Basi di Dati Attive

Basi di Dati Passive

- le politiche di reazione nei vincoli d'integrità referenziale sono il primo esempio della necessità di introdurre un comportamento **reattivo** nelle basi di dati "mettendo a fattor comune" parte del comportamento attivo

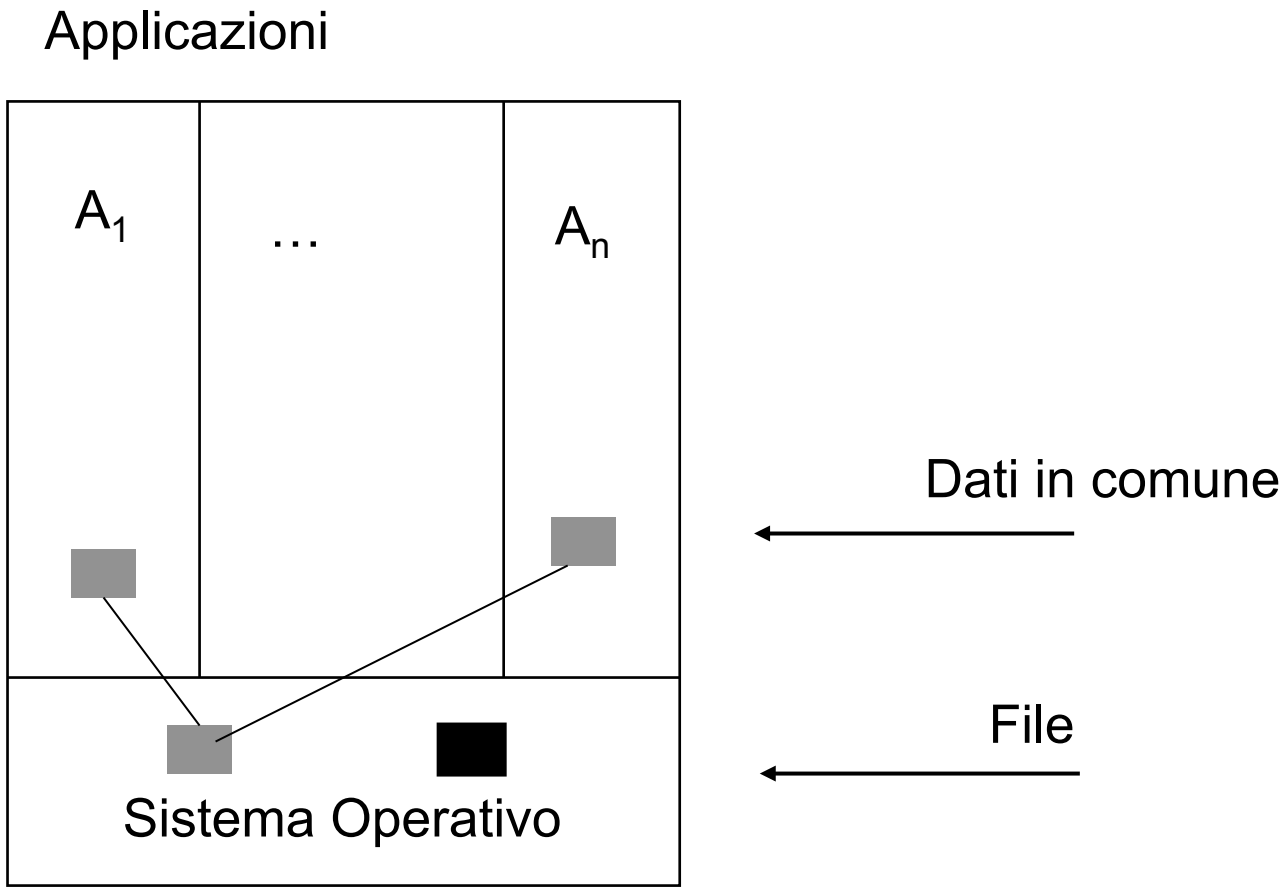
```
on < delete | update >
   < cascade | set null | set default |
     no action >
```

- l'idea è di introdurre dei costrutti linguistici dedicati a questo scopo.
- tali costrutti sono detti regole (attive) e mettono a fattor comune parte del comportamento procedurale dell'applicazione che è così "condivisa" tra le varie applicazioni realizzando "l'indipendenza della conoscenza

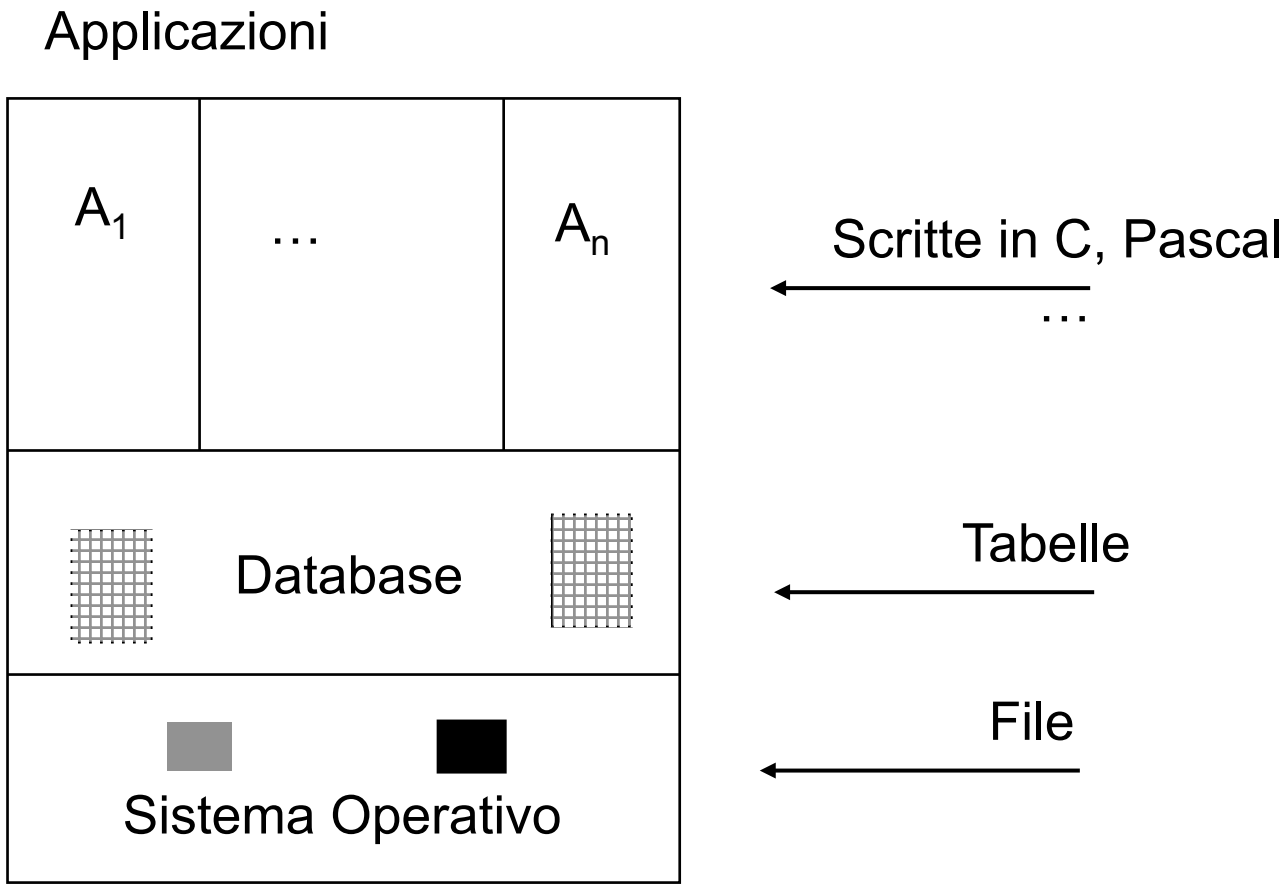
Basi di Dati Attive

- BD con una componente per la gestione di regole attive **Evento-Condizione-Azione** (regole di produzione con eventi):
  - eventi: modifiche alla base di dati
  - valuta una condizione e, in base al valore di verità
  - esegue una o più azioni
- hanno comportamento **reattivo** (in contrasto con **passivo**): eseguono non solo le transazioni utenti ma anche le regole
- i DBMS commerciali (SQL3) usano i **trigger** per modellare le regole che sono definite, come nello schema

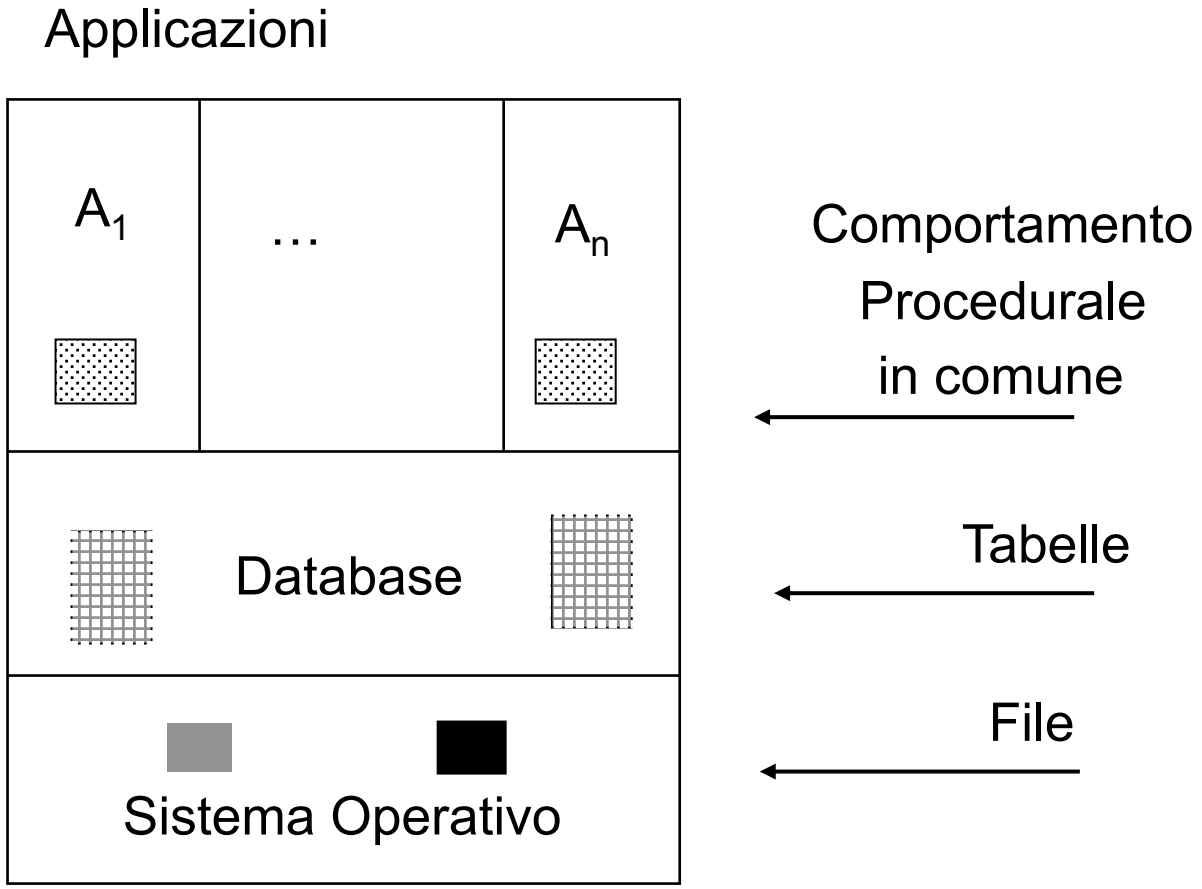
Anni 70: senza DBMS



Anni 80: con DBMS



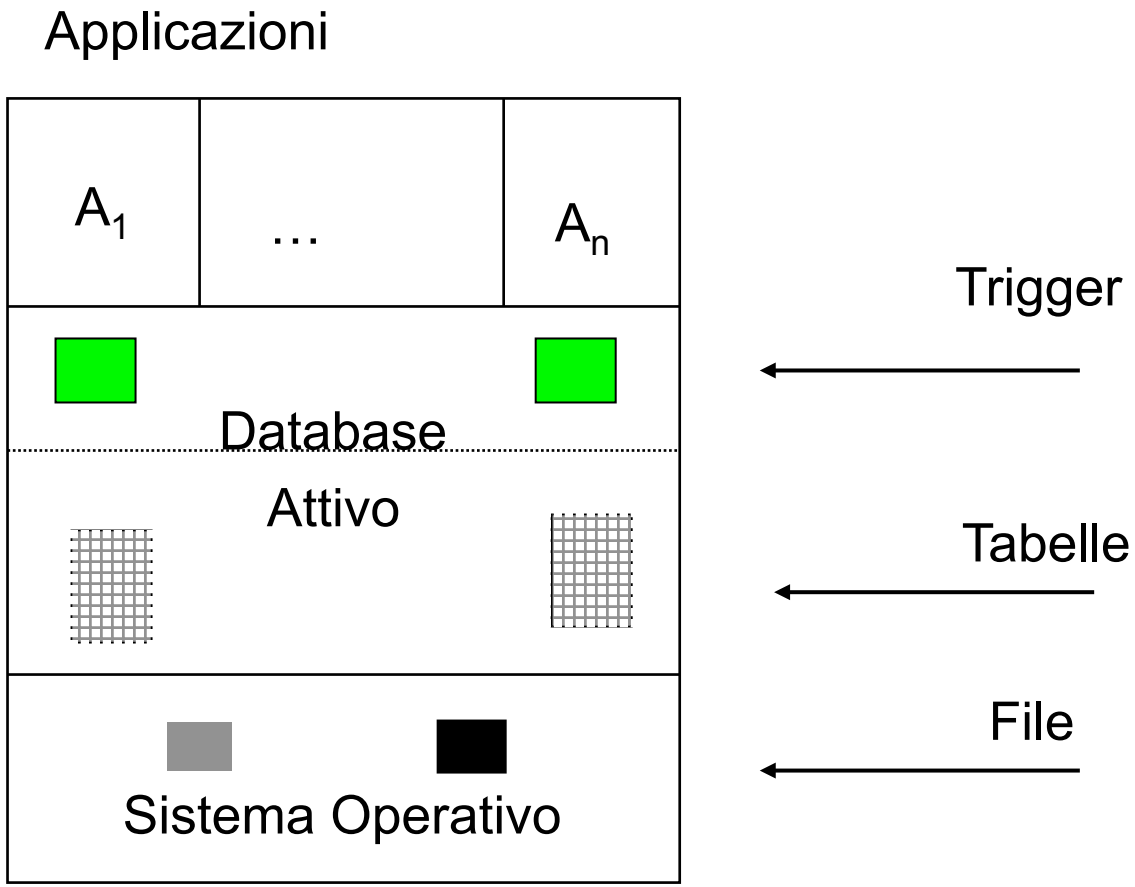
Anni 80: con DBMS e stored procedure



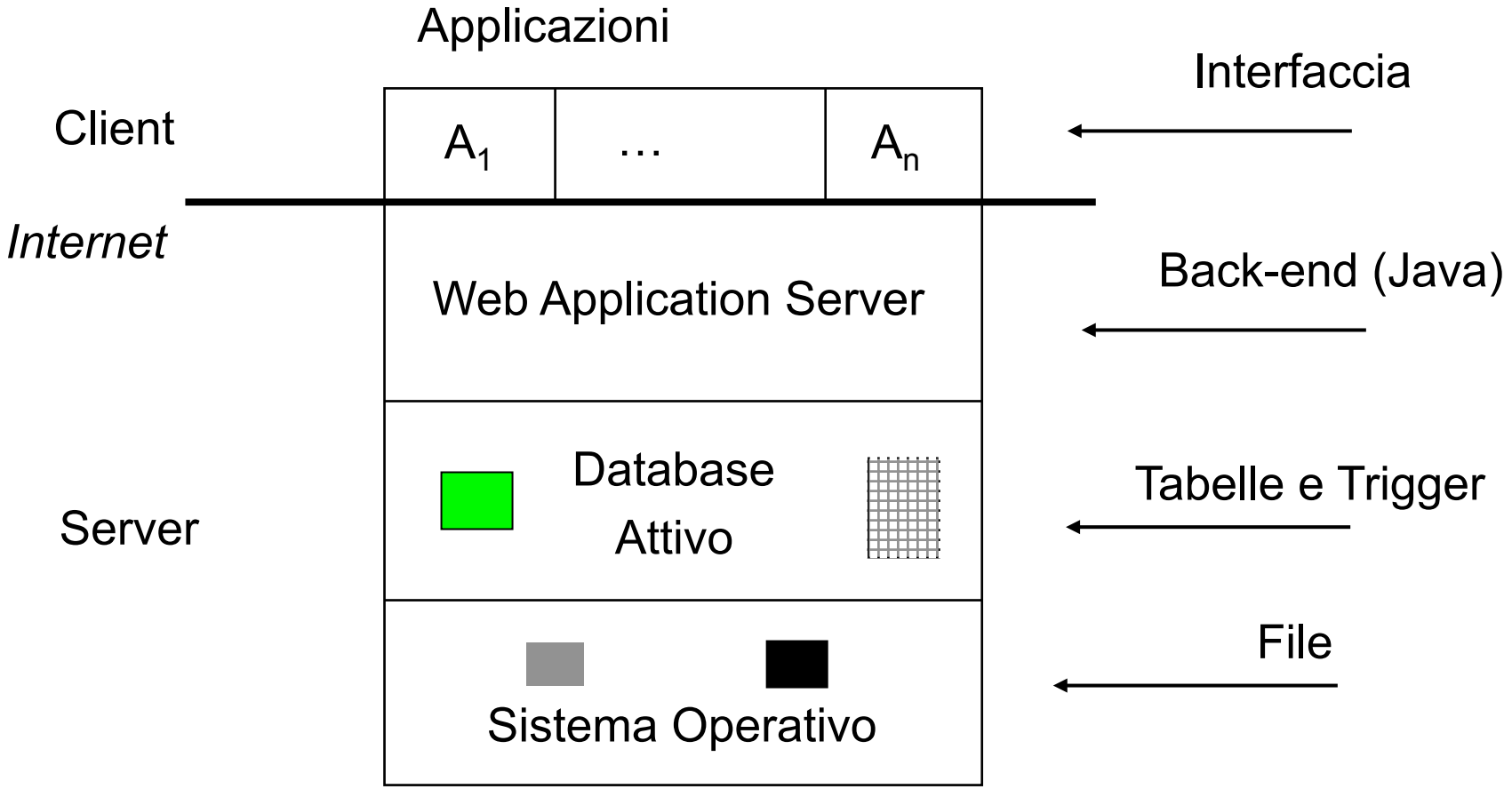
Store procedure

- Le stored procedure (già viste) sono state un'altra alla necessità di esprimere il comportamento procedurale messo a fattor comune tra le varie applicazioni dal DBMS a livello di schema
- Le stored procedure non aderiscono a nessuno standard ed hanno il problema dell'impedance mismatch con il linguaggio usato per definirle
- Il risultato è l'introduzioe dei trigger a livello di schema che devono modellare il comportamento procedurale messo a fattor comune tra le varie applicazioni e sono gestite sotto il controllo del DBMS

Anni 90: DBMS Attivo



Evoluzione (anni 00)



Trigger

- definiti con istruzioni DDL (`create trigger`)
  - basati sul paradigma evento-condizione-azione (ECA):
    evento: modifica dei dati, specificata con `insert, delete, update`
    condizione (opzionale) predicato SQL
    azione: sequenza di istruzioni SQL (o estensioni, ad esempio PL/SQL in Oracle)
  - intuitivamente:
    quando si verifica l'evento (attivazione)
    se la condizione è soddisfatta (valutazione)
    allora esegui l'azione (esecuzione)
  - ogni trigger fa riferimento ad una tabella (target): risponde ad eventi relativi a tale tabella

Trigger: granularità e modalità

- granularità
  - di ennupla (row-level): attivazione per ogni ennupla coinvolta nell'operazione
  - di operazione (statement-level): una sola attivazione per ogni istruzione SQL, con riferimento a tutte le ennuple coinvolte ("set-oriented")
- modalità
  - immediata: subito dopo (o subito prima) dell'evento
  - differita: al commit

Modello computazionale

- Sia $T^U = U_1; \ldots ; U_n$ la transazione utente
- Se le regole di P hanno la forma E, C →A con E, Evento, C, Condizione ed A, Azione, allora

- Sematica **Immediata** genera $T^I = U_1; \underline{U^P_1}; \ldots U_n; \underline{U^P_n}$
- Sematica **Differita** genera $T^D = U_1; \ldots U_n; \underline{U^P_1}; \ldots; \underline{U^P_n}$

- Dove $\underline{U^P_i}$ rappresenta la sequenza di azioni indotte da U_i su P
- Problemi:
  - Confluenza
  - Terminazione
  - Equivalenza

Trigger in Oracle, sintassi

```
create trigger TriggerName
  Mode Event {, Event}
  on TargetTable
  [[referencing Reference]
    for each row
    [when SQLPredicate]]
    PL/SQLBlock
```

- *Mode:* `before` o `after`
- *Event:* `insert, update, delete`
- `for each row` specifica la granularità
- *Reference:* permette di definire nomi di variabili (utilizzabili solo per granularità di ennupla):

  `old as` *OldVariable* | `new as` *NewVariable*

Trigger in Oracle, semantica

- modalità **immediata** (sia `after` sia `before`)
- schema di esecuzione:
  - trigger `before statement`
  - per ogni ennupla coinvolta:
    - trigger `before row`
    - operazione
    - trigger `after row`
  - trigger `after statement`
- in caso di errore, si disfa tutto
- priorità fra i trigger
- massimo 32 trigger attivati in cascata

Trigger in Oracle, esempio

```
create trigger Reorder
   after update of QtyAvbl on Warehouse
   when (new.QtyAvbl < new.QtyLimit)
   for each row
       declare X number;
   begin
       select count(*) into X
       from PendingOrders
       where Part = new.Part;
       if X = 0
       then
              insert into PendingOrders
              values (new.Part, new.QtyReord, sysdate);
       end if;
   end;
```

Trigger in Oracle, esempio, 2

WAREHOUSE

Part	QtyAvbl	QtyLimit	QtyReord
1	200	150	100
2	780	500	200
3	450	400	120

```
T1: update Warehouse
        set QtyAvbl = QtyAvbl - 70
        where Part = 1

T2: update Warehouse
        set QtyAvbl = QtyAvbl - 60
        where Part <= 3
```

Trigger in DB2, sintassi

```
create trigger TriggerName
  Mode Event on TargetTable
  [referencing Reference]
  for each Level
  [when (SQLPredicate)]
    SQLProceduralStatement
```

- *Mode:* `before` o `after`
- *Event:* `insert, update, delete`
- `for each` *Level* specifica la granularità
- *Reference:* permette di definire nomi di variabili (a seconda della granularità):

```
      old as OldTupleVar | new as NewTupleVar
old_table as OldTableVar | new_table as NewTableVar
```

Trigger in DB2, semantica

- modalità **immediata** (sia `after` sia `before`)
- i `before` trigger non possono modificare la base di dati, a parte varianti sulle modifiche causate dall'evento (e quindi non possono in generale attivare altri trigger)
- in caso di errore, si disfa tutto
- nessuna priorità fra i trigger (ordine definito dal sistema), interazione con le azioni compensative sui vincoli di integrità referenziale
- massimo 16 trigger attivati in cascata

Trigger in DB2, esempio

```
foreign key (Supplier)
   references Distributor
   on delete set null

create trigger SoleSupplier
   before update of Supplier on Part
   referencing new as N
   for each row
   when (N.Supplier is not null)
   signal sqlstate '70005' ('Cannot change supplier')

create trigger AuditPart
   after update on Part
   referencing old_table as OT
   for each statement
   insert into Audit
   values(user, current-date, (select count(*) from OT))
```

Estensioni (di solito non disponibili)

- eventi temporali (anche periodici) o “definiti dall’utente”
- combinazioni booleane di eventi
- clausola instead of: non si esegue l'operazione che ha attivato l'evento, ma un’altra azione
- esecuzione “distaccata”: si attiva transazione autonoma
- definizione di priorità
- regole a gruppi, attivabili e disattivabili
- regole associate anche a interrogazioni (non solo aggiornamenti)

Proprietà delle regole

- terminazione (essenziale)
- confluenza
- determinismo delle osservazioni

Applicazioni

- funzionalità interne
  - controllo dei vincoli di integrità
  - replicazione
  - gestione delle viste
    - materializzate: propagazione
    - virtuali: modifica delle interrogazioni
- funzionalità applicative: descrizione della dinamica (comportamento) delle basi di dati