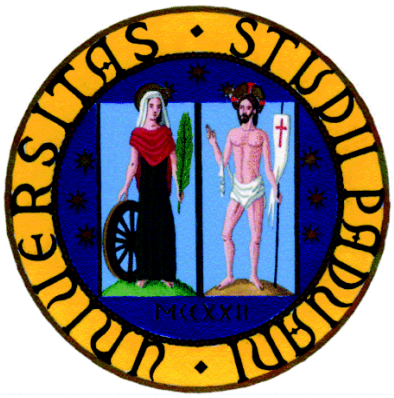
UNIVERSITA' DEGLI STUDI DI PADOVA

FACOLTA' DI SCIENZE STATISTICHE



LAUREA IN SCIENZE STATISTICHE E TECNOLOGIE INFORMATICHE

Implementazione sul database "VisionSQL" di una procedura per il log delle operazioni effettuate: stage presso la "Vision Software Gestionale"

Implementation of a procedure for the log of the executed operations in "VisionSQL" database: stage at "Vision Software Gestionale"

Relatore: Prof. MASSIMO MELUCCI

Laureando: IRENE SGUOTTI

Matricola: 573443-STI

ANNO ACCADEMICO 2009/2010

INDICE ANALITICO

OPERATION LOG

ATTIVITA' di TEST

1. Introduzione

Questa tesi nasce da un'esperienza di stage fatta nel reparto di programmazione di un'azienda produttrice di software gestionali. Ho deciso di fare questa esperienza perché dopo molti anni di studio volevo conoscere il mondo del lavoro e soprattutto capire come tutto ciò che ho imparato nei libri e ciò che mi è stato insegnato all'università viene applicato nella realtà lavorativa. Tantissime persone, anche gli stessi professori, mi ripetevano sempre che il lavoro è molto più complesso degli esempi e delle prove svolte all'università e durante il mio percorso di studi. La facoltà offre la possibilità di svolgere uno stage, così ho deciso di approfittare di questa occasione e includere nella mia carriera universitaria un approccio diretto con il mondo lavorativo.

La relazione è suddivisa in due parti. La prima parte è formata da una parte introduttiva nella quale presento l'azienda e i prodotti che essa offre ed una parte nella quale spiego il progetto e il lavoro che ho svolto in azienda. La seconda parte invece è caratterizzata da un'attività di test ed analisi statistiche per verificare l'efficienza dell'applicazione dopo le modifiche da me apportate e termina con le conclusioni sull'attività e sull'esperienza svolta.

Anticipo fin dall'inizio che la mia esperienza di stage è stata molto positiva perché mi è piaciuto molto sia il progetto che ho svolto sia l'ambiente in cui ho lavorato. Credo inoltre che questo stage sia stato ottimo per vedere applicati nella pratica tantissimi concetti teorici riguardanti non solo le basi di dati ma anche la statistica stessa.

2. L’azienda e gli obiettivi dello stage

2.1 L’azienda

L’azienda presso la quale ho svolto lo stage è la “Vision Software Gestionale”, situata a Pernumia, in provincia di Padova.
E’ un’azienda nuova, ma con una storia che parte dal 1981, che si occupa di informatica e più precisamente di quella parte dell’informatica dedicata alla gestione aziendale. Inizialmente l’attività era dedicata ad aziende, enti pubblici, studi professionali e centri di elaborazione dati, e prevedeva la gestione di tutte le problematiche informatiche quali la progettazione dei sistemi, l’hardware, le reti, i sistemi operativi e il software applicativo. Successivamente, ritenuta indispensabile la specializzazione, l’attività venne dedicata in particolare modo alle aziende. La Vision Software quindi ora offre software applicativi in grado di supportare le piccole e medie imprese nella gestione delle complesse dinamiche aziendali, offrendo strumenti e servizi che consentono di integrare i vari processi produttivi, i cicli economici e finanziari in modo semplice, stabile e veloce.
I prodotti che “Vision” offre sono soluzioni applicative in ambiente Windows per la gestione contabile, finanziaria, commerciale, di magazzino o di produzione per l’industria e l’azienda di produzione; soluzioni rivolte ad aziende che intendono importare sul sistema informatico non solo la semplice gestione amministrativa o di magazzino ma tutta una serie di operazioni più o meno legate all’attività aziendale che di norma non vengono implementate dai classici gestionali.
Il software gestionale su cui l’azienda investe è chiamato “Vision” e nasce come prodotto internazionale grazie alla funzionalità “MYLAB Make Your Language By Yourself”, che consente di creare con la massima facilità l’intera interfaccia operativa del prodotto nella lingua desiderata.
Su Vision “standard” sono stati poi costruiti una serie di programmi specifici per diverse attività tra cui:

- VISION ENERGY : è il software per la gestione contabile, finanziaria, commerciale e di magazzino dedicato alle aziende che si occupano di distribuzione all'ingrosso di prodotti petroliferi e gas combustibile.

- VISION FRESH : è un software gestionale per il settore ortofrutticolo e ittico. Si occupa di tutti gli aspetti gestionali di queste aziende ed e' rivolto in particolare a Cooperative di produzione e distribuzione, Commissionari di Mercati Ortofrutticoli, Grossisti.

- VISION TRASPORTI : è il software gestionale per autotrasportatori e consente di gestire con maggiore precisione e redditività i viaggi e i rapporti con gli autisti e con i clienti e al tempo stesso consente un buon controllo degli automezzi e dei costi di gestione.

- VISION PREF : è un insieme integrato di procedure software per gestire al meglio l'azienda di produzione di prefabbricati, dall'offerta alla determinazione dei costi, alla gestione del magazzino materie prime e prodotti finiti, dalla contabilità industriale alla gestione delle normative antinfortunistiche.

- VISION ERP : è il software gestionale dedicato alle piccole e medie aziende industriali, commerciali e dei servizi, che vogliono mantenere e accrescere la loro competitività in un mercato globale sempre piu' dinamico.

- VISION SQL : è il software gestionale dedicato alle medie e piccole aziende industriali, commerciali e dei servizi, che gestiscono notevoli quantità di dati e hanno la necessità di lavorare con grande velocità e stabilità. Aziende che grazie ad analisi dei dati in tempo reale vogliono prendere decisioni importanti in tempi rapidi, aziende in crescita che affrontano con grande competitività le sfide del mercato, sicure di essere supportate da un software gestionale all'avanguardia.

2.2 Gli obiettivi dello stage

La definizione del progetto di stage è stata un po' difficoltosa in quanto inizialmente il progetto concordato con il tutor aziendale doveva riguardare la creazione di trigger per garantire l'integrità dei dati nel database; poi però tale progetto è stato modificato in quanto era troppo simile a quello di un altro collega/stageista frequentante la facoltà di Scienze Statistiche. Si era quindi deciso successivamente di assegnarmi il compito di studiare la struttura del database in relazione alla creazione di nuovi indici, utili per migliorare le prestazioni della base di dati. Questo progetto però risultava di difficile applicazione perché prevedeva una sorta di collaborazione con alcune aziende clienti per capire quali fossero le operazioni compiute più frequentemente. Data l'impossibilità di realizzare questa collaborazione, il tutor aziendale mi ha proposto di lavorare ad un altro progetto che aveva intenzione di realizzare da tempo, cioè la creazione di una nuova tabella da inserire nel database che contenesse una traccia di tutte le operazioni di inserimento, cancellazione e modifica effettuate tramite l'applicativo gestionale "VisionSQL".
In realtà il DBMS utilizzato (SQL Server) gestisce un proprio LOG interno delle transazioni ma non esiste uno strumento che consenta di consultare lo storico di tutte le operazioni effettuate. Mi è stato quindi chiesto di implementare una nuova funzionalità che consentisse ad una certa categoria di utenti di poter analizzare lo storico delle attività svolte sui record del database. La consultazione e la gestione dell'elenco delle operazioni effettuate, infatti, deve essere limitata solo alla categoria degli utenti amministratori. Una funzionalità di questo tipo viene già venduta da terze parti come tool aggiuntivo, ma il suo costo può essere anche di alcune migliaia di euro. Io quindi ho dovuto implementare la funzionalità direttamente all'interno del database del gestionale. L'utilità principale del lavoro fatto è stata quindi quella di agevolare chi si occupa della manutenzione del database che in questo modo può avere a disposizione uno strumento che gli permetta di sapere tutte le operazioni che sono state effettuate nel database e anche chi le ha eseguite. In futuro la nuova funzionalità da me aggiunta potrà essere ulteriormente arricchita aggiungendo la possibilità di effettuare delle statistiche sull'utilizzo del

gestionale riguardanti, ad esempio, il tipo di operazioni eseguite più frequentemente, il numero medio di documenti inseriti quotidianamente, il numero medio di accessi giornalieri al database ed altre.
Le attività che ho svolto durante lo stage per la realizzazione di questa tabella contente lo storico di tutte le operazioni di inserimento, modifica e cancellazione effettuate possono essere suddivise in tre fasi.
Durante la prima fase ho dovuto come prima cosa esaminare la struttura del database, ossia analizzare tutte le tabelle per capire cosa contenessero e per capire quali relazioni avessero tra di loro. Successivamente ho dovuto documentarmi per imparare ad utilizzare il linguaggio T-SQL, estensione proprietaria del linguaggio SQL utilizzata da Microsoft, e per capire le modalità di funzionamento di trigger e stored procedure. Dopo questa fase di studio ho capito che per inserire i dati nella tabella avrei dovuto utilizzare una stored procedure, ossia un insieme di istruzioni in linguaggio T-Sql che possono essere eseguite migliaia di volte, senza doverle sempre riscrivere. Inizialmente però, per prendere confidenza con l'utilizzo delle stored procedure, ne ho creata una semplice, chiamata "SPOperation", che ricevesse come parametri di input il nome della tabella e l'iniziale dell'operazione effettuata (I/U/D) e che emettesse a video un semplice messaggio contenente il tipo di operazione (inserimento, modifica o cancellazione) e il nome della tabella nella quale l'operazione fosse stata effettuata. Successivamente ho richiamato in alcuni trigger esistenti la prima bozza della stored procedure per verificare il funzionamento dei messaggi all'esecuzione di un'operazione.
Una volta completata la prima fase sono passata alla seconda, nella quale ho dovuto creare la nuova tabella "LOGDB" e di conseguenza modificare la stored procedure in modo che inserisse i dati nella tabella. I campi che ho deciso di inserire in questa nuova tabella sono i seguenti:

- il codice dell'utente che compie l'operazione
- la data in cui è stata effettuata l'operazione
- il nome della tabella
- il tipo di operazione: inserimento, modifica o cancellazione

- la descrizione del campo modificato o il codice a cui è riferito un inserimento o una cancellazione
- la chiave primaria, formata da uno o più attributi, della tabella

Terminata la creazione, ho modificato la stored procedure per poter inserire, ad ogni operazione, i dati nella tabella. Per ricavare l'utente che ha effettuato una certa operazione e il valore della chiave primaria appena inserita o cancellata ho dovuto aggiungere nella stored procedure due nuovi parametri di input: la sessione dell'utente e il codice. Nel corpo poi ho inserito anche le istruzioni per ricavare la data e l'insieme di attributi che formano la chiave primaria della tabella. Per la creazione del campo modificato invece ho dovuto creare una funzione in Microsoft Visual FoxPro. Infine ho esteso la chiamata della stored procedure a tutti i trigger.
Conclusa anche questa fase, prima di passare alla terza ed ultima fase sono riuscita a sviluppare anche una fase facoltativa che mi è stata proposta. Questa è consistita (I) nel criptare la tabella in modo da renderla consultabile solo agli amministratori e (II) nel creare l'interfaccia di consultazione in "VisionSQL" della tabella "LOGDB".
Infine, la terza fase ha previsto un'attività di test della nuova funzionalità. Ho dovuto quindi confrontare i tempi di risposta delle query di cancellazione, inserimento e modifica in presenza e in assenza della chiamata alla stored procedure all'interno dei trigger. Tutta l'attività di test è stata documentata con verbali compilati secondo i modelli in uso in azienda. Lo stage si è concluso con la presentazione delle attività svolte al reparto di programmazione al fine di riportare nell'ambiente di produzione quanto realizzato.
Conclusa questa breve introduzione delle attività che ho svolto durante il periodo di stage, procedo con una descrizione più approfondita, sia teorica che pratica, degli argomenti, in modo da avere alla fine un'idea più concreta dell'utilità del lavoro che ho svolto.

3. Analisi del database

3.1 Richiami concettuali

Prima di procedere con l'analisi specifica del database utilizzato, mi sembra opportuno richiamare alcuni concetti di base riguardanti le basi di dati e i DBMS, il modello relazionale, i vincoli d'integrità e le chiavi.

- **Le basi di dati e il DBMS**

Un sistema di gestione di basi di dati (in inglese Data Base Management System, abbreviato con DBMS) è un sistema software in grado di gestire collezioni di dati che siano grandi, condivise e persistenti, assicurando la loro affidabilità e privatezza.
Una base di dati è una collezione di dati gestita da un DBMS.
Vediamo con chiarezza le caratteristiche dei DBMS e delle basi di dati citate nelle definizioni:

- Le basi di dati possono essere *grandi*, nel senso che possono avere anche dimensioni maggiori della memoria centrale disponibile.
- Le basi di dati sono *condivise*, nel senso che applicazioni e utenti diversi possono accedere, secondo determinate modalità, a dati comuni. Ciò riduce anche la ridondanza dei dati, ossia il ripetersi di dati uguali.
- Le basi di dati sono *persistenti*, ossia hanno un tempo di vita che non è limitato a quello delle singole esecuzioni dei programmi che le utilizzano. I dati in memoria centrale, invece, hanno una vita che inizia e termina con l'esecuzione del programma che li gestisce.
- I DBMS garantiscono l'*affidabilità*, ossia la capacità del sistema di conservare intatto il contenuto delle basi di dati in caso di malfunzionamenti hardware e software.
- I DBMS garantiscono la *privatezza* dei dati. Ogni utente, riconosciuto in base ad un nome d'identificazione specificato, può svolgere solo

determinate operazioni sui dati, a seconda delle autorizzazioni che gli sono state concesse.

- **Il modello relazionale**

Il modello relazionale è un modello di rappresentazione dei dati implementato nei DBMS, basato sull'algebra relazionale e la teoria degli insieme ed è strutturato attorno alla definizione di relazione.
Dati $n > 0$ insiemi, $D_1, D_2,\ldots,D_n$, si chiama prodotto cartesiano di $D_1, D_2,\ldots,D_n$, e si indica con $D_1 \times D_2 \times \ldots \times D_n$, detti domini, l'insieme delle n-uple $(v_1, v_2, \ldots, v_n)$ tali che v_i appartiene all'insieme D_i, per $1 \leq i \leq n$. Una relazione è un sottoinsieme del prodotto cartesiano $D_1 \times D_2 \times \ldots \times D_n$.
La tabella è la rappresentazione grafica normalmente accettata per rappresentare le relazioni.
Il grado della relazione è il numero n delle componenti del prodotto cartesiano mentre la cardinalità della relazione è il numero di elementi della relazione.
L'attributo è il termine utilizzato per indicare il nome di una colonna di una tabella, quindi è il nome dato ad un dominio della tabella.
Spiegando in modo informale, e come nella pratica viene spesso utilizzato, possiamo utilizzare le seguenti definizioni: una relazione è una tabella, la cardinalità è il numero di righe, il grado è il numero di colonne, un attributo è una colonna ed una tupla è una riga.

- **Vincoli d'integrità e chiavi**

Le strutture del modello relazionale permettono di organizzare le informazioni per le applicazioni in maniera semplice ed efficiente. È necessario però utilizzare dei meccanismi che assicurino la correttezza delle informazioni, in modo che non possano mai esistere delle istanze che non rappresentano correttamente il mondo applicativo. Il vincolo d'integrità può essere visto come una proprietà che deve essere rispettata affinché i dati possano essere considerati corretti per l'applicazione.

I principali vincoli d'integrità sono quelli che specificano:

- quali attributi possono assumere valore nullo,
- quali attributi sono chiavi,
- quali attributi sono chiavi esterne.

Valori nulli

I valori nulli sono dei valori speciali che indicano la mancanza di informazione. Vengono quindi utilizzati quando non è possibile specificare il valore di un attributo e non è specificato nessun valore di default.

Chiavi

I vincoli di chiave sono i più importanti del modello relazionale. Essi garantiscono l'univocità delle tuple di una relazione.

Si possono specificare tre tipi di chiavi:

- una superchiave identifica univocamente ogni ennupla. Essa infatti è un sottoinsieme di attributi di una relazione tale che in nessuna istanza della relazione possano esistere due ennuple diverse che coincidono su tutti gli attributi.
- una chiave è una superchiave minimale, ossia tale che eliminando un attributo non si ha più una superchiave.
- una chiave primaria è una delle chiavi, solitamente quella con meno attributi.

Le chiavi primarie, a differenza delle chiavi, non possono assumere valori nulli perché altrimenti non sarebbe garantita l'univocità delle tuple della relazione.

In molti casi reali però è possibile che all'interno di una tabella nessun sottoinsieme di attributi identifichi univocamente ogni riga. È necessario quindi, aggiungere alla relazione un attributo, un codice, che, pur non avendo significato per l'applicazione che si vuole creare, garantisca l'univocità di ogni riga.

Chiavi esterne

Le chiave esterne sono necessarie per rappresentare le associazioni nel modello relazionale.

Quando due tabelle vengono associate, la chiave primaria di una tabella viene inserita nell'altra come chiave esterna e il vincolo che ne scaturisce sta nel fatto che l'insieme dei valori che la chiave esterna può assumere è l'insieme dei valori assunti dalla chiave primaria a cui è riferita. Il vincolo di chiave esterna viene definito anche vincolo di integrità referenziale o foreign key.

3.2 Microsoft SQL Server e SQL Management Studio 2005

Microsoft SQL Server è un DBMS relazionale prodotto da Microsoft. Nelle prime versioni era utilizzato per basi di dati medio-piccole, ma a partire dalla versione 2000 è stato utilizzato anche per la gestione di basi dati di grandi dimensioni. L'ultima versione rilasciata è "SQL Server 2008" ma durante lo stage io ho utilizzato la versione SQL Server 2005.

In realtà esistono sei versioni di SQL Server 2005, cinque delle quali a pagamento e una gratuita. La versione gratuita, SQL Server 2005 Express, a differenza delle versioni a pagamento ha delle restrizioni: può utilizzare una sola CPU, 1 GigaByte di RAM e supporta database con dimensioni massime fino a 4 GigaByte. Durante tutto lo svolgimento del mio stage io ho utilizzato questa versione.

SQL Server Express include una console di amministrazione chiamata "SQL Server Management Studio Express". Questa permette di gestire ogni fase della creazione e manutenzione di una base di dati, assistendo l'utente nella creazione di tabelle, diagrammi, trigger, stored procedure e tutto quello che può essere utile per l'amministrazione del database. Permette inoltre di effettuare backup e restore dei database, per garantire una maggior sicurezza in caso di malfunzionamenti o guasti. Uno strumento molto importante di SQL Server Management Studio Express è il piano di esecuzione. Tramite questo strumento è possibile valutare le prestazioni della nostra richiesta al server e capire quindi se la query che abbiamo impostato è performante o meno. Creata

la query, premendo il pulsante piano di esecuzione, questa viene eseguita mostrando le statistiche di esecuzione. I risultati vengono divisi per le singole operazioni che il motore di database ha dovuto compiere e per ognuna di esse viene indicata la percentuale di utilizzo da parte del sistema, ovvero come ha inciso nell'esecuzione della query.

3.3 Creazione dei diagrammi

Il database di “VisionSQL” è composto da quasi duecento tabelle, tra le quali alcune contengono più di sessanta campi. Risulta quindi difficile per chi non ha progettato e realizzato personalmente la base di dati, ricordarsi a memoria tutte le tabelle e le relazioni esistenti tra loro. Durante la prima parte dello stage mi è stato chiesto di costruire, assieme ad un collega stagista, qualche diagramma. Ciò mi è stato utile per analizzare meglio il database ed evitare di commettere degli errori. I diagrammi infatti sono una rappresentazione grafica delle relazioni tra le varie tabelle. Durante il mio percorso di studi, mi è sempre stato insegnato di costruire come prima cosa il diagramma del database, e successivamente di progettarlo. In questo caso, però, ho dovuto fare la cosa inversa, ossia partire dal database già realizzato e creare i diagrammi. A causa della numerosità delle tabelle e delle relazioni la creazione di un diagramma “totale” sarebbe stata difficoltosa e lo stesso diagramma sarebbe risultato di difficile interpretazione. Sono risultati quindi molto più utili dei diagrammi che rappresentassero porzioni di database. I diagrammi che ho creato quindi rappresentano solo delle parti di database, e in particolare mi è stato chiesto di creare solo i diagrammi per le tabelle più utilizzate. In tutti i diagrammi fatti, per capire meglio le relazioni, la tabella interessata si trova al centro ed attorno ci sono tutte le tabelle ad essa collegate.
Per la creazione dei diagrammi ho utilizzato “SQL Server Management Studio 2005” e per cercare tutte le tabelle che fossero tra loro collegate ho utilizzato una query creata anche grazie all’aiuto del team di programmazione.

Descrivo di seguito le procedure eseguite per la creazione di un diagramma con l'utilizzo di "SQL Server Management Studio 2005", riportando anche la query utilizzata per ricercare le chiavi esterne.
All'apertura di Management Studio appare una schermata attraverso la quale è possibile connettersi al server attraverso l'autenticazione di Windows o autenticandosi con le credenziali di utente presente in SQL Server, come mostrato in figura 3.1:

Figura 3.1: Connessione al server

Dopo aver effettuato la connessione al server, sulla sinistra vengono visualizzati tutti i database presenti e per ognuno di essi è possibile visualizzarne i diagrammi, le tabelle e tutte le altre funzionalità utili per la gestione del database. Tra queste, sotto la voce Programmabilità, sono presenti le stored procedure. Espandendo invece il menu di una tabella compaiono una serie di altri menu tramite i quali è possibile visualizzare le colonne, le chiavi, i trigger e gli altri elementi associati alla tabella. Per creare un nuovo oggetto, sia esso colonna, trigger o stored procedure, sarà sufficiente cliccare con il tasto destro del mouse sulla voce corrispondente e selezionare Nuovo. Se invece si espande il menu dei diagrammi è possibile visualizzare i diagrammi già creati, oppure crearne di nuovi. Il tutto è mostrato in figura 3.2:

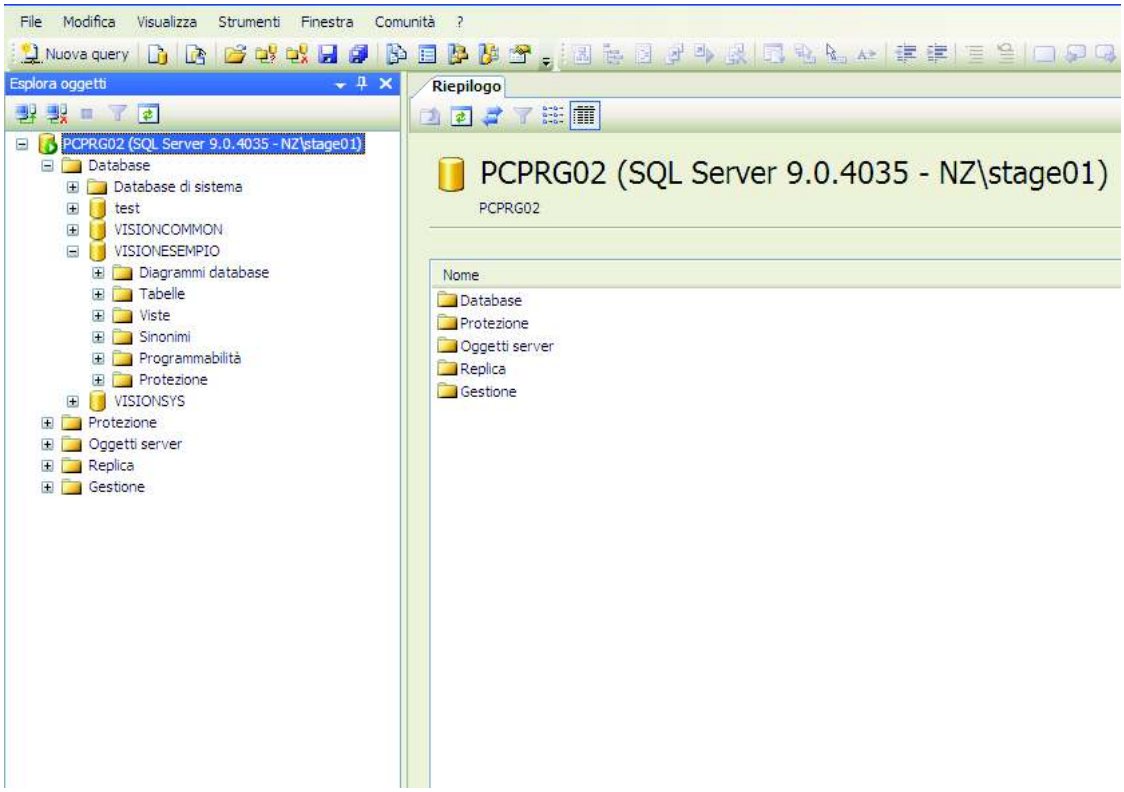


Figura 3.2: Schermata iniziale di Management Studio

Al momento della creazione di un nuovo diagramma apparirà sulla parte destra dello schermo una nuova finestra editor nella quale si presenterà automaticamente una finestra contenente l'elenco di tutte le tabelle. Da questo elenco è possibile selezionare tutte le tabelle che devono essere rappresentate; poi si può passare alla creazione di tutte le relazioni. Per semplificare la lettura dei diagrammi, come detto in precedenza, si è scelto di mettere al centro la tabella di interesse e attorno ad essa tutte le tabelle da collegare. Le tabelle attorno sono quindi quelle che contengono la chiave primaria della tabella al centro del diagramma e quelle la cui chiave primaria è presente nella tabella che stiamo rappresentando. Come menzionato in precedenza, per trovare tutte le tabelle contenenti la chiave primaria della tabella di interesse ho utilizzato una query che controlla tutte le colonne di tutte le tabelle del database. La riporto qui di seguito, visto il grande utilizzo che ne ho fatto.

```
SELECT table_name, column_name
FROM information_schema.columns
WHERE data_type = 'varchar' and column_name like '%codCespite%'
and left(table_name,2) <> 'P_'
ORDER BY table_name, column_name
```

Per creare una relazione è sufficiente selezionare la chiave primaria di una tabella e trascinarla nel corrispondente campo della tabella che si vuole collegare. Si aprirà così una finestra nella quale inserire il nome della relazione e controllare che la relazione sia stata creata correttamente, come mostrato in figura 3.3:

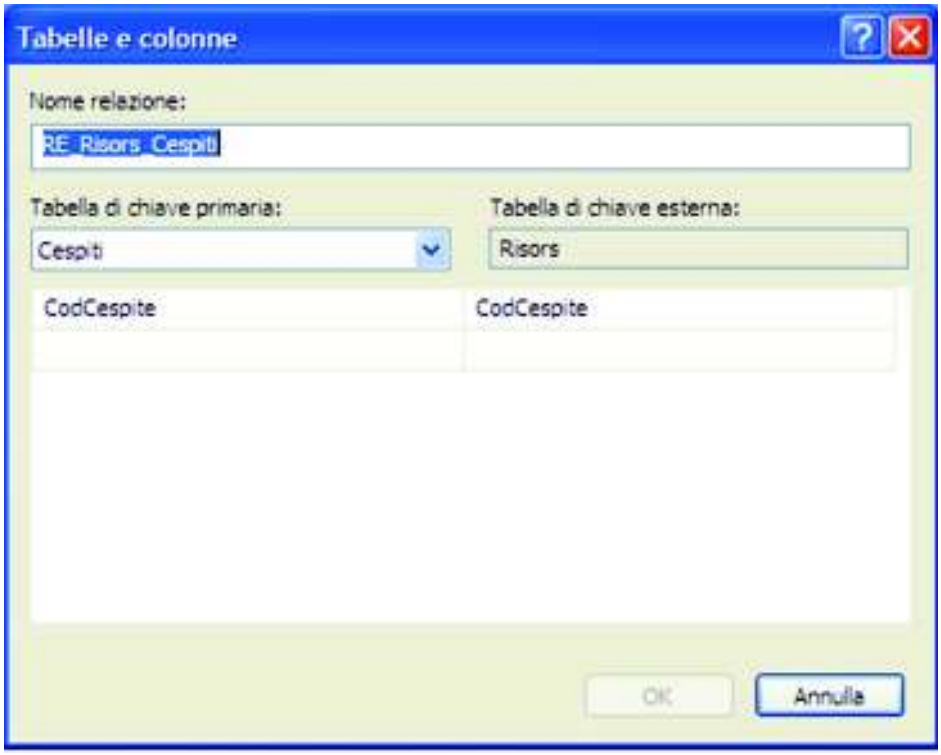


Figura 3.3: Nome della relazione

Dall'immagine si nota che la tabella Cespiti è collegata alla tabella Risors attraverso la chiave primaria dei Cespiti, CodCespite. Ciò significa che la relazione tra Cespiti e Risors è di tipo uno a molti e quindi una risorsa può essere collegata ad un solo cespite, mentre un cespite può essere collegato a più risorse. Un'altra cosa importante da notare è il nome delle relazione:

RE_Risors_Cespiti. La relazione è stata chiamata così perché tra le due tabelle non esiste un vincolo di chiave esterna, ma esiste una relazione che sarà poi gestita con un trigger. Tutto ciò è deciso nella schermata successiva, figura 3.4:

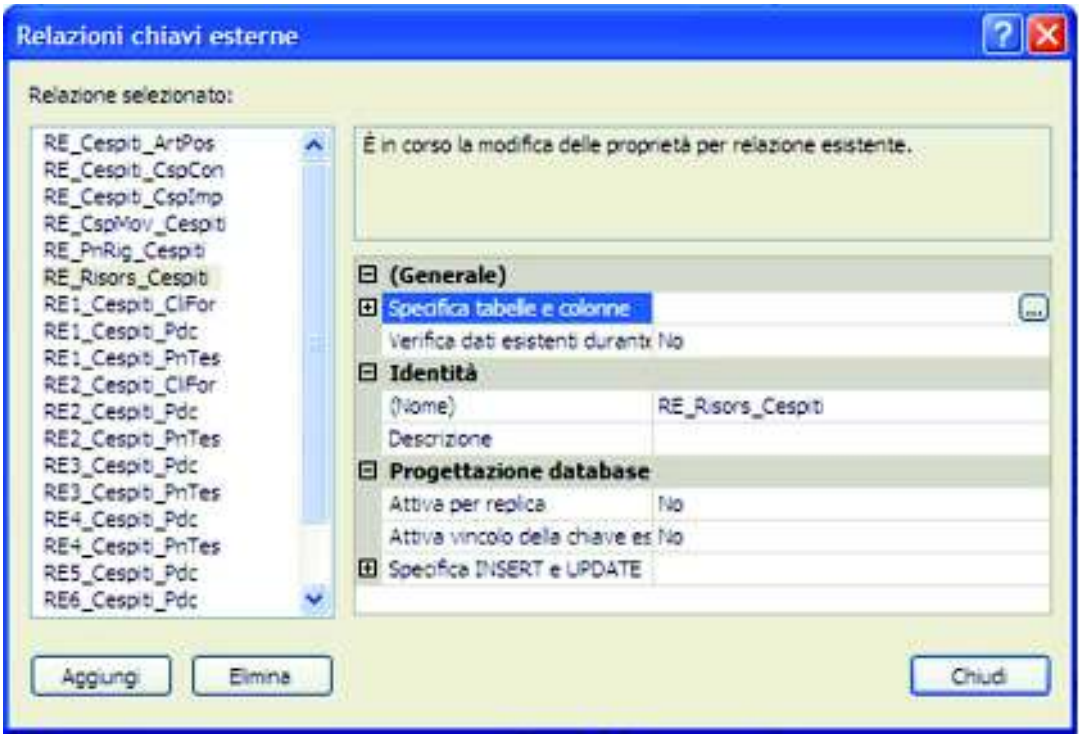


Figura 3.4: Relazioni chiavi esterne

Come vediamo da questa figura infatti la voce “Attiva il vincolo della chiave esterna” non è attiva. Se invece fosse attiva, la relazione si sarebbe chiamata “FK_Risors_Cespiti”. Il tipo di relazione, se di chiave esterna o meno, si vede anche graficamente; infatti le relazioni di chiave esterna sono disegnate con una linea continua mentre le semplici relazioni senza vincoli sono rappresentate con una linea tratteggiata. In questa schermata si possono anche specificare alcune proprietà della relazione, come ad esempio le operazioni da effettuare nel caso vengano eseguiti inserimenti, cancellazioni o modifiche. Queste operazioni possono essere specificate solo quando il vincolo di chiave esterna è attivo. Ad esempio nel nostro caso, se la relazione tra Cespiti e Risors fosse una relazione con il vincolo di chiave esterna, potremmo specificare che nel momento in cui un cespite venisse cancellato, fosse eseguita una cancellazione a cascata, ossia venisse cancellata anche la

risorsa ad esso collegata. Con lo stesso procedimento vengono create tutte le altre relazioni fino ad ottenere lo schema finale, come mostrato in figura 3.5:

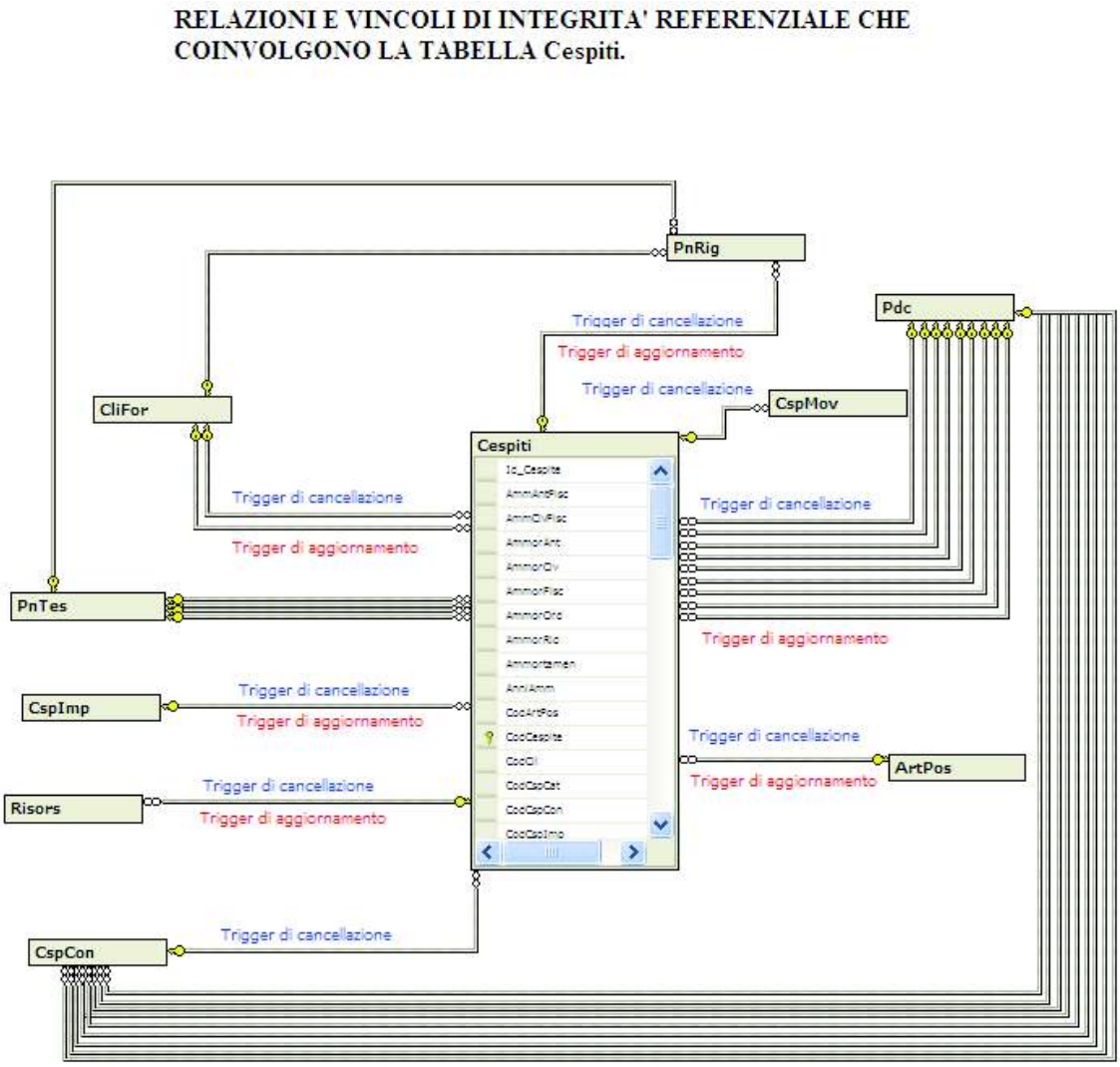


Figura 3.5: Digramma della tabella Cespiti

In questo diagramma, come in tutti i diagrammi da me creati, la tabella della quale vogliamo conoscere tutte le relazioni è situata al centro con attorno tutte le tabelle ad essa collegate.

OPERATION LOG

Operation Log è il nome del progetto che ho portato a termine; “LOGDB” è la tabella che ho dovuto creare e che contiene lo storico di tutte le operazioni di inserimento, modifica e cancellazione. La realizzazione del progetto è avvenuta per gradi. Inizialmente, infatti, ho realizzato una stored procedure che emettesse a video dei messaggi contenenti il tipo di operazione effettuata e in quale tabella. Successivamente ho creato la tabella e modificato la stored procedure in modo da andare a scrivere i dati in essa. Infine ho ottimizzato il lavoro criptando la tabella e creando l’interfaccia di consultazione in “VisionSQL”.

4. Prima bozza di “SPOperation”

4.1 Le Stored Procedure

Le stored procedure sono presenti in Microsoft SQL Server fin dalle sue prime versioni e rappresentano una parte molto importante e utile della programmazione Transact-SQL, estensione proprietaria del linguaggio SQL sviluppata da Microsoft. Esse sono un insieme di istruzioni T-SQL che vengono memorizzate sul server con un nome che le identifica e accettano in input dei parametri. Una stored procedure può avere una dimensione massima di 128 MegaByte e il numero massimo di parametri che può ricevere in input è 2100.
La loro maggiore utilità sta nel fatto che possono essere richiamate in qualsiasi momento durante l’esecuzione di altre istruzioni. E’ quindi possibile inserire al loro interno query molto complesse ed evitare così di doverle riscrivere ogniqualvolta siano necessarie, evitando così di commettere errori. Una stored procedure viene in genere costruita per eseguire una determinata funzione, nel senso che la sequenza di istruzioni che essa racchiude al suo interno serve per portare a termine un preciso obiettivo. Essa, infatti quando viene eseguita, riceve in input dei parametri e al termine restituisce gli stessi valori che si sarebbero ottenuti eseguendo semplicemente la sequenza di istruzioni. Proprio

per questo l'utilizzo delle stored procedure permette di avere meno righe di codice rispetto a quelle che si avrebbero non utilizzandole, rendendo così più interpretabile e scorrevole la lettura del codice.
Un altro motivo per cui è preferibile l'uso delle stored procedure è che consentono di ridurre il traffico in rete. Una stored procedure può infatti contenere al suo interno centinaia di righe di codice ma per la sua esecuzione è sufficiente una singola istruzione. Questo riduce notevolmente il traffico in rete perché, grazie ad esse, può essere eseguita solo l'istruzione necessaria per richiamarla, anziché tutte le righe di codice che contiene.
Riepilogando, quindi, i vantaggi provenienti dall'utilizzo delle stored procedure in Microsoft SQL Server sono molteplici, infatti:

- sono memorizzate direttamente sul server e ciò aumenta di conseguenza le prestazioni dell'applicazione. Esse riducono inoltre il traffico in rete, infatti una stored procedure viene compilata un'unica volta al momento dell'inserimento e successivamente per utilizzarla serve una singola istruzione di esecuzione, evitando così lo svolgimento di centinaia di righe di codice che saranno racchiuse nel suo corpo.
- possono essere eseguite anche migliaia di volte in un certo programma. Permettono quindi di non dover riscrivere delle query molto complesse ogni volta che il programma ne richiede l'utilizzo. Questo aiuta anche a migliorare l'interpretabilità e la manutenzione dell'applicazione.

4.2 Tipi di stored procedure

In Microsoft SQL Server 2005 esistono vari tipi di stored procedure: stored definite dall'utente, stored estese e stored di sistema.

- Stored Procedure definite dall'utente

Possono essere, a loro volta, di due tipi: Transact-SQL e CLR. Entrambe le stored procedure accettano in input e restituiscono in output parametri definiti

dall'utente. La loro diversità sta nel fatto che le stored procedure di tipo Transact-SQL sono insiemi di istruzioni scritte nell'omonimo linguaggio mentre una di tipo CLR è un riferimento ad un metodo denominato Common Language Runtime.

- Stored Procedure Estese

Consentono di creare programmi esterni con un linguaggio di programmazione diverso come C e tali che SQL Server riesca a caricare ed eseguire in modo dinamico. Questo tipo di stored procedure però verrà rimossa nelle versioni future.

- Stored Procedure di sistema

Sono un tipo speciale di procedura utilizzate per eseguire molte attività amministrative di SQL Server. Esse sono archiviate nel database Resource, database di sola lettura che contiene tutti gli oggetti di sistema di SQL Server. Vengono identificate con il prefisso sp_NomeStoredProcedure proprio per indicare che si tratta di una stored procedure di sistema. E' consigliabile, infatti, chiamare le stored Transact-SQL con spNomeStoredProcedure per differenziarle da quelle di sistema.

4.3 Creazione di una stored procedure con Management Studio

Per creare la stored procedure, che ho chiamato "SPOperation", ho utilizzato Microsoft SQL Server Management Studio Express. Con questo programma la creazione di una stored procedure risulta abbastanza semplice, infatti, dopo l'apertura del programma e la scelta del database nel quale lavorare, espandendo la cartella del database si trovano varie voci tra le quali i diagrammi del database, le tabelle e la programmabilità. Entrando nella cartella programmabilità si trovano le stored procedure, le funzioni ed altre voci. Per inserire una nuova stored procedure dobbiamo cliccare con il tasto destro sopra l'omonima cartella e scegliere la voce "Nuova Stored Procedure". Si apre

così una nuova finestra editor con l’intestazione della stored procedure e i comandi di creazione, come mostrato in figura 4.1:

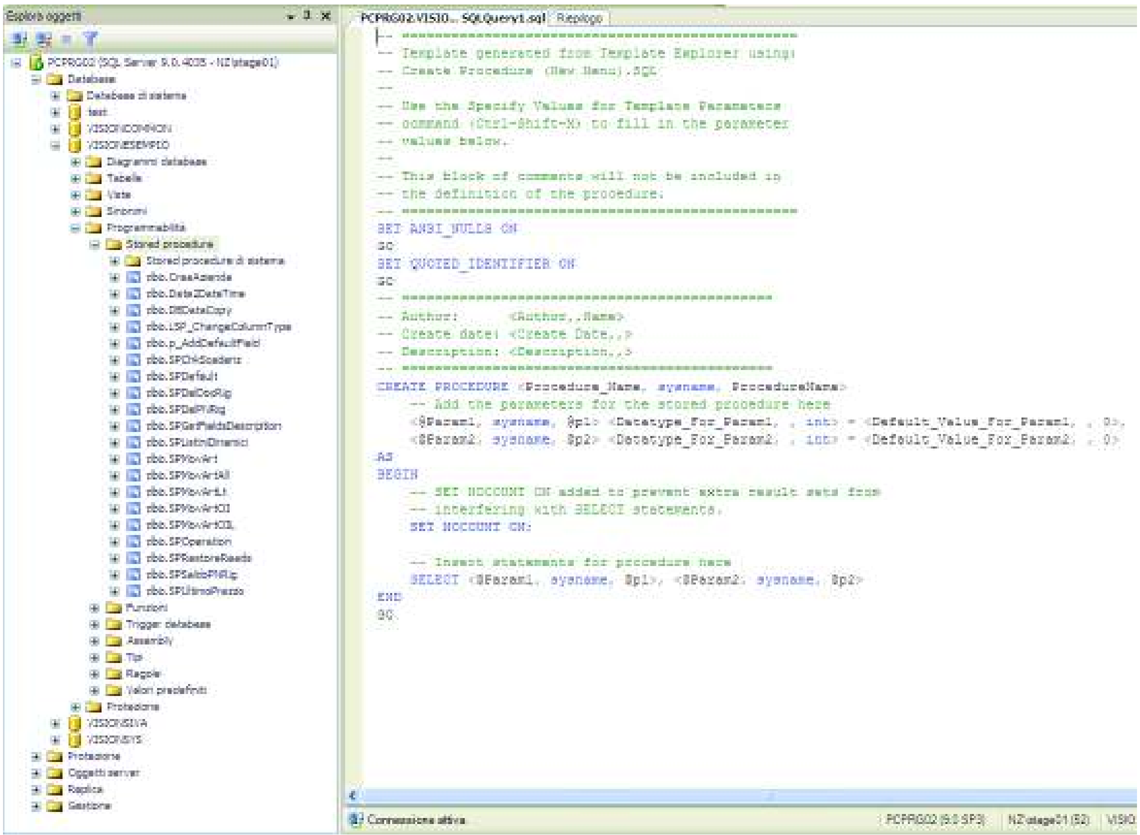


Figura 4.1: Le stored procedure in Microsoft SQL Server Management Studio

Nella creazione di una stored procedure di tipo Transact-Sql è molto importante dichiarare i parametri che questa dovrà ricevere in input quando verrà eseguita. I comandi per la creazione, come mostrato anche nella figura precedente, sono i seguenti:

\- -

```
CREATE PROCEDURE [dbo].[NomeStoredProcedure]
        @NomeParametro1 TipoParametro = ValoreDiDefault
        @NomeParametro2 TipoParametro = ValoreDiDefault
        ……..
        @NomeParametroN TipoParametro = ValoreDiDefault
```

```
AS
BEGIN
        Istruzioni T-SQL
END
```

Il “TipoParametro” indica il tipo di dato che può assumere un parametro. Un parametro può essere una stringa (VARCHAR), un numero (Integer), o può assumere valori logici (bit), e molti altri. Il “ValoreDiDefault” è il valore che viene assegnato ad un certo parametro nel caso in cui l’utente non gli attribuisse alcun valore.

Ad esempio, supponendo che il @NomeParametro1 sia di tipo VARCHAR, il valore di default per una stringa potrebbe essere uno spazio vuoto ‘ ’, mentre se fosse di tipo Integer, il valore di default per un numero potrebbe essere 0.

Dopo la dichiarazione dei parametri si inizia a creare il corpo della stored procedure, dichiarando come prima cosa le variabili che saranno utilizzate solo all’interno di essa. I comandi per la dichiarazione delle variabili sono:

```
DECLARE @NomeVariabile AS TipoVariabile;
```

La dichiarazione delle variabili prevede che il nome di una variabile inizi con il carattere chiocciola (@) e che sia definito il tipo della variabile. Quando una variabile viene dichiarata il valore che essa assume è NULL. Per inizializzarla è quindi necessario inserire nel corpo della stored procedure la seguente istruzione:

```
SET @NomeVariabile = ‘Valore’
```

Dopo aver inizializzato le variabili si procede inserendo le istruzioni necessarie per il funzionamento della stored procedure.

Terminata la creazione della stored procedure sarà possibile richiamarla ogniqualvolta se ne abbia la necessità. I comandi per l'esecuzione della stored procedure sono i seguenti:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
EXEC [dbo].[NomeStoredProcedure]
        @NomeParametro1 = ValoreParametro1
        @NomeParametro2 = ValoreParametro2
        ……..
        @NomeParametroN = ValoreParametroN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Il "ValoreParametro1" indica il valore che il Parametro1 assume. Ad esempio se inserissimo l'esecuzione della stored procedure nel trigger di inserimento della tabella Cespiti e il nome parametro fosse "@Tabella", per indicare il nome della tabella, il valore che dovrebbe assumere sarebbe 'Cespiti'.

4.4 I trigger

L'esecuzione della stored procedure, come detto in precedenza, è stata inserita in tutti i trigger di ogni tabella del database. Mi sembra quindi corretto, prima di iniziare a spiegare il lavoro che ho svolto, introdurre, in modo teorico, la definizione di trigger e spiegare la loro utilità.
Un trigger è un tipo speciale di stored procedure che si attiva automaticamente quando si scatena un evento. Esistono due tipi di trigger:

- i trigger DDL, si attivano quando si scatena un evento DDL (Data Definition Language) nel server o nel database;
- i trigger DML, vengono richiamati quando si verifica un evento DML (Data Manipulation Language) nel database.

I trigger nei quali ho inserito il codice per l'esecuzione della stored procedure sono di tipo DML. Gli eventi che fanno scatenare questo tipo di trigger sono le istruzioni di INSERT, UPDATE e DELETE su una tabella.

A loro volta i trigger DML possono essere di due tipi: AFTER o INSTEAD OF. I primi, di tipo AFTER, vengono richiamati dopo l'esecuzione dell'istruzione di inserimento, modifica o cancellazione. I secondi invece, di tipo INSTEAD OF, vengono eseguiti in sostituzione dell'evento che li ha richiamati.
I trigger vengono utilizzati per vari motivi:

- mantenere l'integrità referenziale tra le tabelle senza dover necessariamente porre un vincolo,
- mantenere l'integrità dei dati di ogni singola tabella,
- monitorare i campi di una tabella ed eventualmente generare eventi ad hoc.

Si deve comunque stare attenti all'utilizzo che si fa dei trigger perché aggiungono un carico di extra-lavoro a SQL Server. Finché si lavora con tabelle con poche centinaia di righe l'uso del trigger non sconvolge le prestazioni del sistema, ma se si dovesse lavorare con tabelle con milioni di righe sarebbe meglio riflettere per capire quanto sia conveniente l'implementazione di un trigger.

4.5 Prima bozza della stored procedure "SPOperation"

Non avendo mai utilizzato le stored procedure, ho iniziato il mio lavoro creando una semplice stored procedure che venisse richiamata ogniqualvolta venisse effettuata un'operazione di inserimento, modifica o cancellazione e che emettesse dei messaggi contenenti il tipo di operazione e il nome della tabella in cui era stata effettuata.
"SPOperation" è il nome della stored procedure che ho creato. La prima bozza era una stored procedure che riceveva in ingresso due parametri: il nome della tabella e il tipo di operazione, e restituiva un messaggio in cui era spiegato il tipo di operazione e in quale tabella era avvenuta. Il parametro riguardante il tipo di operazione effettuata era di un solo carattere: I per inserimento, U per modifica e D per cancellazione.

Per semplificare la comprensione riporto di seguito la prima bozza di "SPOpertation".

```
-- ================================
-- Author:          Irene Sguotti                          Intestazione della
-- Create date:     01/06/2010                             Stored Procedure
-- Description:     SP per Operation Log
-- ================================
CREATE PROCEDURE [dbo].[SPOperation]
        -- Add the parameters for the stored procedure here
        @NomeTabella VARCHAR(50) = '',                     Dichiarazione dei
        @Ingresso CHAR(1) = ''                             parametri d'ingresso
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

        DECLARE @Message VARCHAR(500);                     Dichiarazione delle variabilli

        SET @Message = ''                                  Inizializzazione della variabile
        --Controllo il tipo di operazione che è stata eseguita
        IF (@Ingresso = 'I')
        BEGIN
                SET @Message = @Message + N'Inserimento' + CHAR(13)
        END;

        IF (@Ingresso = 'U')
        BEGIN
                SET @Message = @Message + N'Aggiornamento' + CHAR(13)
        END;

        IF (@Ingresso = 'D')
        BEGIN
                SET @Message = @Message + N'Cancellazione' + CHAR(13)
        END;

        SET @Message = 'Nella tabella ' + @NomeTabella + ' è stata effettuata una operazione di' +
        CHAR(13) + @Message;

        PRINT @Message                                     Stampo a video il messaggio

        SELECT @Message AS Messaggio

END
```

Il codice di questa prima bozza è molto semplice. Dopo la dichiarazione dei due parametri (@NomeTabella e @Ingresso) e della variabile @Message, ho inserito delle istruzioni IF per sapere il tipo di operazione effettuata. Ciò è

servito perché il parametro @Ingresso è composto da un solo carattere (I/U/D), quindi ho dovuto specificare che I indica l'inserimento, U la modifica e D la cancellazione. Dopo queste istruzioni, avevo tutte le informazioni necessarie per creare il messaggio nel quale ho specificato la tabella in cui fosse stata effettuata l'operazione e il tipo di operazione eseguita.

4.6 Alcuni tentativi di funzionamento

Lo scopo della creazione di questa prima stored procedure è stato quello di fare in modo che ogniqualvolta venisse eseguita un'operazione di inserimento, modifica o cancellazione apparisse automaticamente a video un messaggio contenente i dettagli dell'operazione effettuata. È stato quindi necessario richiamare la stored procedure al verificarsi di ogni operazione di inserimento, modifica e cancellazione. Non esiste però un metodo per fare attivare automaticamente una stored procedure. Le uniche "procedure" che vengono richiamate in automatico allo scatenarsi di un evento di INSERT, UPDATE o DELETE, sono i trigger, perciò è stato necessario aggiungere a questi il comando per eseguire la stored procedure. Se non avessi utilizzato questo metodo, avrei dovuto inserire in tutti i trigger di ogni tabella le istruzioni contenute nel corpo della stored procedure aumentando così il carico di lavoro e diminuendo le prestazioni stesse del programma.
Per verificare il funzionamento della prima bozza di "SPOperation" ho richiamato la stored procedure nei trigger delle tabelle "Art", "CliFor" e "Agenti". Il codice che ho inserito, come ho spiegato precedentemente, in questi trigger è il seguente:

```
-- Irene Sguotti 03/06/2010: Aggiunto richiamo alla stored procedure
EXEC dbo.SPOperation
        @NomeTabella = 'Art',
        @Ingresso = 'D'
```

Nell'esempio, sopra mostrato, l'esecuzione della stored procedure è stata inserita nel trigger di cancellazione della tabella "Art". I parametri che ho

passato a “SPOperation” sono “Art” per il nome della tabella e “D” per il tipo di operazione.
Dopo aver inserito nei trigger le istruzioni per richiamare la stored procedure nel momento in cui vengono inseriti, modificati o cancellati i dati, ho provato a fare degli inserimenti, delle modifiche e delle cancellazione in "VisionSQL” per verificare che “SPOperation” emettesse i messaggi che ho composto all’interno di essa.
In fase di cancellazione di un articolo, ad esempio, dopo l’esecuzione di “SPOperation”, il messaggio che viene emesso è il seguente:

Figura 4.2: Messaggio emesso da “SPOperation”

Terminati i tentativi di funzionamento in tutte le tabelle in cui ho richiamato la stored procedure, ho continuato il mio progetto di stage perfezionando “SPOperation”.

5. Perfezionamento di “SPOperation”

5.1 Creazione della tabella “LOGDB” e modifica di “SPOperation”

Dopo aver completato la prima bozza di “SPOperation”, ho proseguito il mio progetto di stage creando una nuova tabella “LOGDB”. Questa è stata creata per dare la possibilità agli utenti di consultare lo storico delle operazioni di inserimento, modifica e cancellazione effettuate. “LOGDB”, infatti, ha il compito di contenere tutte le informazioni relative ad un’operazione. L’inserimento di tali informazioni in questa tabella deve essere effettuato dalla stored procedure; quindi ho dovuto modificare “SPOperation” in modo che, invece di emettere i messaggi contenenti il tipo di operazione effettuata e il nome della tabella interessata, andasse ad inserire nella tabella “LOGDB” le informazioni relative ad una certa operazione.
La creazione della tabella “LOGDB”, grazie a Management Studio, è stata molto semplice e veloce. Dopo aver scelto il database in cui voler creare la tabella bisogna posizionarsi sopra la voce “Tabelle”, cliccare con il tasto destro del mouse sopra ad essa ed infine selezionare la voce “Nuova Tabella”. Si aprirà così una nuova finestra dove sarà necessario inserire semplicemente i campi della tabella. Nell’inserimento dei campi bisognerà specificare, oltre al nome, il tipo di dato, il valore di default che questo assume in caso di mancato inserimento e la possibilità o meno di assumere valore “NULL”. Completata la fase di inserimento dei campi basterà semplicemente salvare la nuova tabella appena creata, assegnandole il nome che si desidera.
I campi che la tabella “LOGDB” contiene sono i seguenti:

- Id: è la chiave primaria della tabella;
- CodUtente: indica l’utente che ha effettuato l’operazione;
- Operation: contiene il tipo di operazione che è stata eseguita;
- Tabella: indica il nome della tabella in cui è stata fatta l’operazione;
- ModField: in caso di operazioni di UPDATE contiene la descrizione dei campi modificati con valori vecchi e nuovi; in caso di operazioni di

DELETE o INSERT contiene il valore della chiave primaria cancellata o inserita;

- Data: indica la data in cui è stata effettuata l'operazione;
- PK: contiene il sottoinsieme di attributi che formano la chiave primaria della tabella in cui è avvenuto l'inserimento, la modifica o la cancellazione.

La creazione della tabella "LOGDB", avvenuta con Management Studio, corrisponde alle seguenti istruzioni T-SQL:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
CREATE TABLE [dbo].[LOGDB](
	[Id_LOGDB] [int] IDENTITY(1,1) NOT NULL,
	[CodUtente] [varchar](30) NOT NULL CONSTRAINT [DF_LOGDB_CodUtente]  DEFAULT ((0)),
	[Operation] [varchar](15) NOT NULL CONSTRAINT [DF_LOGDB_Operation]  DEFAULT (''),
	[Tabella] [varchar](30) NOT NULL CONSTRAINT [DF_LOGDB_Tabella]  DEFAULT ((0)),
	[ModField] [varchar](max) NOT NULL CONSTRAINT [DF_LOGDB_ModField]  DEFAULT ((0)),
	[Data] [datetime] NOT NULL CONSTRAINT [DF_LOGDB_Data]  DEFAULT (((1)/(1))/(1900)),
	[PK] [varchar](100) NOT NULL CONSTRAINT [DF_LOGDB_PK]  DEFAULT ((0)),
 CONSTRAINT [PK_LOGDB] PRIMARY KEY CLUSTERED
(
	[Id_LOGDB] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Terminata la creazione della tabella "LOGDB", ho iniziato a modificare la stored procedure.

"SPOperation" deve raccogliere tutte le informazioni riguardanti un'operazione di inserimento, modifica o cancellazione in modo da inserirle e salvarle nella tabella "LOGDB", la quale poi potrà essere consultata dagli utenti. La "nuova" stored procedure, a differenza della prima bozza precedentemente creata, per ottenere tutte le informazioni di un'operazione dovrà ricavare l'utente che l'ha

effettuata, la chiave primaria della tabella in cui è stata fatta, la data in cui è stata eseguita e costruire il valore del campo 'ModField'.

Per ricavare il codice dell'utente che ha effettuato l'operazione ho dovuto inserire nella stored procedure un nuovo parametro: @Spid. Questo parametro contiene la sessione in cui è stata effettuata l'operazione e viene estratta con una semplice query, aggiunta in tutti i trigger. La query che ho aggiunto nei trigger è la seguente:

```
SELECT @Sessione = SPID FROM VISIONSYS.dbo.UsrLog
```

Con questa query salvo dentro alla variabile @Sessione, la sessione in cui l'utente ha effettuato una certa operazione, ricavandola dalla tabella "UsrLog".

Nella stored procedure poi ho aggiunto una query che, attraverso la sessione, ricava il codice dell'utente. Il codice che permette di effettuare ciò è il seguente:

```
SELECT @CodUtente = CodUsrLog FROM VISIONSYS.dbo.UsrLog WHERE SPID = @Spid;
```

Per la ricerca del sottoinsieme di attributi che formano la chiave primaria della tabella in cui è stata effettuata l'operazione ho dovuto aggiungere nella stored procedure la seguente query:

```
SELECT @CodChiave = @CodChiave + ' ' + CU.COLUMN_NAME

FROM INFORMATION_SCHEMA.TABLES AS T LEFT JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS
AS TC  ON (TC.TABLE_CATALOG = T.TABLE_CATALOG AND TC.TABLE_SCHEMA = T.TABLE_SCHEMA AND
TC.TABLE_NAME = T.TABLE_NAME AND TC.CONSTRAINT_TYPE = 'PRIMARY KEY')
LEFT JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS CU ON (CU.CONSTRAINT_CATALOG =
TC.CONSTRAINT_CATALOG AND CU.CONSTRAINT_SCHEMA = TC.CONSTRAINT_SCHEMA AND
CU.CONSTRAINT_NAME = TC.CONSTRAINT_NAME)

WHERE T.TABLE_NAME = @NomeTabella

ORDER BY CU.COLUMN_NAME
```

In questa query ho utilizzato l'INFORMATION_SCHEMA. Si tratta di un metodo offerto da SQL Server 2005 che fornisce una vista dello schema delle informazioni di un database. L'INFORMATION_SCHEMA può fornire l'elenco di una parte del database a seconda delle esigenze del programmatore. Nel mio lavoro ho utilizzato:

- "INFORMATION_SCHEMA.TABLE": fornisce l'elenco di tutte le tabelle presenti nel database,
- "INFORMATION_SCHEMA.TABLE_CONSTRAINTS": produce l'elenco di tutte le tabelle aventi un vincolo,
- "INFORMATION_SCHEMA.KEY_COLUMN_USAGE": fornisce l'elenco di tutte le tabelle che hanno un vincolo di chiave primaria.

Attraverso queste istruzioni sono riuscita ad estrarre la chiave primaria della tabella in cui è stata effettuata la stored procedure e salvarla nella variabile "@CodChiave".

Per ricavare la data in cui l'operazione è stata eseguita è stato sufficiente aggiungere una singola istruzione:

```
SET @Data = GETDATE();
```

Con l'istruzione GETDATE() assegno alla variabile "@Data", tutte le volte in cui viene richiamata la stored procedure, la data e l'ora in cui un'operazione è stata effettuata.

Successivamente ho iniziato a creare il contenuto del campo 'ModField'. Questo campo ha una duplice funzione:

- per gli inserimenti e le cancellazioni deve contenere il valore della chiave primaria del record che è appena stato inserito o cancellato
- per le modifiche invece deve contenere la spiegazione dei cambiamenti che sono stati apportati quindi per ogni campo deve essere mostrato il

valore vecchio, quello prima della modifica, e il valore nuovo, quello dopo la modifica.

Per implementare la prima funzione che deve svolgere questo campo è stato sufficiente aggiungere alla stored procedure un nuovo parametro @Cod. A questo parametro viene assegnato il valore della chiave primaria inserita o cancellata nel momento in cui viene richiamata la stored procedure nei trigger.

```
-- Irene Sguotti 11/06/2010: Richiamo la Stored Procedure
EXEC dbo.SPOperation
        @NomeTabella = 'Agenti',
        @Ingresso = 'I',
        @Spid = @Sessione,
        @Cod = @CodAgente
```

Nell'esempio la stored procedure viene richiamata nel trigger di inserimento della tabella Agenti. Notiamo che al parametro @Cod viene assegnata la variabile @CodAgente. Questa variabile, grazie alla tabella inserted (o deleted in caso di cancellazione) contiene il valore della chiave primaria, in questo caso CodAgente, che è stata appena inserita (o cancellata). Le tabelle inserted e deleted infatti sono delle tabelle temporanee create e gestite automaticamente da SQL Server nelle quali vengono archiviate copie delle righe interessate nell'esecuzione delle istruzioni INSERT o DELETE. Nella tabella inserted quindi sono archiviate copie delle nuove righe mentre nella tabella deleted copie delle righe che sono state cancellate. Da queste tabelle quindi posso ricavarmi il valore inserito o cancellato della chiave primaria, salvarlo in una variabile e poi assegnarlo al parametro della stored procedure. Nel corpo di "SPOperation", creo poi una frase così formata:

```
SET @ModField = @CodChiave + ': ' + @Cod
```

La variabile @ModField comprende il nome della chiave primaria situato nella variabile @CodChiave e il valore appena inserito o cancellato della chiave contenuto nel parametro @Cod.

Per le operazioni di modifica nel campo 'ModField' ho dovuto inserire il nome del campo modificato con il relativo vecchio valore, ossia quello precedente alla modifica, e il nuovo valore, quello attuale. Per completare questo compito ho incontrato delle difficoltà perché avrei dovuto confrontare tutti i valori contenuti nelle tabelle inserted e deleted e recuperare solo quelli differenti. La modifica infatti può essere vista come la cancellazione del valore vecchio e l'inserimento del valore attuale. SQL Server però non dispone di funzioni e comandi che mi permettessero di confrontare e recuperare i valori diversi delle tabelle inserted e deleted. E' necessario però spiegare che l'applicativo "VisionSQL" è stato sviluppato in linguaggio Microsoft Visual FoxPro. Questo linguaggio mi ha permesso di portare a termine il mio compito ed implementare la seconda funzionalità del campo 'ModField' attraverso la seguente funzione da me creata:

```
&& Irene Sguotti 30/07/2010
*************************************************************************
*Funzione per l'Operation Log
*************************************************************************
FUNCTION InsOperation(cAlias as String, Tabella as String)
**!** cAlias      : Alias da utilizzare per verificare le modifiche
   LOCAL cSqlStmt as String
   LOCAL cChiavePrimaria as String
   LOCAL cField as String

**!** Creazione della query per estrarre il nome della chiave primaria della tabella
  TEXT TO cSelect TEXTMERGE NOSHOW
         CU.COLUMN_NAME
   ENDTEXT

   TEXT TO cFrom TEXTMERGE NOSHOW
     INFORMATION_SCHEMA.TABLES AS T LEFT JOIN
     INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
     ON (TC.TABLE_CATALOG = T.TABLE_CATALOG AND TC.TABLE_SCHEMA = T.TABLE_SCHEMA
     AND TC.TABLE_NAME = T.TABLE_NAME AND TC.CONSTRAINT_TYPE = 'PRIMARY KEY')
     LEFT JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS CU
     ON CU.CONSTRAINT_CATALOG = TC.CONSTRAINT_CATALOG
```

```
    AND CU.CONSTRAINT_SCHEMA = TC.CONSTRAINT_SCHEMA AND
    CU.CONSTRAINT_NAME = TC.CONSTRAINT_NAME
  ENDTEXT

  TEXT TO cWhere TEXTMERGE NOSHOW
        T.TABLE_NAME = <<Format(cAlias)>>
  ENDTEXT
**!** Fine creazione query

**!** Creazione del cursore per il recupero della chiave primaria
  UseInTable("crPK")

**!** Funzione per l'esecuzione della query e l'estrazione della chiave primaria
  UseTable(cFrom, oApp.nConnAz, "crPK", cSelect, cWhere)

**!**Salvataggio della chiave primaria nella variabile
  cChiavePrimaria = ""
  SCAN
    cChiavePrimaria = cChiavePrimaria + ' ' + STRTRAN(Format(crPK.Column_Name, .T.), """", "")
  ENDSCAN

  cSqlStmt = ""
  cField = ""

**!** Istruzioni di confronto tra il nuovo valore "EVALUATE" e il vecchio "OLDVAL" e creazione
**!** della variabile cField contenente il nome del campo modificato, il valore vecchio e quello
**!** nuovo
  SELECT(cAlias)
  FOR n = 1 TO FCOUNT(cAlias)
        IF NOT (OLDVAL(FIELD(n)) == EVALUATE(FIELD(n)))
          IF TYPE("EVALUATE(FIELD(n))") == "D" OR TYPE("EVALUATE(FIELD(n))") == "T"
                cField = cField + ;
                        ALLTRIM(FIELD(n)) + " - " + oApp.aLang[3930] + """ + ;
                        Format(DTOC(OLDVAL(FIELD(n))), .T.)  + "' - " + ;
                        oApp.aLang[3931] + """ + Format(DTOC(EVALUATE(FIELD(n))), .T.) + ;
                        """ + CHR(13)
          ELSE
                cField = cField +;
                        ALLTRIM(FIELD(n)) + " - " + oApp.aLang[3930] + ;
                        IIF(TYPE(FIELD(n)) == "N", """", """) + ;
                        Format(OLDVAL(FIELD(n)), .T.) + ;
                        IIF(TYPE(FIELD(n)) == "N", """", """) + " - " + ;
                        oApp.aLang[3931] + ;
```

```
                        IIF(TYPE(FIELD(n)) == "N", "'", "") + ;
                        Format(EVALUATE(FIELD(n)), .T.) + ;
                        IIF(TYPE(FIELD(n)) == "N", "'", "") + CHR(13)
            ENDIF
        ENDIF
    ENDFOR

**!** Inserimento dei dati nella tabella LOGDB
    IF !EMPTY(cField)
        cSqlStmt="INSERT INTO LOGDB (Tabella, PK, Operation, CodUtente, ModField, Data)"+;
                CHR(13) + ;
                "VALUES " + Format(Tabella) + Format(cChiavePrimaria) + " 'U' " + ;
                Format(oApp.UserName) + cField + Format(DATE()) + CHR(13)

        UpdateData(cSqlStmt, oApp.nConnAz)
    ENDIF
ENDFUNC
**!** Fine Funzione
```

Questa funzione, denominata “InsOperation”, contiene quasi le stesse istruzioni della stored procedure. La differenza principale, a parte per il linguaggio di programmazione, sta nel fatto che “SPOperation” raccoglie le informazioni delle operazioni di inserimento e cancellazione mentre “InsOperation” si occupa delle informazioni delle operazioni di modifica. “InsOperation”, infatti, attraverso un confronto tra i nuovi valori, ricavati con il comando EVALUATE(), e i vecchi valori, ricavati con OLDVAL(), ricava le informazioni necessarie per creare il contenuto del campo ‘ModField’. Conclusa la creazione della tabella e la modifica di “SPOperation” ho esteso la chiamata della stored procedure a tutti i trigger e “testato” il mio lavoro.

5.2 Modifica dei trigger

In questa fase del mio progetto ho esteso la chiamata di “SPOperation” a tutti i trigger di inserimento e cancellazione. Nei trigger di modifica non è servito aggiungere alcuna chiamata perché la chiamata alla funzione “InsOperation” è

stata inserita in Fox. In definitiva il codice che ho dovuto aggiungere nei trigger di cancellazione ed inserimento per l'esecuzione della stored procedure è:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

-- Irene Sguotti 01/06/2010: Richiamo la stored procedure
EXEC dbo.SPOperation
        @NomeTabella = 'Agenti',
        @Ingresso = 'D',
        @Spid = @Sessione,
        @Cod = @CodAgente
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Non tutti gli attributi che formano le chiavi primarie delle tabelle sono dichiarati come tipo di dato VARCHAR, ossia carattere. Alcuni, infatti, possono essere dichiarati come Integer, numeri, o altro. Il parametro @Cod però è un carattere, quindi nel caso in cui una chiave primaria non dovesse essere un carattere, prima di richiamare la stored procedure, è necessario convertire tale chiave in carattere, con le seguenti istruzioni:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

@Codice = CONVERT(VARCHAR(100), @IDAgente)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Nell'esempio ho ipotizzato che la chiave primaria della tabella Agenti sia IDAgente, numerica e quindi l'ho convertita in modo che il tipo di dato sia VARCHAR e quindi sia un carattere e non più un numero.
Alcune tabelle non avevano i trigger di inserimento, quindi ho dovuto crearli.
Sistemata l'esecuzione della stored procedure, ho testato il mio lavoro inserendo, modificando e cancellando dei dati attraverso "VisionSQL".
Dopo aver fatto qualche operazione, per verificare il funzionamento della stored procedure, controllavo, attraverso SQL Server, se nella tabella LOGDB erano state inserite le informazioni riguardanti le operazioni da me effettuate.

Id_LOGDB	CodUtente	Operation	Tabella	ModField	Data	PK
64	ADMINISTRATOR	Inserimento	RISORS		15/06/2010 12....	CodRisors
65	ADMINISTRATOR	Cancellazione	ATTIVIT		15/06/2010 12....	Id_Attivit
66	ADMINISTRATOR	Cancellazione	ALLEGATI		15/06/2010 12....	Id_Allegato
67	ADMINISTRATOR	Cancellazione	RISORS		15/06/2010 12....	CodRisors
68	ADMINISTRATOR	Inserimento	DIPEN		15/06/2010 12....	CodDipen
69	ADMINISTRATOR	Modifica	DIPEN	Vecchio Valore:'Dipen XX' - 'Nuovo Valore:'Dipen tets'	15/06/2010 0.0...	CodDipen
70	ADMINISTRATOR	Cancellazione	ATTIVIT		15/06/2010 12....	Id_Attivit
71	ADMINISTRATOR	Cancellazione	ALLEGATI		15/06/2010 12....	Id_Allegato
72	ADMINISTRATOR	Cancellazione	DIPEN		15/06/2010 12....	CodDipen
73	ADMINISTRATOR	Inserimento	ASPEST		15/06/2010 12....	CodAspEst
74	ADMINISTRATOR	Modifica	ASPEST	Vecchio Valore:'aspEsr xx' - 'Nuovo Valore:'aspEsr xx 22'	15/06/2010 0.0...	CodAspEst
75	ADMINISTRATOR	Cancellazione	ASPEST		15/06/2010 12....	CodAspEst
76	ADMINISTRATOR	Modifica	LINGUE	Vecchio Valore:'linguaUS' - 'Nuovo Valore:'lingua test'	15/06/2010 0.0...	CodLingua
77	ADMINISTRATOR	Modifica	LINGUE	Vecchio Valore:'linguaCC' - 'Nuovo Valore:''	15/06/2010 0.0...	CodLingua
78	ADMINISTRATOR	Modifica	LINGUE	Vecchio Valore:'' - 'Nuovo Valore:'lingua prova'	15/06/2010 0.0...	CodLingua
79	ADMINISTRATOR	Cancellazione	LINGUE		15/06/2010 12....	CodLingua
80	ADMINISTRATOR	Inserimento	PORTI		15/06/2010 12....	CodPorto
81	ADMINISTRATOR	Modifica	PORTI	Vecchio Valore:'' - 'Nuovo Valore:'002211'	15/06/2010 0.0...	CodPorto
82	ADMINISTRATOR	Modifica	PORTI	Vecchio Valore:'porto xx' - 'Nuovo Valore:'porto yy'	15/06/2010 0.0...	CodPorto
83	ADMINISTRATOR	Cancellazione	PORTI		15/06/2010 12....	CodPorto
84	ADMINISTRATOR	Inserimento	SETTATT		15/06/2010 12....	CodSettAtt
85	ADMINISTRATOR	Modifica	SETTATT	Vecchio Valore:'settore xx' - 'Nuovo Valore:'settore zz'	15/06/2010 0.0...	CodSettAtt
86	ADMINISTRATOR	Cancellazione	SETTATT		15/06/2010 12....	CodSettAtt

Figura 5.1: Tabella LOGDB

Si nota che la stored procedure funziona correttamente, inserendo le informazioni delle operazioni nella tabella LOGDB.

5.3 Criptatura di “LOGDB”

Conclusi i tentativi di funzionamento di “SPOperation”, per le modifiche appena apportate, ho proseguito il mio stage svolgendo una fase facoltativa che prevedeva un’attività di criptatura della “LOGDB” e la costruzione dell’interfaccia grafica per rendere la tabella consultabile tramite il software gestionale “VisionSQL”.

Mi è stato chiesto di criptare la tabella per rendere i dati maggiormente protetti e accessibili solo agli utenti di tipo “ADMINISTRATOR” e quindi consultabili solo per coloro che gestiscono l’applicazione.

Nel linguaggio Transact-SQL esistono vari metodi per la criptatura dei dati. Il metodo da me scelto prevede l’utilizzo di una specifica chiave, una chiave simmetrica, attraverso la quale è possibile inserire nella tabella “LOGDB” i dati criptati. Questo metodo ritorna solo valori di tipo VARBINARY, quindi oltre a

modificare la stored procedure e la funzione, ho dovuto modificare il tipo di dato dei campi 'CodUtente', 'Tabella', 'ModField' e 'CodChiave'.
Dopo aver trasformato, da VARCHAR a VARBINARY, i campi della tabella "LOGDB" ho creato all'interno del database la chiave simmetrica. Il codice utilizzato per la creazione di questa chiave è il seguente:

- -

```
CREATE SYMMETRIC KEY NomeChiave WITH ALGORITHM = TRIPLE_DES ENCRYPTION
BY PASSWORD = 'NomePassword'
```

- -

Successivamente ho modificato la stored procedure criptando i dati prima di inserirli nella tabella. Per criptare i dati è stato necessario utilizzare questa riga di codice:

- -

```
EncryptByKey(Key_GUID('NomeChiave'),@NomeVariabile)
```

- -

Riporto qui di seguito la stored procedure "SPOperation" definitiva che inserisce le informazioni delle varie operazioni nella tabella "LOGDB".

```
-- =============================================
-- Author:                  Irene Sguotti
-- Create date: 01/06/2010
-- Description:   SP per Operation Log
-- =============================================
ALTER PROCEDURE [dbo].[SPOperation]
        -- Add the parameters for the stored procedure here
        @NomeTabella VARCHAR(100) = '',
        @Ingresso CHAR(1) = '',                                    Dichiarazione dei
        @Spid int = 0,                                               parametri
        @Cod VARCHAR(100) = ''

AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

        DECLARE @Data DATETIME;
        DECLARE @CodChiave VARCHAR(100);
```

```
        DECLARE @CodUtente VARCHAR(20);
        DECLARE @ModField VARCHAR(500);                    Dichiarazione delle
        DECLARE @TabellaCript VARBINARY(8000);                  variabili
        DECLARE @UtenteCript VARBINARY(8000);
        DECLARE @ChiaveCript VARBINARY(8000);
        DECLARE @ModCript VARBINARY(8000);
        DECLARE @Operation BIT;

        SET @Operation = 0;
        SET @Data = GETDATE();                             Inizializzazione delle
        SET @CodChiave = '';                                    variabili
        SET @CodUtente = '';
        SET @ModField = '';
         -- Estrazione Chiave Primaria della tabella considerata
        SELECT @CodChiave = @CodChiave + ' ' + CU.COLUMN_NAME
        FROM INFORMATION_SCHEMA.TABLES AS T
         LEFT JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
         ON (TC.TABLE_CATALOG = T.TABLE_CATALOG AND
                TC.TABLE_SCHEMA = T.TABLE_SCHEMA AND               Query per
                TC.TABLE_NAME = T.TABLE_NAME AND                   estrarre la
                TC.CONSTRAINT_TYPE = 'PRIMARY KEY')                chiave primaria
         LEFT JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS CU
         ON (CU.CONSTRAINT_CATALOG = TC.CONSTRAINT_CATALOG AND
                CU.CONSTRAINT_SCHEMA = TC.CONSTRAINT_SCHEMA AND
                CU.CONSTRAINT_NAME = TC.CONSTRAINT_NAME)
        WHERE T.TABLE_NAME = @NomeTabella
        ORDER BY CU.COLUMN_NAME

         -- Estrazione del codice dell'utente che ha effettuato l'operazione
        SELECT @CodUtente = CodUsrLog                              Query per
        FROM VISIONSYS.dbo.UsrLog                                   estrarre
        WHERE SPID = @Spid;                                         l'utente

        SET @ModField = @CodChiave + ': ' + @Cod          Creazione del campo 'ModField'

        -- Apro la chiave simmetrica
        OPEN SYMMETRIC KEY Key01 DECRYPTION BY PASSWORD = 'NomePassword'

        -- Cripto il nome della tabella, il codice dell'utente e la/le chiave/i primaria/e
        SET @TabellaCript =  EncryptByKey(Key_GUID('Key01'),@NomeTabella)
        SET @UtenteCript =  EncryptByKey(Key_GUID('Key01'),@CodUtente)          Criptatura
        SET @ChiaveCript =  EncryptByKey(Key_GUID('Key01'),@CodChiave)           dei dati
        SET @ModCript =  EncryptByKey(Key_GUID('Key01'),@ModField)

        -- Inserisco i dati criptati nella tabella LOGDB
        INSERT INTO LOGDB (Operation, Tabella, Data, PK, CodUtente, ModField)
        VALUES (@Ingresso,@TabellaCript,@Data,@ChiaveCript,@UtenteCript,@ModCript)

        CLOSE SYMMETRIC KEY Key01
END
```

Dopo aver modificato la stored procedure, ho apportato le stesse modifiche anche alla funzione “InsOperation” e ho fatto alcuni tentativi per verificare se i dati venivano criptati o meno.
Dopo aver eseguito alcune operazioni di insermento, modifica e cancellazione nella tabella “LOGDB” venivano inseriti i seguenti dati:

PCPRG02.VISIO...IO - dbo.LOGDB | Riepilogo

Id_LOGDB	CodUtente	Operation	Tabella	ModField	Data	PK
2419	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2420	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2421	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2422	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2423	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2424	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2425	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2426	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2427	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2428	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2429	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2430	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2431	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2432	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2433	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2434	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2435	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2436	<Dati binari>	I	<Dati binari>	<Dati binari>	02/08/2010 11....	<Dati binari>
2437	<Dati binari>	D	<Dati binari>	<Dati binari>	02/08/2010 12....	<Dati binari>
2438	<Dati binari>	D	<Dati binari>	<Dati binari>	02/08/2010 12....	<Dati binari>
2439	<Dati binari>	D	<Dati binari>	<Dati binari>	02/08/2010 12....	<Dati binari>
2440	<Dati binari>	D	<Dati binari>	<Dati binari>	02/08/2010 12....	<Dati binari>

Figura 5.2: Contenuto della tabella “LOGDB”

Terminata la criptatura della tabella “LOGDB” ho iniziato a creare l’interfaccia grafica per la consultazione.

5.4 Creazione dell’interfaccia di consultazione

La creazione dell’interfaccia di consultazione è stata fatta attraverso Microsoft Visual FoxPro. Questo è un linguaggio di programmazione che integra la programmazione di tipo procedurale con quella orientata agli oggetti.

Ho creato quindi una form grafica chiamata “I_OperLog”, la quale è divisa in due parti: la prima parte è riservata all’impostazione dei filtri per mirare la ricerca delle informazioni delle varie operazioni a seconda delle esigenze dell’utente, mentre la seconda parte è formata da una griglia nella quale viene visualizzato lo storico delle operazioni che può quindi essere consultato.

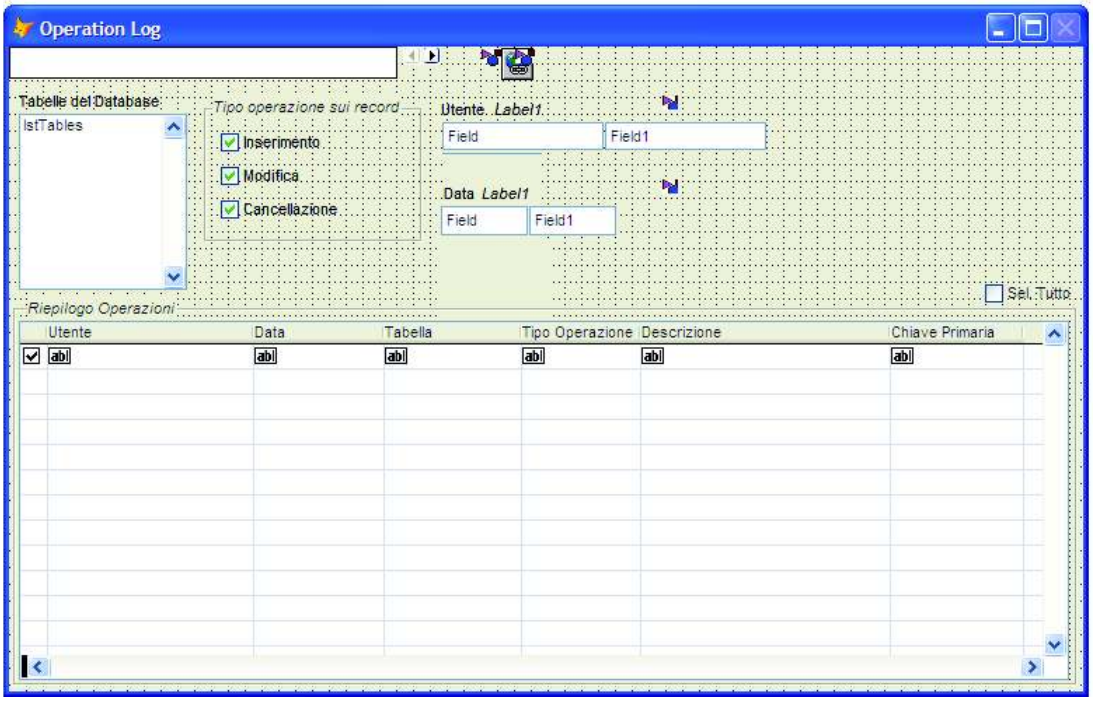


Figura 5.4: Form “I_OperLog”

Naturalmente il lavoro svolto per la creazione di questa form non si è limitato solamente alla creazione della parte grafica ma anche alla scrittura del codice che permette di popolare la lista delle operazioni, applicare i filtri, cancellare righe della tabella “LOGDB”, esportare il contenuto della tabella in file con diverse estensioni (.pdf, .xls, ecc) e rendere la tabella visibile solo agli utenti di tipo “ADMINISTRATOR”. Non riporto il codice che ho inserito per non appesantire troppo la lettura.

Conclusa la creazione della form, ho potuto finalmente effettuare delle operazioni di inserimento, modifica e cancellazione attraverso “VisionSQL” e controllare lo storico delle operazioni direttamente da esso. Dopo alcune operazioni ho aperto la tabella “OperationLog”:

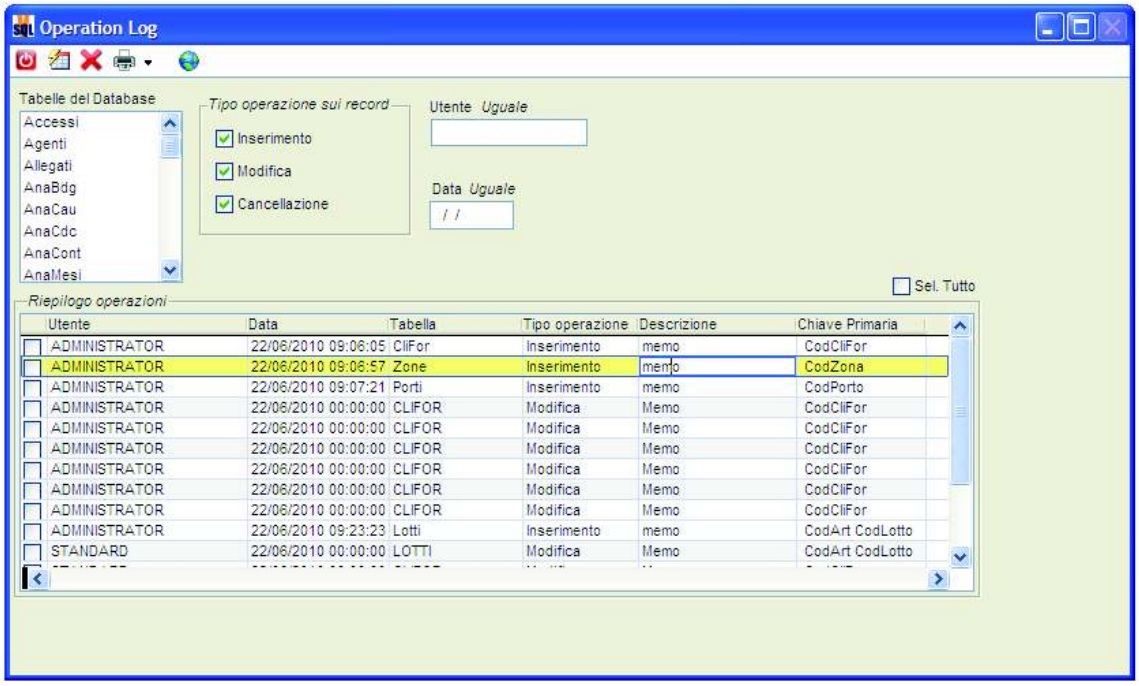


Figura 5.5: “Operation Log”

Questa tabella è visualizzabile e può essere utilizzata solo dagli utenti “ADMINISTRATOR”, da tutti gli altri utenti non può essere visualizzata.

Da questa interfaccia è possibile impostare alcune condizioni di ricerca. E’ possibile infatti consultare le operazioni: o di alcune tabelle, anziché di tutte, oppure di un certo tipo di operazione o che sono state compiute da un determinato utente oppure che sono state effettuate in una certa data. Se non viene impostato alcun filtro viene visualizzato lo storico generale che mostra tutte operazioni effettuate in tutte le tabelle, da qualsiasi utente e in tutte le date. Nella griglia, sottostante i filtri, è visualizzato lo storico delle operazioni. La colonna “Descrizione” contiene la parola “memo”. Con un doppio click sopra ad essa appare per gli inserimenti e le cancellazioni il nome e il valore della chiave primaria appena inserita o cancellata mentre per le modifiche oltre al nome e al valore della chiave primaria viene visualizzato anche il nome del campo modificato con il relativo valore prima della modifica e dopo la modifica. Se vengono effettuate più modifiche nello stesso momento e nella stessa tabella, nella descrizione ci saranno tante righe quante saranno state le modifiche.

ATTIVITA' di TEST

Nell'ultima parte di stage ho potuto osservare da vicino come la statistica può essere molto utile anche in ambito informatico. Mi è stato chiesto di condurre un'attività di test sulle procedure e le modifiche da me apportate all'applicazione. Ho dovuto quindi controllare se dopo l'introduzione della stored procedure, "SPOperation", e della funzione, "InsOperation", i tempi di risposta del database "VisionSQL" fossero aumentati. Dopo aver rilevato tutti i tempi di risposta dell'applicazione in presenza e in assenza delle modifiche fatte ho quindi elaborato i dati calcolando le statistiche necessarie per la verifica d'ipotesi sopra citata.

6. Analisi delle procedure

6.1 Introduzione

Il lavoro che ho svolto in quest'ultima parte di stage è stato quello di effettuare delle operazioni di inserimento, modifica e cancellazione direttamente dall'applicativo "VisionSQL" e di confrontare i tempi di risposta del database in presenza e in assenza della stored procedure o della funzione.
Per ottenere le misurazioni dei tempi di risposta ho creato, grazie all'aiuto del team di programmazione, delle istruzioni in Microsoft Visual FoxPro che generassero un file di log contenente i tempi richiesti. In realtà il file di log conteneva il millisecondo di inizio dell'operazione effettuata e quello di fine. La parte più difficoltosa di questa fase è stato capire dove inserire le istruzioni per generare il file di log. "VisionSQL" infatti, prima di effettuare una qualsiasi operazione, chiede all'utilizzatore, attraverso un messaggio di input, la conferma del comando eseguito. Il tempo che doveva essere rilevato nel file non doveva quindi tener conto del tempo impiegato dall'utilizzatore per confermare l'operazione.
Dopo aver inserito le istruzioni ed aver effettuato le rilevazioni ho potuto eseguire i test che mi erano stati richiesti e stabilire se il lavoro svolto diminuiva in maniera inaccettabile l'efficienza dell'applicazione.

6.2 Raccolta dei dati

L'attività di test è stata effettuata confrontando, per ogni operazione eseguita, i tempi di risposta in presenza della stored procedure o della funzione con quelli in assenza per determinare se la differenza fosse statisticamente significativa. Fin da subito ho notato che per inserimenti e modifiche la differenza era molto piccola, quindi, per queste due operazioni, l'introduzione della stored procedure e della funzione non rallentava eccessivamente l'applicazione. Per quanto riguarda le cancellazioni, invece, mi sono accorta subito che i tempi in presenza di "SPOperation" aumentavano, questo perché nella tabella "LOGDB" oltre ad essere riportata l'operazione di DELETE della tabella interessata, devono essere riportate anche tutte le cancellazioni che il trigger esegue in cascata. Per questo motivo ho eseguito i test nelle tabelle aventi il maggior numero di attributi e il maggior numero di vincoli.
La raccolta dei dati è avvenuta rilevando sessanta osservazioni per ogni operazione, trenta in presenza della stored procedure e trenta in assenza.
I test sono stati effettuati in un database di media dimensione di un'azienda cliente, SIVA, e per le tabelle Art e CliFor. Queste due tabelle contengono la descrizione rispettivamente di un articolo o di un Cliente/Fornitore. Infine ho condotto anche un'attività di test per le operazioni di cancellazione nella tabella DocTes. Questa tabella contiene i documenti di trasporto, le fatture di vendita, le fatture di acquisto e altro. Ho scelto di eseguire i test in questa tabella perché un documento, ad esempio una fattura di acquisto, può contenere molte righe (io ne ho inserite 40) e quindi mi è sembrato interessante confrontare il tempo di cancellazione in assenza e presenza di "SPOperation".

6.3 Elaborazione dei dati

Inizialmente ho pensato di elaborare i dati utilizzando la statistica test "t di Student". La scelta di questo test è stata abbastanza immediata in quanto il lavoro che mi è stato chiesto di svolgere può essere visto dal punto di vista statistico come una verifica d'ipotesi per confrontare le medie dei tempi

provenienti da due popolazioni diverse, una in presenza della stored procedure e l'altra in assenza. Successivamente però ho dovuto adoperare il test "t di Welch" a causa della mancanza di un requisito fondamentale per l'applicazione della "t di Student". Oltre al test, ho costruito degli intervalli di confidenza per il tempo medio nelle due popolazioni.

È importante ricordare che per utilizzare il test "t di Student" è necessario controllare che i seguenti assunti di base siano rispettati:

- le osservazioni delle popolazioni devono essere normali;
- le osservazioni all'interno delle due popolazioni devono essere indipendenti ed identicamente distribuite (i.i.d.);
- le osservazioni tra le due popolazioni devono essere indipendenti;
- le popolazioni devono avere media e varianza finite; le varianze delle due popolazioni devono poter essere considerate uguali.

In merito al primo assunto le osservazioni, ossia i tempi, delle due popolazioni possono essere considerate normali. Ciò perché il Teorema del Limite Centrale ci assicura che qualunque sia la forma della distribuzione di n variabili i.i.d., la somma di esse tende asintoticamente ad una distribuzione normale. Alcuni studi hanno dimostrato che per avere una buona approssimazione è sufficiente $n > 30$ se la distribuzione iniziale è almeno simmetrica, mentre con $n > 50$ l'approssimazione risulta essere buona qualunque sia la distribuzione di partenza.

Come detto in precedenza ho raccolto trenta osservazioni per ogni campione, le quali avevano una distribuzione abbastanza simmetrica, quindi la normalità può essere presunta.

Appena ho iniziato il lavoro mi sono accorta che l'indipendenza tra le osservazioni non era garantita perché SQL Server adotta un meccanismo di "caching". Questo meccanismo prevede il salvataggio in una memoria temporanea di alcune informazioni utili per l'esecuzione di una query, le quali possono essere utilizzate se la query viene rieseguita, diminuendo così i tempi di esecuzione.

Per risolvere questo problema, prima di ogni operazione effettuata, ho eseguito le seguenti istruzioni di codice T-Sql in modo da svuotare la cache e garantire così l'indipendenza delle osservazioni.

```
DBCC DROPCLEANBUFFERS
DBCC FREEPROCCACHE
```

In merito all'indipendenza tra le due popolazioni, non c'è motivo per dubitare che questa sia rispettata.
Infine, per quanto riguarda l'ultimo assunto, la variabilità è sicuramente, come la media, finita per la natura dell'esperimento. L'unico problema che ho incontrato è stato nel determinare l'omoschedasticità delle osservazioni, infatti dopo aver calcolato, con l'aiuto del software statistico R, alcuni test per verificare l'omogeneità delle varianze nelle popolazioni mi sono resa conto che in realtà questo assunto non era rispettato. In nessuna delle popolazioni ho potuto infatti presumere l'omoschedasticità. Per questo motivo non ho potuto utilizzare il test "t di Student" quindi ho dovuto cercare una statistica test alternativa, che tenesse conto della non omogeneità delle varianze. Una statistica che tiene conto dell'eterogeneità delle varianze è la "t di Welch".
Prima di fornire i risultati ottenuti dal test, mi sembra utile per la comprensione riportare le formule utilizzate.

6.4 Formule utilizzate

Per meglio capire ciò che ho eseguito in quest'attività di test riporto qui di seguito le definizioni di media, varianza, intervallo di confidenza e veridica d'ipotesi per concludere con una comparazione delle statistiche test "t di Student" e "t di Welch".
Sia X una variabile aleatoria composta da n osservazioni: $X = (x_1, x_2, \ldots, x_n)$

La **media aritmetica** è un indice di posizione e si calcola:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

È la somma delle n osservazioni che compongono la variabile X.

La **varianza** è un indice di variabilità e misura la lontananza dei dati dalla media. In sostanza è la media degli scarti al quadrato e si indica con:

$$var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Molto spesso però, per campioni di piccole dimensioni la varianza campionaria risulta essere uno stimare distorto. Per questo motivo molto spesso si preferisce utilizzare la varianza campionaria corretta. Si definisce corretta proprio perché è calcolata moltiplicando la varianza per il fattore di correzione, $\frac{n}{n-1}$, e la formula è la seguente:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

L'**intervallo di confidenza** è un intervallo calcolato attorno alla stima puntuale. La stima puntuale consiste nel calcolo di un singolo valore che meglio approssima il parametro sconosciuto (ad esempio per stimare μ si utilizza la media campionaria). La stima puntuale però può differire dal "vero" valore del parametro incognito quindi per ovviare a questo problema si calcola un intervallo di confidenza entro il quale risiede con alta probabilità il vero valore del parametro ignoto. L'intervallo di confidenza per la media di una popolazione avente varianza ignota viene calcolato nel seguente modo:

$$IC = \left[\bar{x} - t_{n-1,\alpha/2} \cdot \sqrt{\frac{\hat{\sigma}^2}{n}} \;,\; \bar{x} + t_{n-1,\alpha/2} \cdot \sqrt{\frac{\hat{\sigma}^2}{n}}\right]$$

Una **verifica d'ipotesi** statistica ha lo scopo di fornire gli strumenti e i metodi per stabilire, sulla base dei risultati osservati, se l'ipotesi formulata può essere accettata o meno.

L'ipotesi nulla, indicata con H_0, è l'ipotesi che si vuole sottoporre a verifica. Nel mio caso:

H_0: $\mu_1 - \mu_2 = 0$ oppure H_0: $\mu_1 = \mu_2$

In quest'ipotesi μ_1 indica la media dei tempi osservati in presenza della stored procedure mentre μ_2 indica la media dei tempi osservati in assenza. Stiamo supponendo che le due medie siano uguali e quindi che la loro differenza sia nulla.

Ogni ipotesi nulla è contrapposta con un'ipotesi alternativa ed è indicata con H_1.

Esistono tre tipi di ipotesi alternative:

- H_1: $\mu_1 - \mu_2 \neq 0$
- H_1: $\mu_1 < \mu_2$
- H_1: $\mu_1 > \mu_2$

Nel mio caso l'ipotesi alternativa che interessa saggiare è H_1: $\mu_1 > \mu_2$, ossia che il tempo medio rilevato in presenza della stored procedure sia maggiore di quello rilevato in assenza di essa.

Fissata l'ipotesi nulla ed alternativa si deve definire una regola che permetta di decidere se accettare o rifiutare l'ipotesi nulla. Questa regola viene chiamata test.

Il test **"t di Student"** è un test parametrico che viene utilizzato quando la varianza di una popolazione è ignota. Per essere adoperato devono essere rispettati, come già detto in precedenza, i seguenti assunti di base:

- Normalità delle popolazioni,
- Omoschedasticità delle popolazioni, ossia uguaglianza delle varianze,
- Indipendenza delle osservazioni tra i gruppi e entro i gruppi.

Se tutti questi assunti sono rispettati si può calcolare il valore osservato della statistica test:

$$t^{oss} = \frac{(\overline{x_1} - \overline{x_2}) - (\mu_1 - \mu_2)}{S\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

con $S^2 = \frac{(n_1-1)S_1^2+(n_2-1)S_2^2}{n_1+n_2-2}$.

Questa statistica si distribuisce come una t con $n_1 + n_2 – 2$ gradi di libertà. In questa formula è stata utilizzata la varianza pooled, ossia il rapporto tra la somma delle sue devianze e i gradi di libertà.

Nelle osservazioni da me rilevate l'assunto di omoschedasticità non è rispettato, quindi ho dovuto utilizzare il test **"t di Welch"**. Questo è abbastanza simile alla "t di Student" e differisce solo per il calcolo dei gradi di libertà, in quanto questi vengono calcolati tenendo conto della non omogeneità delle varianze.

La statistica test, come per la "t di Student", è la seguente:

$$t^{oss} = \frac{(\overline{x_1} - \overline{x_2}) - (\mu_1 - \mu_2)}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

Essa si distribuisce ancora come una t e l'unica differenza sta nel fatto che i gradi di libertà non sono più $n_1 + n_2 – 2$ ma vengono calcolati con l'equazione di Welch-Satterthwaite:

$$df = \frac{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)^2}{\frac{S_1^4}{n_1^2 \cdot (n_1 - 1)} + \frac{S_2^4}{n_2^2 \cdot (n_2 - 1)}}$$

Dopo aver determinato la statistica test da utilizzare si può calcolare il valore osservato, t^{oss}, e decidere se accettare o rifiutare l'ipotesi nulla. Nel mio caso ho ipotizzato che i tempi medi in presenza e in assenza della stored procedure siano uguali, contro un'ipotesi alternativa che i tempi medi in presenza siano maggiori di quelli in assenza, diminuendo così le prestazioni dell'applicazione. Per un test di questo tipo l'ipotesi nulla viene rifiutata se $t^{oss} \geq t_{\alpha,n_1+n_2-2}$.

L'ipotesi di uguaglianza dei tempi medi viene accettata se il valore osservato sarà più piccolo del quantile calcolato ad un livello di significatività α di una t con $n_1 + n_2 – 2$ gradi di libertà.

In definitiva il sistema d'ipotesi che ho verificato è il seguente:

$$\begin{cases} H_0: \mu_1 - \mu_2 = 0 \\ H_1: \mu_1 - \mu_2 > 0 \end{cases}$$

Dato che i miei campioni erano tutti composti da trenta osservazioni risulta: $n_1 = n_2 = 30$ perciò, utilizzando un livello di significativitàα = 0.05, l'ipotesi nulla viene rifiutata se $t^{oss} \geq t_{0.05\,,58}$.

6.5 Risultati ottenuti

Riporto qui di seguito i valori stimati di media e varianza corretta e l'intervallo di confidenza.

- Tabella **Art**

Inserimento	*Con "SPOperation"*	*Senza "SPOperation"*
MEDIA	0,017433333	0,016633333
VARIANZA CORRETTA	1,40471E-05	4,99885E-06
I.C.	[0,01603384-0,0188328]	[0,0157984-0,01746818]

Modifica	*Con "InsOperation"*	*Senza "InsOperation"*
MEDIA	0,016766667	0,0167
VARIANZA CORRETTA	1,6392E-05	8,01034E-06
I.C.	[0,0152548-0,01827845]	[0,01564318-0,0177568]

Cancellazione	*Con "SPOperation"*	*Senza "SPOperation"*
MEDIA	9,018066667	1,497866667
VARIANZA CORRETTA	0,403747306	0,021519361
I.C.	[8,7808037-9,25532964]	[1,44309073-1,5526426]

- Tabella **CliFor**

Inserimento	*Con "SPOperation"*	*Senza "SPOperation"*
MEDIA	0,066	0,066233333
VARIANZA CORRETTA	9,7931E-06	1,01851E-05
I.C.	[0,06483148-0,0671685]	[0,06504166-0,0674250]

Modifica	*Con "InsOperation"*	*Senza "InsOperation"*
MEDIA	0,068366667	0,068
VARIANZA CORRETTA	4,03333E-06	6,897E-06
I.C.	[0,0676168-0,06911657]	[0,06701940-0,0689806]

Cancellazione	*Con "SPOperation"*	*Senza "SPOperation"*
MEDIA	8,1883	1,730466667
VARIANZA CORRETTA	0,349476217	0,018522464
I.C.	[7,9675585-8,40904149]	[1,6796479-1,78128548]

- Tabella **DocTes**

Cancellazione	*Con "SPOperation"*	*Senza "SPOperation"*
MEDIA	40,53366667	3,765533333
VARIANZA	1,16955881	0,116977267
I.C.	[39,962581-41,1047522]	[3,5849237-3,94614296]

Osservando gli intervalli di confidenza è facile notare che molto probabilmente l'aggiunta della stored procedure, almeno in cancellazione, ha provocato un aumento dei tempi di risposta dell'applicazione stessa. Per quanto riguarda le operazioni di modifica e inserimento non sembrano invece esserci sostanziali differenze. Per controllare l'esattezza delle considerazioni appena fatte è sufficiente calcolare le statistiche test "t di Welch" e i relativi gradi di libertà come spiegato in precedenza. I risultati ottenuti sono i seguenti:

TABELLA	EVENTO	t^{oss}	df	p-value
ART	*INSERT*	1.004	47.32	0.1602
	UPDATE	0.0739	51.879	0.4707
	DELETE	63.1625	32.083	< 2.2e-16
CLIFOR	*INSERT*	-0.2859	57.978	0.612
	UPDATE	0.6075	54.275	0.2730
	DELETE	58.3075	32.065	< 2.2e-16
DOCTES	*DELETE*	125.547	16.773	< 2.2e-16

La tabella conferma le considerazioni fatte in precedenza: l'inserimento della stored procedure ha comportato un aumento statisticamente significativo dei tempi di risposta dell'applicazione per le operazioni di cancellazione, mentre non ha diminuito la velocità dell'applicazione per le operazioni di inserimento e modifica.

Si nota infatti, osservando gli intervalli di confidenza sopra inseriti, che per le tabelle CliFor e Art l'aumento è stato abbastanza contenuto mentre per DocTes è risultato più evidente. Ciò è dovuto, come già detto, al grande numero di cancellazioni a cascata imposte dal trigger. Tutti gli aumenti risultano comunque inaccettabili soprattutto per le aziende che non sono interessate a sfruttare questa nuova funzionalità. Per ovviare a questo problema ho consigliato al tutor aziendale di inserire tra le impostazioni avanzate del programma la possibilità di abilitare o disabilitare a piacimento la funzionalità. Questo ha risolto però solo una parte del problema, ossia ha permesso di garantire la non perdita di efficienza dell'applicativo a coloro che non desiderano utilizzare "SPOperation". Per quanto riguarda invece i futuri utilizzatori è necessario considerare che i test sono stati eseguiti su un calcolatore abbastanza datato e quindi non molto potente. La potenza di calcolo influisce molto sulla velocità di risposta dell'applicativo quindi un calcolatore più recente otterrà prestazioni migliori di quelle da me rilevate.

7. Conclusioni

Giunti alla fine di questa relazione è arrivato il momento di tirare le conclusioni riguardo questa esperienza di stage.

Durante il mio percorso mi sono scontrata con due grandi mondi: l'informatica e la statistica. Per quanto riguarda l'informatica, mi sono imbattuta nella gestione e amministrazione di una base di dati. Ciò non è stato un lavoro semplice e immediato. Le prime settimane di stage infatti le ho vissute in uno stato di gran confusione poi però grazie alle conoscenze di base, apprese durante le lezioni all'università, all'aiuto del tutor aziendale e con un po' di documentazione personale sono riuscita a concludere con buon esito, a mio parere, il mio progetto. Il mondo della statistica in questo caso è intervenuto come supporto per l'informatica. La statistica, come già spiegato, è stata utile per verificare se il mio progetto poteva creare danni all'efficienza dell'applicazione. Durante tutto lo stage ho quindi potuto, come speravo, vedere applicati al mondo lavorativo moltissimi concetti teorici imparati all'università. Tutto ciò che ho imparato negli studi però non è stato sufficiente ad evitarmi i problemi. Questo perché ogni problema ha una propria soluzione e per trovare la soluzione bisogna trovarsi davanti al problema.

La mia esperienza di stage è stata molto positiva anche per quanto riguarda l'ambiente lavorativo. Ho potuto infatti svolgere il mio lavoro in maniera autonoma senza essere aiutata continuamente da qualcuno. Per tutti i problemi incontrati, infatti, prima di chiedere aiuto al team di programmazione o al tutor aziendale, sempre disponibili entrambi, cercavo di una soluzione da sola, attraverso le librerie elettroniche.

In definitiva credo che l'esperienza fatta presso la "Vision Software Gestionale" sia stata complessivamente buona non solo per conoscere il mondo lavorativo ma anche come arricchimento personale. Spero infatti di poter applicare e perfezionare le nuove conoscenze acquisite e le nuove abilità lavorative in un futuro contesto lavorativo.

8. Bibliografia e sitografia

- Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone
  BASI DI DATI – Modelli e linguaggi di interrogazione, McGraw-Hill
- INFERENZA STATISTICA I – dispensa delle lezioni, Prof.ssa A. Brogini
- STATISTICA DESCRITTIVA – dispensa delle lezioni, Guido Masarotto
- Domenico Piccolo
  STATISTICA PER LE DECISIONI – La conoscenza umana sostenuta dall'evidenza empirica, Il Mulino
- http://technet.microsoft.com/it-it/library/ms203721(SQL.90).aspx
  Documentazione in linea di SQL Server
- http://msdn.microsoft.com/en-us/library/724fd5h9.aspx
  Documentazione in linea di Microsoft Visual FoxPro
- http://en.wikipedia.org/wiki/Welch's_t_test
  Formula del test "t di Welch"
- http://www.vsh.it/
  Sito dell'azienda "Vision Software Gestionale"