sono richiamabili come normali funzioni predefinite.
  - `SELECT contaStudenti();`
  - `SELECT ... WHERE contaStudenti() > ...`

- Es: Media degli esami sostenuti da almeno un terzo degli studenti:

```
SELECT Materia, AVG(Voto)
FROM Esami
GROUP BY Materia
HAVING COUNT(*) > contaStudenti()/3;
```

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

Funzioni

- Le funzioni possono avere dei parametri di input
- Esempio:

  Media dei voti per una provincia, ritorna A, B o C se la media è negli intervalli
  - A: [18-22]
  - B: (22,26]
  - C: (26,30]

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

Funzioni


```
CREATE FUNCTION Media (AProvincia CHAR(2)) RETURNS CHAR(1)
BEGIN
    DECLARE Media FLOAT;
    DECLARE Res CHAR(1);

    SELECT AVG(e.Voto) INTO Media
    FROM Studenti s JOIN Esami e ON (s.Matricola=e.Candidato)
    WHERE s.Provincia=AProvincia;

    IF Media BETWEEN 18 AND 22 THEN
       SET Res="A";
    ELSEIF Media <= 26 THEN
       SET Res="B";
    ELSE
       SET Res="C";
    END IF;

    RETURN Res;
END
```

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

- Procedura per l'inserimento di uno studente

```
CREATE PROCEDURE insStud
    (IN  num INT, ANome VARCHAR(10), ACognome VARCHAR(10),
         AMatricola CHAR(6), AProvincia CHAR (2),
     OUT ok BOOL)

BEGIN
  DECLARE numStudenti INT;
  SELECT COUNT(*) INTO numStudenti FROM Studenti;
  IF numStudenti<num THEN
     SET ok=TRUE;
     INSERT INTO Studenti (Nome, Cognome, Matricola, Provincia)
                  VALUES (ANome,ACognome,AMatricola,AProvincia);
  ELSE
     SET ok=FALSE;
  END IF;
END
```


- La procedura si richiama con `CALL`
  - Es.

    `CALL insStud(7,'Giulio','Cesare','71228','BL',@o);`
  - Nota: `@o` variabile utente (si opera con `SET @o= ...`, ecc.)

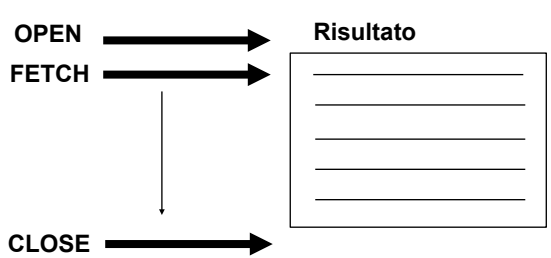
- NOTA: Occorre cambiare i delimitatori

delimiter $

<definizione>$

delimiter ;

- Meccanismo per ottenere uno alla volta gli elementi di una relazione
  - Definito con un'**espressione SQL**, poi
  - Si **apre** per far **calcolare** al DBMS il risultato e poi
  - con un opportuno comando si trasferiscono i campi della prossima ennupla in opportune variabili del programma.



Cursore: sintassi

- Dichiara il cursore, associandogli una query
  - `DECLARE cursor_name CURSOR FOR SELECT_statement;`

Cursore: sintassi

38

- Dichiara il cursore, associandogli una query
  - `DECLARE cursor_name CURSOR FOR SELECT_statement;`
- Apre il cursore
  - `OPEN cursor_name;`

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Cursore: sintassi

38

- Dichiara il cursore, associandogli una query
  - `DECLARE cursor_name CURSOR FOR SELECT_statement;`
- Apre il cursore
  - `OPEN cursor_name;`
- Recupera la prossima riga, memorizzandone i campi in una lista di variabili
  - `FETCH cursor_name INTO variable list;`

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Cursore: sintassi

38

- Dichiara il cursore, associandogli una query
  - `DECLARE cursor_name CURSOR FOR SELECT_statement;`
- Apre il cursore
  - `OPEN cursor_name;`
- Recupera la prossima riga, memorizzandone i campi in una lista di variabili
  - `FETCH cursor_name INTO variable list;`
- Chiude il cursore
  - `CLOSE cursor_name;`

7. SQL per le applicazioni Corso di Basi di Dati

Friday, May 23, 2014

Cursore: quando finiscono i dati

- Quando non ci sono più dati si solleva un errore di tipo "**NOT FOUND**" che va catturato con un **HANDLER**
  - **DECLARE** <type> **HANDLER FOR** <cond>
    statement
  - **<type>** -> **CONTINUE, EXIT**
  - **<cond>** -> condizione di errore (specifico valore o classi `SQLWARNING, NOT FOUND, SQLEXCEPTION`)

Cursore: quando finiscono i dati

39

- Quando non ci sono più dati si solleva un errore di tipo "**NOT FOUND**" che va catturato con un **HANDLER**
  - **DECLARE** <type> **HANDLER FOR** <cond>
    statement
  - **<type>** -> **CONTINUE, EXIT**
  - **<cond>** -> condizione di errore (specifico valore o classi `SQLWARNING, NOT FOUND, SQLEXCEPTION`)
- Esempio:
  - **DECLARE CONTINUE HANDLER FOR NOT FOUND**
    SET Done = 1

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

Cursore: Esempio


```
CREATE PROCEDURE Migliori (IN Soglia INT) /* fa qualcosa sulle materie */
BEGIN                                     /* con media > Soglia       */
  DECLARE Done INT DEFAULT 0;
  DECLARE Media FLOAT;
  DECLARE CMateria CHAR(3);

  DECLARE Cur CURSOR FOR
          SELECT Materia, AVG(Voto) FROM Esami GROUP BY Materia;

  DECLARE CONTINUE HANDLER FOR NOT FOUND
          SET Done = 1;

  OPEN Cur;
  REPEAT
     FETCH Cur INTO CMateria, Media;
     IF NOT Done THEN
         IF Media > Soglia THEN
            ...
         END IF;
     END IF;
  UNTIL Done END REPEAT;
  CLOSE cur;
END
```

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

Cursore: Esempio


```
CREATE PROCEDURE Migliori (IN Soglia INT) /* fa qualcosa sulle materie */
BEGIN                                     /* con media > Soglia       */
  DECLARE Media FLOAT;
  DECLARE CMateria CHAR(3);

  DECLARE Cur CURSOR FOR
          SELECT Materia, AVG(Voto) FROM Esami GROUP BY Materia;

  BEGIN
    DECLARE EXIT HANDLER FOR NOT FOUND
            BEGIN END;

    OPEN Cur;
    LOOP
      FETCH Cur INTO CMateria, Media;
      IF Media > Soglia THEN
            ...
      END IF;
    END LOOP;
  END;
  CLOSE cur;
END
```

7. SQL per le applicazioni Corso di Basi di Dati
Friday, May 23, 2014

Variabili

- **Assegnamento**
  - **SET** @var=expr
  - **SET** @var:= ...
- Es.

```
SET @a=0;
SELECT if(@a,@a:=@a+1,@a:=1) AS Num, s.Cognome, s.Nome
FROM Studenti s;

SELECT (@nome:=s.Nome)
FROM   Studenti s
WHERE  s.Matricola='71347'
```


Variabili: ordine di valutazione

- Quello già indicato per SELECT ...

```
SET @a=71347;

SELECT if(@a,@a:=@a+1,@a:=1) AS Num, s.Matricola,
       s.Cognome, s.Nome
FROM   Studenti s
WHERE  s.Matricola=@a;
```

```
+-------+-----------+---------+---------+
| Num   | Matricola | Cognome | Nome    |
+-------+-----------+---------+---------+
| 71348 | 71347     | Zeri    | Giorgio |
+-------+-----------+---------+---------+
```


Trigger

- Procedure eseguite dal DBMS in corrispondenza del verificarsi di un determinato evento
  - **CREATE TRIGGER** Nome
    PrimaODopoDi Evento {, Evento}
    ON Tabella
    [Granularità]
    Azione

```
PrimaODopoDi  := BEFORE | AFTER

Evento := INSERT | DELETE | UPDATE OF Attributi

Granularità :=  FOR EACH ROW | FOR EACH STATEMENT
```


Esempio di Trigger

- Controlla se il voto da inserire per un esame è nel range corretto

```
CREATE TRIGGER ControlloVoto
BEFORE INSERT ON Esami
FOR EACH ROW
BEGIN
  IF (NEW.Voto < 18) OR (NEW.Voto > 30)
      THEN RAISE_APPLICATION_ERROR(-21, 'Voto errato');
  END IF;
END;
```

- MySQL non offre `RAISE_APPLICATION_ERROR`.

Esempio di Trigger


```
CREATE TRIGGER ControlloStipendio
BEFORE INSERT ON Impiegati
BEGIN
  DECLARE StipendioMedio FLOAT;

  SELECT avg(Stipendio) INTO StipendioMedio
      FROM   Impiegati
      WHERE  Dipartimento = NEW.Dipartimento;
  IF NEW.Stipendio > 2 * StipendioMedio
      THEN RAISE_APPLICATION_ERROR(-2061, 'Stipendio alto')
  END IF;
END;
```


Esempio di Trigger


```
CREATE TRIGGER ControlloStipendio
BEFORE UPDATE ON Impiegati
BEGIN
  IF NEW.Stipendio < OLD.Stipendio
     THEN RAISE_APPLICATION_ERROR(-2068, 'Lo stipendio non
                                          può diminuire')
  END IF;
END;
```


Trigger Attivi e Passivi

- **Trigger Passivi**: verificano una condizione che, se non soddisfatta, determina il fallimento della transazione
  - vincoli di integrità dinamici
  - accessi di utenti che dipendono dallo stato della BD (funz. **USER()**)

Trigger Attivi e Passivi

- **Trigger Passivi**: verificano una condizione che, se non soddisfatta, determina il fallimento della transazione
  - vincoli di integrità dinamici
  - accessi di utenti che dipendono dallo stato della BD (funz. **USER()**)
- **Trigger Attivi**: in corrispondenza di un evento, modificano lo stato della BD
  - business rules (manuntenzione di archivi, generazione ciclica di rapporti, invio di solleciti, ...)
  - memorizzazione di eventi nella BD (logging, auditing)
    - solo transazioni terminate? Dipende dalla semantica dei trigger
  - propagazione ad altre tabelle degli effetti delle modifiche
  - allineamento di dati duplicati
  - vincoli di integrità (es. referenziale, se non gestita dal DBMS)

Vantaggi dei Trigger

- semplifica le applicazioni, sgravate dei controlli nei trigger
- gestione centralizzata dei controlli, con ovvi vantaggi
- vincoli di integrità complessi

Svantaggi dei Trigger

- **Semantica non ovvia**
  - Quando viene eseguito (es. per una update)? Ad ogni riga aggiornata o per ogni statement?
    - Oracle e Postgres permettono di definirlo
    - MySQL per riga
  - Ordine di esecuzione se più trigger sono attivi
  - Effetto cascata e terminazione
  - Parte della transazione o transazione separata?
- **Problemi di progettazione**
  - difficile controllare l'effetto complessivo dei trigger
  - rigidità (stati intermedi non consistenti?)

SQL embedded

SQLJ: SQL embedded in Java

- Standard ISO per istruzioni SQL immerse in programmi Java

  Tradotto da un precompilatore in Java standard + API

```
public static void main(String argv[]){
  Oracle.connect("jdbc:oracle:oci:@localhost:1521:mydb",
                 "baldan", "pwd");
  #SQL iterator GetNomeIter(String Nome, int AnnoNascita);
  GetNomeIter iter;
  #SQL iter = {SELECT Nome, AnnoNascita
               from Studenti where Provincia =:(argv[0])};
  System.out.println("Nomi trovati:");
  while (iter.next()) {
      String nome = iter.Nome();
      int anno = iter.Anno();
      System.out.println("Nome =" + nome + "Anno:" + anno);
  }
  iter.close();
  Oracle.close();
}
```

SQLJ: SQL embedded in Java


```
...
#SQL { SELECT Nome into :nome
       FROM   Studenti
       WHERE  Matricola = :matricola }

System.out.println("Nome =" + nome +
                   "Matricola:" + matricola);
```


SQL embedded in C


```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_annoNascita;
EXEC SQL END DECLARE SECTION
short c_Provincia = "Padova";
EXEC SQL DECLARE stud_info CURSOR FOR
  SELECT S.nome, S.annoNascita
  FROM Studenti S
  WHERE S.Provincia = :c_Provincia
  ORDER BY S.nome;
do {
  EXEC SQL FETCH stud_info INTO :c_snome, :c_annoNascita;
  printf("Nome:%s; AnnoNascita: %s \n ", c_snome, c_annoNascita);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE stud_info;
```


Linguaggio + API

Linguaggio + API

- Libreria (API) di funzioni/classi che operano su basi di dati
  - ricevono come parametro stringhe SQL
  - ritornano il risultato con una logica ad iteratori
- Standard
  - ODBC (origine Microsoft - standard su Windows)
  - JDBC (origine Sun) è l'equivalente in Java
- Dovrebbero essere indipendenti dal DBMS
  - un "driver" gestisce le richieste e le traduce per lo specifico DBMS
  - la BD può essere in rete

JDBC


```
public static void main(String argv[]){
  DriverManager = Class.forName("com.mysql.jdbc.Driver"); // driver per DBMS

  Connection con = // connect
            DriverManager.getConnection("url", "login", "pass");

  Statement stmt = con.createStatement(); // oggetto per comando SQL
  String query = "SELECT Nome, AnnoNascita
                  FROM Studenti WHERE Provincia ='" + argv[0] + " '";
  ResultSet iter = stmt.executeQuery(query);
  System.out.println("Nomi trovati:");
  try { // gestore eccezioni
    while (iter.next()) {   // ciclo sul risultato
        String nome = iter.getString("Nome");
        int anno = iter.getInt("AnnoNascita");
        System.out.println(" Nome: " + nome + "; AnnoNascita: " + anno);
    }
  } catch(SQLException ex) {
    System.out.println(ex.getMessage()+ex.getSQLState()+ex.getErrorCode());}
  stmt.close(); con.close();
}
```


ORM (Object relational mapping)

- Come risolvere il problema di impedenza, conciliando il DBMS relazionale con un linguaggio ad oggetti
  - Varie tecniche basate sulla creazione di uno strato software che 'virtualizza' l'accesso al DB
  - ~ Persistent objects

ORM

- **Idee**:
  - **tabella** del DB <-> **classe**
  - **record** della tabella <-> **oggetto** istanza della classe
  - **colonne** della tabella <-> **attributi** dell'oggetto
  - **instance methods** (**crud** - create, read, update, delete)

```
studente = new Studente(Lino, ....);
studente.save();
```

    - NB: serve sapere se lo studente è nuovo o meno (update o insert?)
  - **class methods -** query sulla tabella
    - gli studenti con voti > k: **`Studenti.getVoti(k))`**

ORM - gestione delle associazioni

- **associazioni univoche** codificate come
  - **metodi** che ritornano un oggetto
    - **`studente.Tutor();`**
  - o semplicemente come **attributi** (di tipo oggetto)
    (lazy fetch evita che caricando un oggetto si carichino oggetti non utili ad esso collegati - es. potrei non voler sapere niente del tutor)
- **associazioni multivalore**
  - metodi che ritornano insiemi (array, ...) di oggetti
    - **`studente.Esami()`**
  - impossibile dimenticarsi di SQL e del database sottostante ... alcune funzionalità lo richiederanno

ORM

- riferimenti
  - Hibernate - Java (il primo ~ 2002)
  - EDM - C# (Entity Data Model) - Microsoft
  - Active records (Ruby on Rails, CakePHP, ...)
- Sulla base di persistenza poggiano vari Web Application Framework basati sul design pattern MVC (Model View Control)

E noi?

- Noi utilizzeremo come linguaggio "ospite" **PHP**
- con l'approccio **linguaggio + API**
- sfruttando una API specifica per MySQL