PROGRAMMAZIONE

Stored procedure

Estende l'SQL con un linguaggio computazionalmente completo.

Le procedure vengono memorizzate all'interno del DBMS.

Resta il problema dell'interazione con l'utente.

Definizione

Una stored procedure è un segmento di codice composto da istruzioni SQL dichiarative e procedurali, che viene memorizzato all'interno della base di dati stessa come parte dello schema, e che può essere invocato da un programma, da una regola attiva (in inglese "trigger") o da un'altra stored procedure.
Dopo che la stored procedure è stata definita è quindi possibile utilizzarla come se rientrasse nell'insieme dei comandi SQL predefiniti, richiamandola e passandole eventualmente dei parametri.
Più precisamente la stored procedure viene memorizzata nel catalog del database, cioè nell'insieme delle tabelle in cui sono descritti tutti gli oggetti del database.

Pregi e difetti di una stored procedure

Il progettista di una base di dati può sfruttare i vantaggi delle stored procedure.
In modo particolare i pregi di una stored procedure sono legati:

- all'aumento delle prestazioni, poiché una stored procedure compilata e memorizzata nel catalog della base di dati viene eseguita più facilmente e velocemente rispetto a delle normali istruzioni SQL che devono essere interpretate di volta in volta.
- alla possibilità di creare regole di business legate ai dati e controlli d'integrità, grazie all'interazione coi trigger, che controllano la violazione dei vincoli, e le stored procedure, che permettono di mettere in atto un'adeguata azione. Il vantaggio dell'utilizzo di una stored procedure, in questi casi, consiste nel fatto che la procedura è memorizzata nel catalog, e può essere richiamata ogni qualvolta si riscontra la violazione per cui è stata progettata.
- alla riduzione del traffico tra l'applicazione che si appoggia al database e il server del database stesso, in quanto con una stored procedure è sufficiente mandare al server il nome della procedura e attendere il risultato. Al contrario se si utilizzano delle istruzioni SQL compilate direttamente dal client, il traffico di dati necessario per ottenere il risultato è maggiore, poiché i risultati delle singole istruzioni vengono tutti scambiati tra il server ed il client.
- al possibile riutilizzo di una stored procedure, in quanto è uno strumento che può essere richiamato da ogni applicazione che utilizza la base di dati. Infatti le stored procedure estendono l'interfaccia della base di dati per tutte le possibili applicazioni, permettendo allo sviluppatore del database di non dover programmare le funzioni che sono supportate dalla stored procedure in ogni possibile applicazione che dovrà interagire con la base di dati.
- alla sicurezza del database, poiché l'amministratore della base di dati può garantire il diritto di accedere alla stored procedure senza bisogno di concedere alcun privilegio sulle tabelle del database sottostante. Quindi le stored procedure si possono considerare come degli oggetti del database separati, con i loro privilegi.

Le stored procedure, pur essendo degli strumenti che offrono grandi potenzialità,

presentano dei difetti, che sono legati:

- al loro contenuto, in quanto una stored procedure contiene in prevalenza delle istruzioni di SQL dichiarative, che rendono di difficile scrittura una stored procedure con regole complesse, che al contrario sono più facilmente implementabili con altri linguaggi di programmazione, come per esempio Java e C. Esiste però la tendenza ad ampliare i linguaggi procedurali tipici di ogni DBMS con estensioni legate ai linguaggi di programmazione, in particolar modo a Java. Ciò avviene, per esempio, nel caso dei DBMS Oracle, in cui è possibile scrivere delle stored procedure estendendo il linguaggio PL/SQL con il linguaggio Java.
- alla mancata possibilità di effettuare un efficace debug all'interno del DBMS, cosa invece possibile con gli altri linguaggi di programmazione esterni al DBMS.
- al mantenimento, perché scrivere e mantenere efficiente nel tempo una stored procedure richiede delle conoscenze specifiche, che non tutti gli sviluppatori possiedono.

Per lo sviluppo di una stored procedure ci si avvale di un linguaggio procedurale, che varia da DBMS a DBMS.
L'American National Standards Institute (ANSI) nel 2003 ha definito un nuovo standard per il linguaggio SQL, denominato SQL:2003. Tra le funzionalità introdotte dal nuovo standard, vi rientra la definizione di un modulo per il linguaggio procedurale, denominata **SQL/PSM** (Persistent Stored Module).
Considerando che lo standard è nato nel 2003, solamente una parte dei DBMS sviluppati recentemente o che hanno introdotto la possibilità di sviluppare stored procedure negli ultimi anni, hanno implementato un linguaggio procedurale basato su SQL/PSM.
In pratica quindi ogni DBMS implementa un linguaggio procedurale differente

Triggers

Realizzano il paradigma (o Pattern) Evento-Condizione-Azione (ECA).

Trigger quindi come azione da svolgere a fronte di un evento, se è valida una condizione.

Gli eventi possono essere:

- insert
- update (record o campo)
- delete

1 Trigger è associato (sensibile) ad 1 solo Evento.

L'azione può essere l'invocazione di una Stored Procedure !

Al trigger sono associati due elementi speciali: OLD e NEW, i record prima e dopo la modifica richiesta (nessun OLD per l'Insert, nessun NEW per il Delete).

I trigger possono permettere l'implementazione di vincoli d'integrità più sofisticati di quelli visti in precedenza.

Si possono realizzare catene di trigger, dove l'azione di uno causa lo scatenarsi di un altro trigger, e cosi' via, fino a generare situazioni estremamente complesse e col rischio di cicli

infiniti (e conseguente blocco del sistema).

Un trigger si può definire come una parte di codice formata da istruzioni procedurali e dichiarative memorizzate nel catalog della base di dati, come avviene per le stored procedure, ma che non viene attivato tramite chiamata, bensì in seguito al verificarsi di un evento nella base di dati, quali ad esempio l'inserimento o la modifica di un record.
I trigger seguono il paradigma Evento-Condizione-Azione (ECA), secondo il quale il trigger viene attivato quando si verifica un determinato evento all'interno del database.
Se è soddisfatta una determinata condizione, allora il trigger esegue un'azione prestabilita. Un trigger è sensibile soltanto a un evento.
Solitamente l'obiettivo di un trigger è la determinazione delle reazioni che devono essere messe in atto al verificarsi di determinate eventi.
In relazione al paradigma ECA, l'evento può essere rappresentato da operazioni SQL di aggiornamento dello stato del database. Al contrario la condizione può essere un qualsiasi predicato SQL, e l'azione che deve essere svolta dal trigger è rappresentata da un comando SQL o dalla chiamata di una stored procedure.

Linguaggi di 4 generazione

Embedded SQL

Meccanismo

Inserire del codice SQL all'interno di un programma scritto in un linguaggio normale.

Il programma verrà PRECOMPILATO da un apposito sistema che traduce le istruzioni SQL in chiamate a librerie particolari che permettono l'interazione tra il programma stesso e il DBMS specifico.

Vi è quindi la necessità di definire delle sezioni e delle parole chiave che permettono al precompilatore di capire cosa deve fare.

Problemi:

- specificità rispetto al linguaggio, al DBMS, all'ambiente (S.O., networking, etc.)
- "conflitto d'impedenza": programmi ragionano per record, SQL per insiemi

Statico

Le istruzioni SQL sono definite una volta per tutte, con al limite dei valori passati come parametri dal programma alle istruzioni SQL.

E' una soluzione che può avere ovviamente dei problemi di flessibilità, ma è molto performante perché il precompilatore può effettuare molte ottimizzazioni, ed inoltre molti errori possono essere scoperti in fase appunto di precompilazione.

Dinamico

Le istruzioni SQL sono costruite a run-time, ad esempio creando la stringa con pezzi diversi a seconda delle esigenze del momento.

Il precompilatore non può effettuare tutti i controlli e le ottimizzazioni del caso statico.

Se il programma non lavora bene possono verificarsi improvvisi errori dovuti a particolari situazioni di esecuzione.

Cursori

I cursori sono una soluzione per risolvere il problema del conflitto d'impedenza: permettono di scorrere (appunto cursore) lungo il set di dati ottenuti da una query.

Un cursore è associato quindi ad una select, e può essere impostato per essere mono o bidirezionale, per permettere o no aggiornamenti dei dati letti.

Fasi di vita di un cursore:

- DECLARE
- OPEN
- FETCH
- CLOSE

Se il cursore è mono-direzionale permetterà di leggere un record alla volta dal primo all'ultimo, attraverso il comando NEXT.

Invece se il cursore è bidirezionale permette di tornare indietro ma anche di saltare tra i vari record:

NEXT, PRIOR, FIRST, LAST, ABSOLUTE, RELATIVA

Call Level Interface (CLI)

Meccanismo

Definizione di una libreria (INTERFACE) che permette di effettuare delle chiamate (CALL) al DBMS tramite comandi SQL.

Questa libreria può essere più o meno generica, astratta rispetto a:

DBMS

Sistema Operativo

Networking

Se viene definito uno Standard, allora più sviluppatori, più società possono fare dei prodotti che sono tra loro INTERCAMBIABILI, cioè hanno la stessa INTERFACCIA.

Se il linguaggio ha poi un meccanismo per collegare dinamicamente diverse librerie che presentano la stessa interfaccia a seconda di opportuni parametri di configurazione allora i programmi avranno una portabilità elevatissima.

Esempio: JDBC

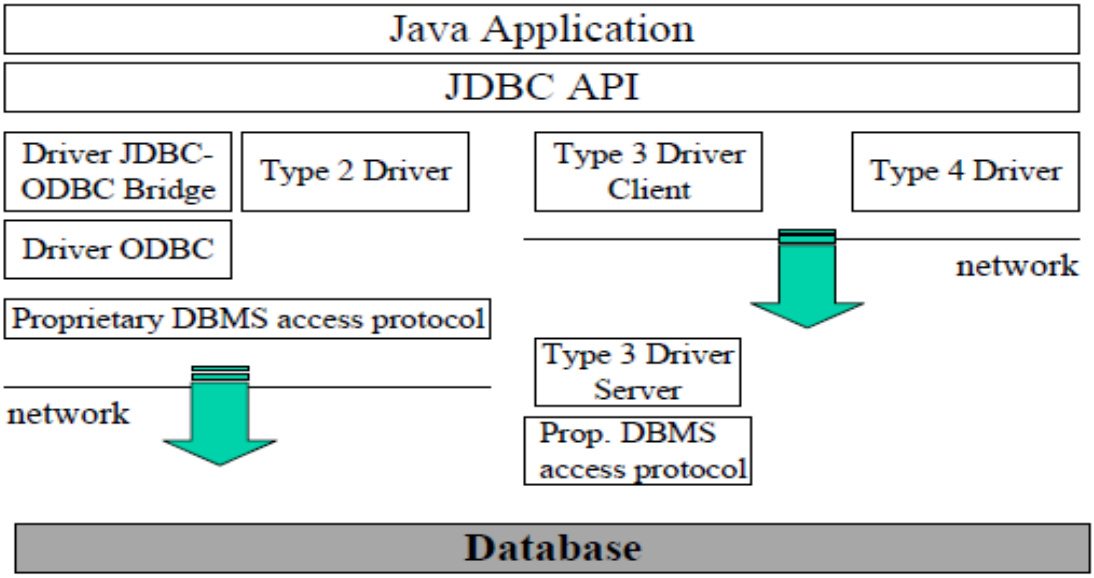
Elementi di un sistema che utilizza JDBC:

- applicazione Java
- JDBC Driver Manager
- Driver / Bridge / Middleware
- eventualmente Native Driver oppure ODBC
- DBMS

Struttura tipica di un programma che utilizza JDBC

- caricamento del driver (eventualmente sulla base di opportuni parametri di configurazione)
- apertura della connessione (con eventuali userId e password)
- creazione del contesto esecutivo (statement)
- esecuzione tramite lo statement delle istruzioni SQL (statiche o dinamiche)
- eventuale lettura dei dati attraverso oggetti di tipo ResultSet
- chiusura dello statement e della connessione

Architettura dei tipi di driver JDBC



I rischi della progettazione dei sistemi informatici

As proposed by the project sponsor

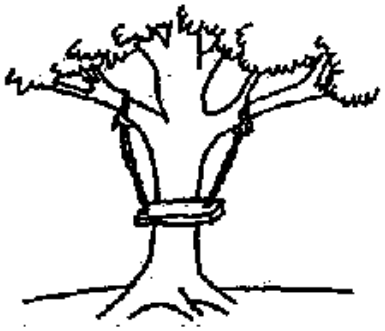
(proposta dal committente)

As specified in the project request

(specificata nei requisiti)

As designed by the senior analyst

(progettata dall'analista)

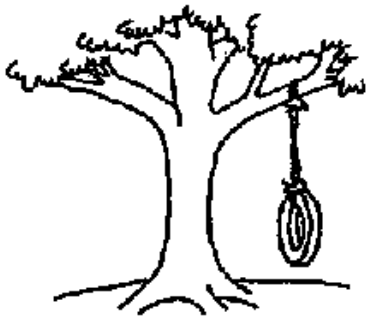


As produced by the programmers

(prodotta dai programmatori)

As installed at the user's site

(installata presso l'utente)



What the user wanted

(ciò che l'utente voleva)

Progettazione di Basi di Dati

Ciclo di vita di un sistema informativo

•**Idea**: identificazione di una necessità da soddisfare e del come soddisfarla

•**Studio di fattibilità**: verifica dell'effettiva realizzabilità dell'idea, valutazione costi e priorità
•**Raccolta e analisi dei requisiti**: raccolta dei desideri e delle necessità a cui va incontro il sistema e studio delle sue proprietà
•**Progettazione**: di dati e funzioni
•**Realizzazione**: codifica delle funzioni progettate e creazione dell'infrastruttura applicativa
•**Validazione e collaudo**: verifica sperimentale del funzionamento del sistema
•**Funzionamento**: vita operativa del sistema
•**Evoluzione**: trasformazione del sistema sulla base delle nuove esigenze (interne od esterne)

Centralità dei dati e necessità della loro modellazione in una fase precoce.

Il dato è più stabile delle funzioni (ma non immutabile)

Ciclo di analisi e progettazione dei dati

- **ANALISI** (CHE COSA)
  - raccolta dei requisiti
  - progettazione concettuale
    - schema concettuale
- **PROGETTAZIONE** (COME)
  - progettazione logica
    - schema logico
  - progettazione fisica
    - schema fisico

Analisi e Progettazione Concettuale

Raccolta dei requisiti

Dove si raccolgono i requisiti:

- dagli utenti del sistema
- nelle documentazioni relative al sistema
- da realizzazioni preesistenti (nello stesso contesto o in altri contesti)

Caratteristiche di una buona raccolta dei requisiti

- corretto livello di astrazione
- semplicità
- completamente esplicito (nulla di lasciato implicito)
- standardizzazione (dei termini e delle strutture)
- glossario

E' sempre e costantemente necessario verificare con l'utente la corretta comprensione da parte di entrambi e la consistenza e coerenza di tutto quanto descritto.

I requisiti non possono essere solo sui DATI, ma anche sulle OPERAZIONI da effettuare su tali dati, avendo quindi un approccio OLISTICO all'analisi.

Principio di indeterminatezza di Heisemberg (reciproco)

L'osservatore influenza il sistema osservato (ma anche il sistema osservato influenza l'osservatore).

Problema intrinseco della raccolta ed analisi dei requisiti: raccogliendo le informazioni se ne viene influenzati e quindi non si è più oggettivi, e interagendo con gli utenti non si è più neutrali.

Il rischio è conservare senza motivo caratteristiche dei sistemi e dei metodi esistenti senza metterli in discussione (e quindi compromettendo il nuovo sistema con eredità del vecchio), o essere prevenuti nei confronti di parti o utenti del sistema sulla base di quanto dettoci in precedenza.

Documentazione di progetto

Il modello E-R rappresenta in modo formale i concetti ma non li descrive, e non è adatto per la rappresentazione di vincoli complessi.

E' necessario integrare il modello E-R con una descrizione (formale o no) delle rimanenti parti del sistema.

Business Rule

Le business rule relative all'analisi dei dati sono di tre tipi:

- descrizione di concetti – in forma testuale libera
- dichiarazione di vincoli d'integrità
  - asserzioni del tipo: <ente del modello> **deve / non deve** <proprietà o espressione>

- derivazione di una proprietà da altre esistenti

  – operazioni aritmetiche o logiche per dedurre proprietà o enti da altre informazioni presenti nel modello, del tipo: <risultato> **si ottiene** <espressione o formula>

Vi sono poi altre business rule che meglio si adattano alla modellazione dei processi aziendali, descrivendo appunto le regole aziendali che governano la trasformazione dei dati, ed hanno la forma:

se <condizione iniziale> **allora** <descrizione del risultato di un processo di trasformazione>

Scenari d'uso

Uno strumento molto importante per l'analisi dei sistemi è la raccolta e la descrizione di scenari d'uso, cioè di precise sequenze di azioni ed eventi (come la sceneggiatura di un film) relativi a processi reali o da realizzare.

Sono utili soprattutto per la descrizione della parte dinamica del sistema, ma poiché i dati dovranno essere utilizzati all'interno di questo sistema dinamico, per ben progettare i dati devono essere noti anche i processi.

Permettono di far vedere (come in un film) all'utente come sarà il sistema, ma anche di formalizzare la descrizione dell'esistente per una più chiara comprensione da parte di progettisti ed analisti.

Modellazione

Schemi e Istanze

schema: sostanzialmente invariante nel tempo, descrive la struttura (aspetto intensionale)
- •nel modello relazionale, le intestazioni delle tabelle

istanza: i valori attuali, che cambiano anche molto rapidamente (aspetto estensionale)
- •nel modello relazionale, il "corpo" di ciascuna tabella

Modelli

modelli concettuali: permettono di rappresentare i dati in modo indipendente dal sistema
- •cercano di descrivere i concetti del mondo reale
- •sono utilizzati nelle fasi preliminari di progettazione

modelli logici: utilizzati per l'organizzazione dei dati da parte dei DBMS
- •utilizzati dalle applicazioni
- •indipendenti dalle strutture fisiche

Motivi per l'uso dei modelli concettuali

Se partiamo a progettare direttamente dallo schema logico della base di dati di un'applicazione:
•da dove cominciamo?
•rischiamo di perderci subito nei dettagli
•dobbiamo pensare subito a come correlare le varie tabelle (chiavi etc.)
•ci sono grosse rigidi tà

Il modello concettuale ha:
- una rappresentazione grafica (più comunicativa)
- maggiore astrazione rispetto ai dettagli implementativi
- quindi più semplice
- quindi più comprensibile per l'utente/committente del sistema

La tecnica di modellazione concettuale più diffusa è quella "**Entity-Relationship**" (E-R)