Capitolo 1

Introduzione

Questo capitolo presenta un'introduzione al lavoro di tesi. In particolare, verrà prima descritta l'azienda presso la quale è stato svolto il lavoro di tesi e poi si illustreranno gli obiettivi e i risultati attesi.

1.1 Le Società I.T.I.A s.r.l e F.R. Trattamenti Galvanici

Il programma sviluppato appartiene a un progetto più complesso, tuttora in lavorazione, effettuato dalla società ITIA s.r.l. come applicazione enterprise per la gestione amministrativa di una società di trattamenti galvanici, la F.R..
Dall'introduzione dei PC in azienda, avvenuta nei primi anni '90, la società F.R. è sempre stata in grado di mettere a disposizione del proprio personale le risorse hardware informatiche più adatte alle proprie esigenze lavorative; tuttavia, il buon "livello" hardware non è mai stato affiancato da un valido supporto software, eccezione fatta per il settore contabile.
Anche nella realtà passata dell'azienda l'obiettivo principale è sempre stato quello di soddisfare le esigenze qualitative dei propri clienti. In quest'ottica l'attenzione è stata rivolta maggiormente nei reparti produzione/qualità. Il graduale aumento commerciale ha condotto alla creazione di ulteriori settori interni sempre più specializzati e collaboranti ed il relativo aumento del personale.
Le esigenze dirigenziali hanno richiesto alla produzione flussi di dati sempre più precisi, real time e dettagliati, mentre dal settore commerciale, qualitativo e amministrativo sono aumentate le richieste di informazioni riepilogative, statistiche e previsionali.
Il continuo aumento di tali richieste, la decentralizzata struttura informatica, la mancanza di una strategia risolutiva di immediato effetto, la non adozione di strumenti per l'office automation, l'impossibilità di rilevazioni in tempo reale della produzione, la scarsa comunicatività dei singoli settori ed altri fattori di minor rilievo, hanno portato ad un forte disagio logistico l'intera azienda.
Dai primi anni del 2000 si è affermata la volontà di risolvere le problematiche precedentemente accennate; gli investimenti in tale direzione sono stati di entità rilevante ma purtroppo non sufficienti.
La soluzione a queste problematiche è stata appunto quella di investire verso una soluzione di tipo enterprise, costruita su misura, evitando l'adattamento di qualche prodotto in commercio "prefabbricato".
Il primo passo è stato quello di incaricare una società (l'I.T.I.A.) che lavorasse all'interno dell' F.R. e che provvedesse integralmente alla realizzazione del progetto; in secondo luogo si è reso necessario ampliare la struttura hardware e provvedere all'addestramento graduale del personale addetto; quindi sono stati installati dei programmi pilota al fine di valutare prestazioni e potenzialità della soluzione scelta.
In data odierna il progetto è in corso di sviluppo ed è orientato alla risoluzione tempestiva delle problematiche relative al processo di lavorazione.

1.2 Obiettivi del lavoro di tesi

La possibilità di lavorare a stretto contatto con i futuri utenti, sebbene sia una situazione non comune, ha permesso un dialogo continuo e proficuo con gli utenti e la realizzazione di un prodotto confezionato su misura.Oltre alle indispensabili specifiche tecniche fornite dal cliente questo stretto contatto ha permesso di capire quali sono le esigenze pratiche dell'utente e quindi raffinare sempre di più il lavoro. Si è così arrivati a diverse release sempre più rispondenti alle reali necessità.
Il progetto nella sua interezza si propone di gestire la parte amministrativa della società che ha commissionato il lavoro e permette di seguire ogni processo lavorativo in tutto il suo svolgersi.
Entrando più nello specifico la richiesta era di poter seguire le fasi lavorative di ogni singolo cliente dalla ricezione dell'ordine fino alla consegna del prodotto lavorato. Per fare questo è stato introdotto il concetto di "cartellino" ovvero un'etichetta (sia digitale che fisica) da associare ad ogni prodotto da lavorare in cui sono annotate tutte le informazioni necessarie. Così facendo è possibile sapere in ogni istante in che fase si trova ogni prodotto in lavorazione, quali processi sta subendo, chi è l'operatore responsabile, il tempo rimanente e altre informazioni. La necessità di avere sotto controllo in maniera così precisa ogni fase lavorativa nasce da una duplice esigenza: da un lato poter sapere con esattezza i costi di ogni singola lavorazione e quindi poter intervenire per migliorare e ottimizzare laddove ciò si renda necessario e dall'altro la possibilità di organizzare il lavoro con estrema precisione in modo da limitare al minimo possibile i fermi macchina delle linee lavorative e poter dare tempi certi di consegna al cliente.
Tramite questo applicativo è possibile monitare non solo le lavorazioni in corso ma anche di programmare con un certo anticipo quelle future in modo da sfruttare al meglio le risorse umane e tecnologiche e poter già nella compilazione del preventivo del cliente sapere il giorno e l'ora in cui la lavorazione verrà eseguita, potendo quindi avere una stima dei costi e della data di ultimazione.
La parte sviluppata da me in questo progetto riguarda la gestione del magazzino dei fornitori di materie prime necessarie per le fasi lavorative. L'impostazione che è stata data al progetto ruota ovviamente attorno al concetto di articolo e alle operazioni che è possibile effettuare su di esso: definizione, cancellazione, modifica nonchè le attività di carico e scarico**.**

1.2 Organizzazione della tesi

La tesi è organizzata come segue. Nel capitolo 2 verranno illustrati i concetti introduttivi alla base di dati utilizzata (Microsoft SQL-Server 7.0) , al linguaggio di programmazione (Microsoft Visual Basic 6.0) e alle Stored Procedure. Il capitolo 3 illustra la progettazione e realizzazione della base di dati oggetto di questo lavoro di tesi e l'impementazione vera e propria del programma applicativo.

Capitolo 2

Concetti di base

In questo capitolo verranno illustrati i concetti che sono alla base di questo lavoro di tesi.

2.1 - Microsoft SQL-Server 7.1

2.1.1 Architettura

L'intero programma si avvale dell'uso di Microsoft SQL-Server 7.0 [3],[7], quindi anche per la gestione del magazzino è stata utilizzata questa specifica base di dati. Per l'interfacciamento con la base di dati si è utilizzato Microsoft ADO, tecnologia D-com che permette un dialogo ad alto livello. Vediamo più nel dettaglio il funzionamento di SQL-Server 7.0

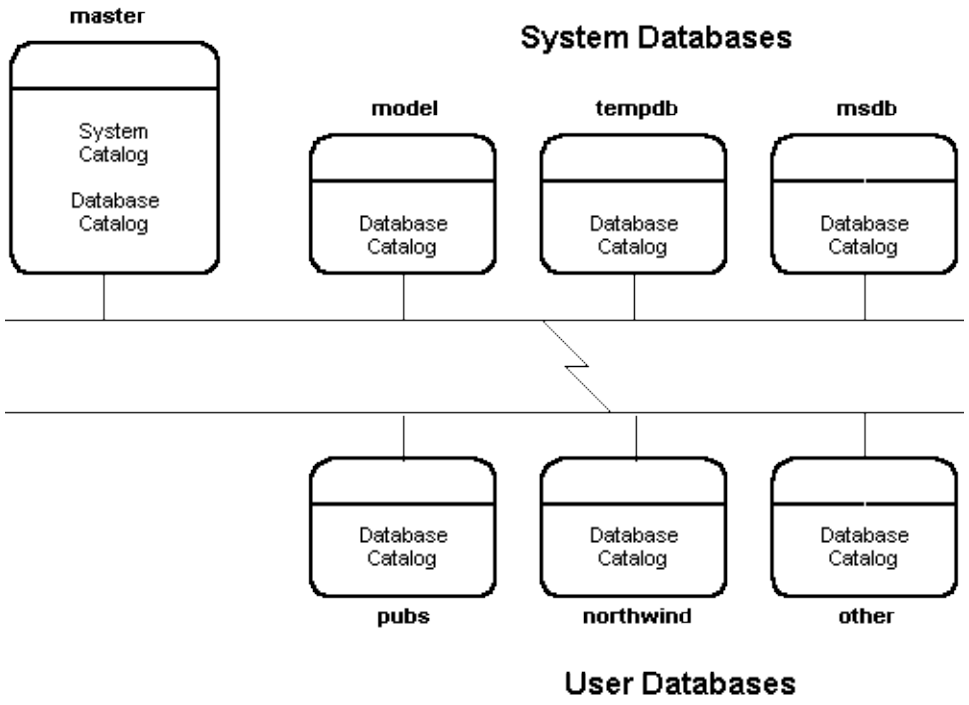


***Figura 2.1.1**: Schema di SQL-Server 7.0*

Il database è la struttura principale di SQL Server e fornisce l'ambiente per archiviare e controllare i dati. Come rappresentato in Figura 2.1.1, SQL Server ha due grosse categorie di databases:

1. Databases di sistema
2. Databases utente

All' interno dei databases di sistema SQL Server memorizza tutte le informazioni e gli oggetti necessari al suo funzionamento.
I databases di sistema che SQL Server crea al momento dell'installazione sono quattro:

1. MASTER contiene le informazioni sul server ad alto livello
2. TEMPDB contiene le tabelle e gli oggetti temporanei
3. MODEL contiene il modello per la creazione di un database tipo
4. MSDB informazioni per il funzionamento di SQL Server Agent (Jobs, Web Assistant, ecc..)

Nei databases utente invece sono memorizzati i dati utente e gli oggetti del database che verranno illustrati successivamente. Il numero di databases di questo tipo che è possibile creare all'interno di SQL Server è 32734.

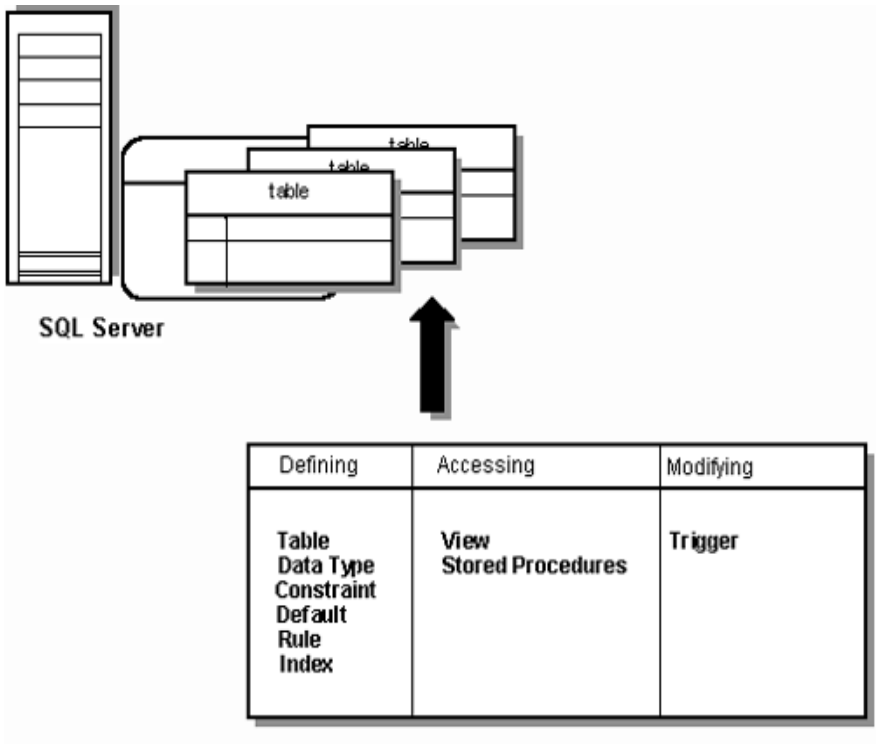


Figura 2.1.2: *Oggetti del Database*

In Figura 2.1.2 è possibile vedere gli oggetti che sono contenuti in un database:

1. **Tabelle**
   Memorizzano i dati che vengono inseriti nel database, sono tra loro in relazione reciproca. Sono fatte da colonne e da righe. Una tabella può avere fino a 1024 colonne e 8092 byte per riga.
   Il numero di tabelle per database può arrivare fino a due miliardi.

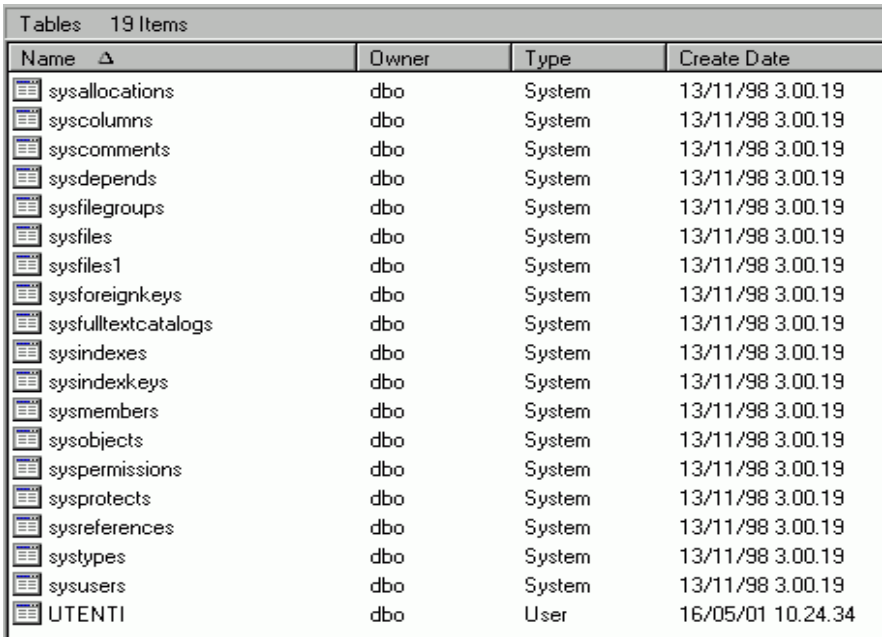
2. **Tipi di dati**
   Definiscono i tipi di dati che possono essere inseriti nelle colonne, possono essere definiti dall'utente.
3. **Obblighi**
   Servono a rafforzare l'integrità del database.
4. **Default**
   Assegna valori predefiniti ad una determinata colonna.
5. **Regole**
   Definiscono vincoli ai dati che vengono inseriti, servono a rafforzare l'integrità del database.
6. **Indici**
   Servono ad ottimizzare l'accesso ai dati contenuti nelle tabelle.
7. **Viste**
   Sono tabelle generate con colonne prese da una o più tabelle.
8. **Procedure Memorizzate (Stored Procedure)**
   Sono set di istruzioni T-SQL, sono dei veri e propri programmi per i databases.
9. **Trigger**
   Sono procedure memorizzate che si attivano in modo autonomo in base allo scatenarsi di determinati eventi come INSERT, UPDATE, DELETE.

L'insieme degli **o**ggetti di database che costituiscono il database prende il nome di SCHEMA. La progettazione di tutti questi oggetti rappresenta il modello di dati.
Tutti questi oggetti possono essere creati in tre modi differenti:

1. Attraverso le procedure guidate di SQL Server 7.0 (wizard)
2. Con istruzioni SQL (CREATE TABLE, CREATE PROCCEDURE, ecc..)
3. Attraverso l'Enterprise Manager Console

SQL Server tiene traccia all'interno di ogni database degli oggetti che vengono creati, modificati o eliminati.
Le informazioni che descrivono gli oggetti del database sono chiamate metadati.

I metadati sono organizzati in un dizionario dei dati che contiene istruzioni come CREATE o ALTER.
Il dizionario dei dati è organizzato in tabelle di sistema (iniziano con il prefisso sys) che sono contenute all'interno di ogni databases utente che viene creato all'interno di SQL Server.



Tables 19 Items

Name	Owner	Type	Create Date
sysallocations	dbo	System	13/11/98 3.00.19
syscolumns	dbo	System	13/11/98 3.00.19
syscomments	dbo	System	13/11/98 3.00.19
sysdepends	dbo	System	13/11/98 3.00.19
sysfilegroups	dbo	System	13/11/98 3.00.19
sysfiles	dbo	System	13/11/98 3.00.19
sysfiles1	dbo	System	13/11/98 3.00.19
sysforeignkeys	dbo	System	13/11/98 3.00.19
sysfulltextcatalogs	dbo	System	13/11/98 3.00.19
sysindexes	dbo	System	13/11/98 3.00.19
sysindexkeys	dbo	System	13/11/98 3.00.19
sysmembers	dbo	System	13/11/98 3.00.19
sysobjects	dbo	System	13/11/98 3.00.19
syspermissions	dbo	System	13/11/98 3.00.19
sysprotects	dbo	System	13/11/98 3.00.19
sysreferences	dbo	System	13/11/98 3.00.19
systypes	dbo	System	13/11/98 3.00.19
sysusers	dbo	System	13/11/98 3.00.19
UTENTI	dbo	User	16/05/01 10.24.34

***Figura 2.1**: Tabelle di un Database*

Nella Figura 2.1 possiamo vedere un esempio di screenshot delle tabelle contenute in un database.

2.1.2 Sicurezza

In un database multi-utente la sicurezza è importante. In SQL Server la sicurezza è implementata su due livelli:

1. a livello del server (login)
2. a livello del database (user)

La login è composta da uno User ID e da una password. Per ogni login all'interno di SQL Server deve esserci un corrispondente utente all'interno di uno specifico database. L'utente deve avere i privilegi opportuni per poter compiere le varie operazioni all'interno del database.

Server login ID

Una server login valida è necessaria per accedere ad SQL Server, il login account è fatto di 3 componenti:
SERVER LOGIN ID (utente_xxx)
SERVER PASSWORD (psw)
DEFAULT DATABASE

Esiste una login molto particolare che viene creata per default da SQL Server al momento dell'installazione: il system administrator (sa) è abilitato a creare database ed utenti, a fare backup, ecc.. insomma ad amministrare il sistema e così via.
Una volta creato un database il sa crea gli utenti che potranno accedere al database, assegnando i privilegi ed i ruoli opportuni. Il creatore del database ne diventa proprietario cioè: database owner(dbo).

Ogni oggetto nel database ha un owner.

2.1.3 Strumenti

SQL Server è un prodotto molto completo e fornisce una vasta gamma di strumenti (grafici e non) utili a monitorare, amministrare e interrogare SQL Server.
Ecco un elenco dei tool:

1. Amministrazione:

- SQL Enterprise Manager

2. Interrogazione:

- Utility da linea di comando isql
- SQL Query Analyzer
- Microsoft Query
- SQL Server WWWeb Assistant
- Microsoft Access

3. Monitoraggio:

- SQL Performance Monitor
- SQL Trace

4. Documentazione:

- SQL Server Book Online
- Help vari (ODBC e altro)

5. Configurazione:

- Client configuration utility
- Server Network utility

6. Altri:

- SQL Server Service Manager

2.1.4 Connessione

Ci sono vari modi di parlare con SQL Server dall'esterno:

1. Attraverso utility client come SQL Query Analyzer
2. Altri database Microsoft: ACCESS

3. Linguaggi di programmazione: Visual Basic
4. Da Internet attraverso le pagine ASP

Attraverso uno di questi metodi è possibile collegarsi ed inviare dei comandi T-SQL a SQL Server.

2.1.5 I Tipi di dato in SQL Server

In SQL-SERVER quando viene creata una tabella si deve definire in modo esatto il tipo di dati che ogni colonna può contenere. SQL Server permette di definire vari tipi di dati utili per immagazzinare informazioni: caratteri, numeri, bytes, date, immagini e oltre a questo è possibile definire tipi di dati personalizzati secondo esigenze specifiche.
Ecco l'elenco e la descrizione dei tipi di dati disponibili con SQL Server 7.0:

Dati binari:

- binary[(n)]
  ha una lunghezza fissa e può contenere fino ad 8000 bytes di dati binari
- varbinary[(n)]
  ha una lunghezza variabile e può contenere fino ad 8000 bytes di dati binari

Dati carattere:

- char[(n)]
  ha una lunghezza fissa e può contenere fino ad 8000 caratteri ANSI (cioè 8000 bytes)
- varchar[(n)]
  ha una lunghezza variabile e può contenere fino ad 8000 caratteri ANSI (cioè 8000 bytes)
- nchar[(n)]
  ha una lunghezza fissa e può contenere fino a 4000 caratteri UNICODE (cioè 8000 bytes, ricordiamo che per i caratteri UNICODE servono 2 bytes per memorizzare un carattere)
- nvarchar[(n)]
  ha una lunghezza variabile e può contenere fino a 4000 caratteri UNICODE (cioè 8000 bytes, ricordiamo che per i caratteri UNICODE servono 2 bytes per memorizzare un carattere)

Dati ora e data:

- datetime
  ammette valori compresi dal 1 gennaio 1753 al 31 dicembre 9999 (precisione al trecentesimo di secondo), occupa uno spazio di 8 byte
- smalldatetime
  è meno preciso del precedente (precisione al minuto), ,occupa uno spazio di 4 byte

Dati monetari:

- money
  Contiene valori monetari da -922337203685477.5808 a 922337203685477.5807 con una precisione al decimillesimo di unità monetaria, occupa 8 bytes di memoria

- smallmoney
  Contiene valori monetari da - 214748.3648 a 214748.3647 con una precisione al decimillesimo di unità monetaria, occupa 4 bytes di memoria.

Dati numerici approssimati:

- float[(n)]
  Contiene numeri a virgola mobile positivi e negativi, compresi tra
  2.23E-308 e 1.79E308 per i valori positivi e tra -2.23E-308 e -1.79E308 per i valori negativi, occupa 8 bytes di memoria ed ha una precisione di 15 cifre
- real
  Contiene numeri a virgola mobile positivi e negativi comprese tra 1.18E-38 e 3.40E38 per i valori positivi e tra -1.18E-38 e -3.40E38 per i valori negativi, occupa 4 bytes di memoria ed ha una precisione di 7 cifre

Dati numerici esatti:

- decimal[(p[, s])]
- numeric[(p[, s])]
  decimal e numeric sono sinonimi per SQL Server 7.0, possono avere valori compresi tra 10^38 -1 e - 10^38 -1. La memoria che occupano per essere immagazzinati varia a seconda della precisione che utilizziamo per rappresentarli, da un minimo di 2 bytes a un massimo di 17 bytes
  p - è la precisione, che rappresenta il numero massimo di cifre decimali che possono essere memorizzate (da entrambe le parti della virgola). Il massimo della precisione è 28 cifre.
  s - è la scala, che rappresenta il numero massimo di cifre decimali dopo la virgola e deve essere minore od uguale alla precisione.
- int
  occupa 4 byte di memoria e memorizza i valori da -2147483648 a 2147483647
- smallint
  occupa 2 byte di memoria e memorizza i valori da -32768 a 32,767
- tinyint
  occupa 1 byte di memoria e memorizza i valori da 0 a 255

Dati speciali:

- bit
  tipicamente è usato per rappresentare i flag, vero/false o true/false o si/no, perché può accettare solo due valori 0 o 1. Occupa un bit ovviamente. Le colonne che hanno un tipo dati bit non possono avere valori nulli e non possono avere indici.
- cursor
  sono usati come varibili in stored procedure oppure come parametri di OUTPUT sempre in stored proc, fanno riferimento ai cursori. Possono essere nulli e non possono essere usati con le istruzioni CREATE TABLE.
- sysname
  è un varchar di 128 caratteri ed occupa 256 bytes, viene usato per assegnare i nomi ad oggetti

del database, come tabelle, procedure, trigger, indici e altro.

- timestamp
  occupa 8 bytes ed è un contatore incrementale per colonna assegnato automaticamente da SQL-Server.
- UNIQUEIDENTIFIER (GUID)
  E' un identificatore unico a livello globale di 16 byte di lunghezza chiamato anche GUID. È generato (molto lentamente) automaticamente da SQL Server.

Dati text ed image:

I dati di questo tipo, non vengono memorizzati nelle normali pagine dati di SQL Server, ma sono trattati in modo speciale su apposite pagine di memorizzazione.

- text
  è un tipo dati a lunghezza variabile, che può memorizzare fino a 2147483647 caratteri.
- ntext
  come il precedente ma memorizza caratteri UNICODE, quindi fino alla metà del precedente, cioè 1073741823 caratteri.
- image
  può memorizzare fino a 2147483647 bytes di dati binari, è solitamente usato per le immagini.

Sinonimi per i tipi di dati

Per assicurare la compatibilità con lo standard SQL-92, SQL Server può usare i seguenti sinonimi per i corrispondenti tipi di dati quando vengono usate istruzioni che fanno parte del data definition language (DDL), come CREATE TABLE, CREATE PROCEDURE o DECLARE @nomevariable (Tabella 2.1).

Sinonimo	Mappato su SQL Server 7.0
Binary varying	Varbinary
char varying	Varchar
character	Char
character	char(1)
character(n)	char(n)
character varying(n)	varchar(n)
Dec	decimal
Double precision	float
float[(n)] for n = 1-7	real
float[(n)] for n = 8-15	float
Integer	Int
national character(n)	nchar(n)
national char(n)	nchar(n)
national character varying(n)	nvarchar(n)
national char varying(n)	nvarchar(n)
national text	ntext

Tabella 2.1 Mappaggio dei tipi di dato

La conversione dei tipi di dato in altri linguaggi

T-SQL non mappa i tipi di dati con quelli di altri linguaggi di programmazione. Nella Tabella 2.2 sono indicate le corrispondenze tra i tipi di dati di SQL Server e quelli di Visual Basic**.** Questo problema ha richiesto particolare attenzione durante la realizzazione dell'applicazione oggetto del presente lavoro di tesi, onde evitare l'uso di tipi troppo piccoli (sia in Visual Basic che in SQL-Server 7.0) e quindi cadere nel rischio di overflow o underflow. In particolare i controlli necessari nella conversione di tipo sono stati fatti via software.

Convertire I VB Data Types in SQL Server Data Types	
Tipi di dati in VB	**Tipi di dati corrispondenti in SQL Server**
Long, Integer, Byte, Boolean	T-SQL Int
Double, Single	T-SQL Float
Currency	T-SQL Money
Date	T-SQL DateTime
Stringhe fino a 255 caratteri	T-SQL VarChar
Stringhe superiori a 255 caratteri	T-SQL Text
Array monodimensionale fino a 255 elementi	T-SQL VarBinary
Array monodimensionale con più di 255 elementi	T-SQL Image

Tabella 2.2 Conversione dei tipi di dato

2.2 - Microsoft Visual Basic 6.0

La nascita di Microsoft Visual Basic [1],[2],[7] ha comportato una rivoluzione nel mondo degli strumenti di sviluppo per Windows, lanciando un nuovo modo di programmare, che vede per la prima volta prevalere l'uso del mouse nei confronti di quello della tastiera.
Tale approccio, detto visuale, si basa sull'uso di oggetti, ovvero di elementi attivi selezionabili su una barra degli strumenti (toolbox), caratterizzati dall'essere descritti da un insieme di valori, detti proprietà ed in grado di eseguire dei metodi, ovvero delle azioni che ne possono influenzare lo stato.
Un'ulteriore particolarità degli oggetti consiste nella capacità di generare degli eventi, a cui il programmatore può associare come risposta degli insiemi di istruzioni, detti procedure (ad esempio quando viene premuto un bottone il programmatore può associare la chiusura di una finestra).
Il linguaggio con cui esse sono scritte si basa sulle regole sintattiche previste dal BASIC, di cui mantiene la semplicità.
Un'applicazione Visual Basic è costituita da una o più finestre, dette *form*, sulle quali si trovano gli oggetti che formano l'interfaccia fra l'applicazione e l'utente. Su un form è possibile inserire pressoché qualsiasi elemento, sia che si tratti di un'immagine, di un box di testo, di una lista o di un pulsante.

2.2.1 Gli eventi

Ogni oggetto inserito all'interno di un form ha associato degli eventi, come per esempio il click del mouse, ed è possibile gestirli tramite delle routine apposite.
Volendo fare in modo che la risposta avvenga in occasione del click del mouse su di un pulsante (ad esempio “btScrivi”) occorre selezionare nella lista degli eventi disponibili l'evento Click. Si provoca così la creazione della procedura btScrivi_Click, di cui è automaticamente visualizzata l'intestazione nella parte inferiore della finestra, seguita dalla frase End Sub, che indica la fine del blocco di codice.
Le istruzioni che devono essere eseguite in risposta all'evento vanno inserite nello spazio compreso fra le righe generate automaticamente, avendo l'accortezza di indicare un solo comando per linea.

2.2.2 I metodi

Come si è già accennato, ogni oggetto è in grado di ricevere dei comandi che possono provocare la variazione di alcune proprietà o la generazione di eventi: i metodi. L'invocazione di un metodo avviene secondo la sintassi:

<oggetto>.<metodo> [<parametro>, ... ,<parametro>]

Al nome dell'oggetto è necessario far seguire, separato da un punto, quello del metodo e gli eventuali parametri.
Ad esempio, l'aggiunta della riga

btScrivi.Move 0,0

nella procedura vista in precedenza, provoca l'esecuzione del metodo Move da parte dall'oggetto btScrivi. La coppia 0, 0 rappresenta l'insieme dei parametri.
L'effetto che si ottiene consiste nello spostamento del bottone nell'angolo superiore sinistro del form, ovvero nel punto di cui i parametri rappresentano le coordinate.

2.2.3 Le variabili

Anche Visual Basic, come tutti i linguaggi di programmazione, prevede l'uso delle variabili, mediante le quali è possibile memorizzare dei valori testuali o numerici in strutture a cui il programma può accedere grazie a un nome assegnato loro in fase di creazione.
Una variabile è detta *locale* quando è definita all'interno di una procedura e la sua creazione avviene quando si fa riferimento ad essa per la prima volta, oppure quando è eseguita l'istruzione Dim, che presenta la seguente sintassi:

Dim <nome> [As <tipo>]

in cui <nome> rappresenta il nome da assegnare alla variabile.
Esso non si sottrae alla regola valida per tutti gli identificatori, ivi compresi i nomi degli oggetti, che impone loro di essere costituiti da delle sequenze di caratteri alfabetici o numerici privi di spazi ed aventi per iniziali delle lettere dell'alfabeto.
È inoltre possibile, anche se non indispensabile (come sottolineato dalla presenza delle parentesi quadre), aggiungere all'istruzione Dim un'indicazione del tipo di dato da creare. Visual Basic prevede numerosi tipi standard di cui i più utilizzati sono integer, usato per rappresentare i dati numerici interi compresi fra -32,768 a 32,767, string, che può contenere delle sequenze (stringhe) alfanumeriche composte al massimo da circa 65500 caratteri, long, in grado di ospitare numeri interi compresi fra -2,147,483,648 e 2,147,483,647, single e double, con cui è possibile memorizzare numeri reali anche di notevole entità.
Se invece è omessa la dichiarazione del tipo, la variabile è definita variant. Questo formato non è previsto dalla maggior parte dei linguaggi di programmazione tradizionali. La sua caratteristica fondamentale è l'universalità. Una variabile variant, infatti, può contenere dei dati aventi qualsiasi formato. È buona norma, tuttavia, non fare largo uso di strutture di questo tipo, in quanto la loro gestione da parte dell'interprete è poco efficiente.
Una variabile locale può anche non essere dichiarata, in questo caso la sua creazione avviene la prima volta in cui si fa riferimento ad essa. Si può impedire la istanziazione automatica delle variabili (per ovvie ragioni di prevenzione di errori accidentali) premettendo la dichiarazione “option explicit” all'inizio del codice.
Dopo l'esecuzione dell'ultima istruzione presente nella procedura in cui sono state create, le variabili locali sono automaticamente distrutte. Per fare in modo che il loro valore sia conservato anche in occasione delle chiamate successive, è necessario dichiarare le variabili sostituendo Static alla parola chiave Dim.
In alcuni casi, si rivela necessario fare in modo che una variabile non sia mai distrutta e sia accessibile anche dalle altre procedure presenti in un form. In questo caso, è necessario selezionare la sezione General nella lista a discesa posta nella parte superiore sinistra della finestra contenente il codice associato al form, la voce Declarations nella lista di destra e dichiarare la variabile per mezzo dell'istruzione Dim.

Quando invece si presenta la necessità di fare in modo che una struttura sia accessibile da tutte le procedure presenti nell'applicazione, indipendentemente dal form in cui esse si trovano, è necessario utilizzare per la dichiarazione la parola chiave Global. Una variabile dichiarata in questo modo è detta globale. La sua dichiarazione è impossibile all'interno di un form. Essa deve essere effettuata in un modulo, ovvero in un file di testo caratterizzato dall'estensione .BAS, che è aggiunto al progetto selezionando la voce Add Module del menu Project.

2.2.4 La connesione a fonti di dati

Sfruttando ADO (ActiveX Data Object) [5],[6],[9] è possibile tramite Visual Basic connettersi direttamente ad una base di dati sfruttando le potenzialità di ADO. Buona parte della potenza e della flessibilità di ADO deriva dal fatto che consente di connettersi a numerosi provider di dati utilizzando lo stesso modello di programmazione, indipendentemente dalle caratteristiche specifiche di ciascun provider.
Poiché tuttavia ciascun provider dispone di caratteristiche uniche, le modalità di interazione tra l'applicazione e ADO variano leggermente a seconda del provider stesso. Le differenze da prendere in considerazione sono in genere riconducibili a una delle tre seguenti categorie:

- Parametri di connessione nella proprietà ConnectionString
- Utilizzo dell'oggetto Command
- Comportamento dell'oggetto Recordset specifico del provider

2.2.5 Proprietà dinamiche specifiche dei provider

Gli insiemi detti Properties degli oggetti Connection, Command e Recordset includono proprietà dinamiche specifiche dei diversi provider che forniscono informazioni sulle funzionalità specifiche del provider in uso oltre alle proprietà incorporate in ADO.
Dopo avere stabilito la connessione e creato tali oggetti, è possibile utilizzare il metodo Refresh dell'insieme Properties dell'oggetto per ottenere le proprietà specifiche del provider in uso.

2.2.6 L'oggetto Connection

Un oggetto Connection rappresenta una sessione univoca con una fonte dati e nel caso di un sistema di base di dati client/server, può equivalere a una effettiva connessione al server. A seconda della funzionalità supportata dal provider, è possibile che alcuni metodi, insiemi o proprietà di un oggetto Connection non siano disponibili.
Con le proprietà, gli insiemi e i metodi di un oggetto Connection, è possibile eseguire le operazioni seguenti:

- Configurare la connessione prima di aprirla utilizzando le proprietà ConnectionString, ConnectionTimeout e Mode.

- Impostare la proprietà CursorLocation per richiamare il Client Cursor Provider, che supporta gli aggiornamenti in modalità batch.
- Impostare la base di dati predefinito per la connessione utilizzando la proprietà DefaultDatabase.
- Impostare il livello di isolamento per le transazioni aperte sulla connessione utilizzando la proprietà IsolationLevel.
- Specificare un provider OLE DB con la proprietà Provider.
- Stabilire, quindi interrompere, la connessione fisica alla fonte dati utilizzando i metodi Open e Close.
- Eseguire un comando sulla connessione utilizzando il metodo Execute e configurare l'esecuzione utilizzando la proprietà CommandTimeout.
- Gestire le transazioni sulla connessione aperta, comprese quelle nidificate se supportate dal provider, utilizzando i metodi BeginTrans, CommitTrans e RollbackTrans e la proprietà Attributes.
- Esaminare gli errori restituiti dalla fonte dati utilizzando l'insieme Errors.
- Leggere la versione dall'implementazione ADO corrente utilizzando la proprietà Version.
- Ottenere le informazioni sullo schema di una base di dati utilizzando il metodo OpenSchema.

2.2.7 L'oggetto Recordset

Un oggetto Recordset rappresenta il set di record ottenuto da una tabella di base o dai risultati di un comando eseguito e viene utilizzato per manipolare i dati forniti da un provider. Per ricavare dei dati da una base di dati tramite un Recordset è necessario specificare una stringa SQL da associare al recordset stesso: il risultato della interrogazione verrà memorizzato nel Recordset. Quando si utilizza ADO, i dati vengono manipolati quasi esclusivamente tramite Recordset. Tutti i Recordset vengono creati utilizzando record (righe) e campi (colonne) e a seconda della funzionalità supportata dal provider, è possibile che alcuni metodi o proprietà non siano disponibili.
In ADO sono disponibili quattro diversi tipi di cursore:

- Cursore dinamico: consente di visualizzare aggiunte, modifiche ed eliminazioni eseguite da altri utenti e di effettuare tutti i tipi di movimento. Consente inoltre di inserire segnalibri se supportati dal provider.
- Cursore direzionale: si comporta come un cursore dinamico ad eccezione del fatto che non consente di visualizzare record aggiunti da altri utenti e di accedere a record cancellati da altri utenti. I dati modificati da altri utenti saranno ancora visibili. Supporta segnalibri e consente di effettuare tutti i tipi di movimento.
- Cursore statico: fornisce una copia statica di un set di record da utilizzare per trovare dati o generare relazioni; consente di inserire segnalibri e di effettuare tutti i tipi di movimento tramite il Recordset. Aggiunte, modifiche o eliminazioni eseguite da altri utenti non saranno visibili. È l'unico tipo di cursore disponibile quando si apre un oggetto Recordset del client (ADOR)
- Cursore a scorrimento in avanti: si comporta esattamente come un cursore dinamico ad eccezione del fatto che consente di scorrere i record solo in avanti. In questo modo si migliorano le prestazioni quando è necessario effettuare un solo passaggio all'interno di un Recordset.

Alcuni provider non supportano tutti i tipi di cursore e se non viene specificato in base all'impostazione predefinita verrà aperto un cursore a scorrimento in avanti.

Quando utilizzati con alcuni provider quali ODBC Provider for OLE DB unitamente a SQL-Server, è possibile creare oggetti Recordset indipendentemente da un oggetto Connection definito in precedenza passando una stringa di connessione con il metodo Open. ADO crea sempre un oggetto Connection, senza tuttavia assegnarlo a una variabile oggetto. Se vengono tuttavia aperti più oggetti Recordset tramite la stessa connessione, è necessario creare e aprire esplicitamente un oggetto Connection, in modo da assegnarlo a una variabile oggetto. Se non si utilizza questa variabile di oggetto quando si aprono gli oggetti Recordset, ADO creerà un nuovo oggetto Connection per ciascun nuovo Recordset, anche se si passa la stessa stringa di connessione.
Quando si apre un Recordset, il record corrente corrisponde al primo record se disponibile, mentre le proprietà BOF e EOF vengono impostate a False. Se non sono disponibili record, l'impostazione per BOF e EOF è True.
È possibile utilizzare i metodi MoveFirst, MoveLast, MoveNext, MovePrevious e Move, e le proprietà AbsolutePosition, AbsolutePage e Filter per riposizionare il record corrente, purché il provider supporti la relativa funzionalità. Gli oggetti Recordset a scorrimento in avanti supportano solo il metodo MoveNext. Quando si utilizzano i metodi Move per visualizzare ciascun record o enumerare il Recordset, è possibile utilizzare le proprietà BOF ed EOF per verificare se è stato superato l'inizio o la fine del Recordset.
Gli oggetti Recordset supportano l'aggiornamento immediato e l'aggiornamento in modalità batch: nell'aggiornamento immediato tutte le modifiche apportate ai dati vengono immediatamente scritte sulla fonte dati di livello inferiore quando si richiama il metodo Update. È inoltre possibile passare serie di valori come parametri con i metodi AddNew e Update e aggiornare contemporaneamente vari campi di un record.
Se il provider supporta l'aggiornamento in modalità batch, è possibile memorizzare nella cache le modifiche apportate a più record, quindi trasmetterle con un sola chiamata alla base di dati con il metodo UpdateBatch. Si tratta di una procedura valida per le modifiche apportate con i metodi AddNew, Update e Delete. Dopo aver richiamato il metodo UpdateBatch, è possibile utilizzare la proprietà Status per verificare e risolvere eventuali conflitti di dati.

2.3 Le stored procedure

Le stored procedure [7][10] rappresentano il "cuore" della programmazione Transact SQL. Presenti fin dalle prime versioni di SQL Server sono gruppi di istruzioni SQL compattati in un modulo e memorizzati nella cache per un successivo utilizzo.
Racchiudere il codice SQL all'interno di procedure memorizzate porta due grossi vantaggi rispetto ai batch di codice SQL tradizionale:

1. Aumento nella velocità di esecuzione del codice SQL e quindi delle performance generali delle applicazioni.
2. Aumento della leggibilità e della portabilità del codice e quindi della scalabilità delle applicazioni.

Le procedure possono essere create sia per uso permanente che temporaneo ed inoltre possono essere avviate in modo automatico quando viene avviato SQL Server.
La quantità di istruzione SQL che può accogliere una procedura è enorme: 128 MB,mentre il numero massimo di parametri che è possibile assegnare ad una procedura è 2100.
Le procedure vengono salvate su una tabella di sistema della base di dati sul quale si sta lavorando dal nome syscomments.
Sql Server stesso possiede una serie di procedure dette di sistema che vengono generate al momento della sua installazione e sono necessarie ad eseguire una serie fondamentale di compiti che vanno dalla creazione dei databases alla loro manutenzione (utenti, permessi, repliche, backup, restore, ecc…).

2.3.1 Come creare le procedure

L'istruzione DDL per la creazione di stored procedure è CREATE PROCEDURE, ecco la sintassi completa:

```
CREATE PROC [ EDURE ] nome_procedura [ ; numero]
    [ { @parametro tipo_di_dati }
          [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]

[ WITH
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]

[ FOR REPLICATION ]

AS istruzione_sql  [ ...n ]
```

E' sufficiente eseguire in un batch l'istruzione CREATE PROCEDURE dichiarando i parametri di input ed output necessari ed infine aggiungere le istruzioni Transact SQL costituenti il corpo vero e proprio della procedura. Ad esempio, creiamo la procedura p_sel_autore nel database pubs, che servirà per recuperare un autore dalla tabella authors in funzioni del proprio ID di identificazione:

```
Use pubs
Go /*inizio del batch SQL per la creazione della procedura*/
CREATE PROCEDURE dbo.p_sel_autore ( @au_id VARCHAR(11) = '' ) AS
 SELECT
      au_lname + ' ' + au_fname AS       Nome
 FROM
      authors
 WHERE
      au_id       =      @au_id
Go /*fine del batch SQL per la creazione della procedura*/
```

Da notare che l'istruzione CREATE PROCEDURE deve essere la prima del batch altrimenti la creazione della procedura fallirà.
Ad esempio il codice sottostante è errato perché prima dell'istruzione CREATE PROCEDURE c'è una SELECT (il batch è il codice SQL compreso tra la parola Go e la successiva)

```
Use pubs
Go /*inizio del batch SQL per la creazione della procedura*/
 SELECT
       au_lname + ' ' + au_fname AS        Nome
 FROM
       authors
/*   La   presenza   di   questa   SELECT   fa   fallire   la   creazione
dell'oggetto*/
CREATE PROCEDURE dbo.p_sel_autore ( @au_id VARCHAR(11) = '' ) AS
 SELECT
      au_lname + ' ' + au_fname AS       Nome
 FROM
      authors
 WHERE
      au_id       =      @au_id
 Go /*fine del batch SQL per la creazione della procedura*/
```

Nella dichiarazione dei parametri di input di una procedura è possibile assegnare per questi dei valori di default, questo è molto utile nella costruzione di procedure efficaci perché i valori che arriveranno dai parametri saranno sempre coerenti e consistenti con la logica della routine.
Se un parametro non possiede un valore di default al momento dell'esecuzione verrà chiesto di passare un valore specifico per quel parametro altrimenti SQL Server segnalerà un errore.

2.3.2 Le opzioni

Nella sintassi dell'istruzione CREATE PROCEDURE esistono alcune opzioni che è possibile specificare durante la creazione di una procedura RECOMPILE e ENCRYPTION.
La prima obbliga la ricompilazione della procedura ogni qualvolta viene eseguita, sostanzialmente la

procedura non viene messa in cache e non viene creato un piano di esecuzione ad hoc richiamabile.
La seconda permette di criptare il contenuto della procedura cosicchè nessuno all'infuori del proprietario del codice sorgente avrà accesso al suo contenuto.

2.3.3 Nidificare procedure

Le procedure possono richiamare ed essere richiamate da altre procedure e così via fino ad un livello di nidificazione pari a 32. Questo limite è imposto da SQL Server per impedire errori di overflow.
l contrario una stored procedure può chiamare altre centinaia di stored procedure al suo interno.

2.3.4 Eseguire una procedura

Ci sono diversi modi per chiamare una procedura, per la precedente possiamo usare due differenti sintassi in funzione del modo con sui vengono passati i parametri (se esistono ovviamente), una implicita ed una esplicita. Per essere più chiari creiamo una nuova procedura più complessa della precedente con più parametri di input:

```
Use pubs
Go
CREATE  PROCEDURE  dbo.p_sel_autore2  (  @state  VARCHAR(2)  ,
@contract  BIT ) AS
 SELECT
      au_lname + ' ' + au_fname AS Nome
 FROM
      authors
 WHERE
      state       =      @state
      AND
      contract    =      @contract
 RETURN(0)
 Go
/*
```

Nella modalità implicita il nome del parametro di input non viene specificato ed è passato correttamente in funzione del suo ordine di chiamata nella procedura:

```
*/
EXEC dbo.p_sel_autore2 'CA', '1' -- Questa chiamata è corretta
-- Ma se invertiamo i parametri l'esecuzione è errata
EXEC dbo.p_sel_autore2 'CA', '1'
/*
```

Nella modalità esplicita invece il nome del parametro di input viene specificato e passato senza che l'ordine di chiamata nella procedura sia importante:

```
*/
-- Questa chiamata è corretta
EXEC dbo.p_sel_autore2 @state = 'Ca', @contract = '1'
-- Ed ora invertiamo i parametri per vedere cosa succede
EXEC dbo.p_sel_autore2 @contract = '1', @state = 'CA'
--Ma anche questa chiamata è corretta, pur invertendo l'ordine di
chiamata
--questo perché abbiamo specificato i nomi dei parametri @contrat e
@state in
--in abbinamento ai valori appropriati
```

La parola chiave RETURN provoca l'uscita incondizionata dalla procedura, in qualunque posizione essa si trovi nel codice: il parser quando la incontra esce e non esegue le istruzioni sottostanti.
Oltre ad uscire è possibile abbinare un codice di uscita (rappresentato da un numero intero) che aggiunge maggiori informazioni alla nostra istruzione RETURN.
Di default il valore di RETURN è 0, se invece si verifica un errore il valore sarà diverso da 0 ovviamente. E' possibile assegnare dei valori all'istruzione RETURN, ad esempio RETURN(-100) esce dalla procedura con codice di errore uguale a –100.

```
DECLARE @ret INTEGER
EXEC @ret = dbo.p_sel_autore2 @contract = '1', @state='CA'
PRINT @ret
```

Stamperà 0, se invece nelle parentesi tonde mettiamo il valore RETURN(–100), l'istruzione PRINT stamperà –100.

2.3.5 Alterare ed eliminare le procedure

Ci sono altre due istruzioni importanti per lavorare con le procedure ALTER e DROP PROCEDURE.
La prima permette di modificare il contenuto di una procedura una volta che è stata create, per esempio cambiando il contenuto della procedura p_sel_autori, facciamo in modo di recuperare i primi 10 autori in ordine decrescente:

```
Use pubs
Go /*inizio del batch SQL per la creazione della procedura*/
ALTER PROCEDURE dbo.p_sel_autore ( @au_id VARCHAR(11) = '' ) AS
 SELECT TOP 10
      au_lname + ' ' + au_fname AS       Nome
 FROM
      authors
 WHERE
```

```
      au_id       =      @au_id
 ORDER BY au_fname DESC
 Go /*fine del batch SQL per la creazione della procedura*/
```

Se invece si vuole eliminare la procedura sarà sufficiente usare l'istruzione DROP PROCEDURE.

```
Use pubs
Go /*la procedura è eliminata*/
DROP PROCEDURE dbo.p_sel_autore
```

Alcune note:

1 - Le variabili in Transact SQL sono locali ed il loro contesto è circostanziato alla sessione in cui vengono create.
2 - Per aumentare l'efficacia di esecuzione di una procedura nella sua esecuzione è bene specificare il nome del proprietario dell'oggetto procedura (tipicamente dbo) ed anche la base di dati nel quale è contenuta

2.3.6 Alcune utili procedure di sistema

Esistono alcune procedure di sistema che possono aiutare a gestire il lavoro con le stored procedure:

sp_help: permette di avere informazioni sulla procedura (uso, tipo di parametri, ecc…)

uso: EXEC **sp_help** *nome_della_procedura*

sp_helptext: permette di vedere il testo di una stored procedure

uso: EXEC **sp_helptext** *nome_della_procedura*

sp_depends: per scoprire le dipendenze da altri oggetti

uso: EXEC **sp_depends** *nome_della_procedura*

sp_rename: per rinominare una procedura

uso: EXEC sp_rename *vecchio_nome_della_procedura, nuovo_nome_della_procedura*

Capitolo 3

Il programma S.P.P.

3.1 – S.P.P. :Supervisione Processo Produttivo

Il lavoro svolto appartiene a un progetto più ampio relativo alla gestione amministrativa di una società di Galvanica

Lo scopo principale per il quale si è reso necessario progettare ed implementare un sistema di tipo Enterprise per l'amministrazione di un ambiente aziendale, si riscontra nella volontà di apportare l'innovazioni tecnologica al fine produttivo.

Nel dettaglio i vantaggi offerti dal progetto:

1. Centralizzazione delle informazioni: tutte le documentazioni vengono registrate nell'unica base di dati aziendale, che permette quindi la distribuzione controllata, l'archiviazione protetta e la trasportabilità di tutti i dati delle varie attività.
2. Elevamento qualitativo di produzione: le rilevazioni semi automatiche permettono ai controllori di processo di verificare, in tempo reale, lo stato di avanzamento lavoro, garantendo un accurato esame dei singoli prodotti ed il relativo miglioramento qualitativo.
3. Aumento delle prestazioni amministrative: gli utenti dei settori amministrativi, vengono isolati dalle problematiche di reperibilità delle informazioni (perché automatizzate) riducendo quindi i tempi di mancata produzione.
4. Miglioramento delle intercomunicazioni: tramite svariati sistemi di comunicazione ma soprattutto grazie ad una pianificata sequenza lavorativa multiutente, ogni settore di attività è messo in condizione di operare (nell'area di validità del proprio settore) con la consapevolezza degli interventi apportati da altri operatori per lo stesso prodotto.
5. Supervisione gerarchica delle attività: tramite funzionalità di impostazioni avanzate, l'organigramma aziendale è implementato nel sistema stesso; la struttura gerarchica trova riscontro quindi nei permessi attribuiti ai singoli utenti. In generale, la struttura ascendente dei livelli, rappresenta la capacità di ottenere o meno, informazioni sempre più riepilogative e confidenziali.
6. Applicazione regole aziendali: data la natura centralizzata dell'architettura degli applicativi, si ottiene la possibilità di impostare gli standard per le regole aziendali con la garanzia che verranno rispettati.

3.2 – La base di dati

In questa sezione verranno prima descritti i problemi che si sono riscontrati durante la realizzazione della base di dati e come questi sono stati risolti. Successivamente verranno descritte le tabelle che compongono la base di dati

3.2.1 I tipi di dati

La corretta scelta dei tipi di dato da utilizzare (soprattutto il loro dimensionamento) è un problema da sempre presente in informatica, in cui si deve raggiungere un compromesso tra spreco di memoria e tipi di dato troppo piccoli. Se da un lato vi è infatti la necessità di ottimizzare lo spazio in memoria, evitando quindi lo spreco di risorse, dall'altro vi è il rischio di avere variabili sottodimensionate con il pericolo di incorrerre nell'overflow o nell'underflow.
Con il termine overflow ci si riferisce al fenomeno di traboccamento di un numero dallo spazio di memoria che questo ha a disposizione. In pratica l'overflow è un errore che si presenta quando si tenta di scrivere un numero più grande del massimo consentito dalla configurazione di n bit.Per quanto si estenda il numero di bit, o si cerchi di affiancare più byte, il fenomeno dell'overflow è ineliminabile poiché la dimensione della memoria è comunque finita.
ll fenomeno dell'underflow è inverso al problema dell'overflow: l'underflow infatti si verifica quando si cerca di memorizzare un numero più piccolo del valore minimo consentito dalla configurazione.
Underflow e overflow sono problemi complementari, entrambi infatti sono causati dal cercare di memorizzare un dato troppo grande per lo spazio disponibile.
Lavorando con un databese (SQL-Server 7.0 nel caso specifico) sorge un altro problema legato alla scelta dei tipi di dato, ovvero come convertire i tipi di dato supportati dalla base di dati con quelli supportati da Visual Basic, in particolare per i tipi di dato numerici che spesso non sono compatibili. Può infatti capitare che si abbiano delle incongruenze tra la precisione di memorizzazione di SQL-Server 7.0 e quella di Visual Basic.
In particolare in Visual Basic non è possibile dichiarare esplicitamente il numero di cifre decimali di un numero a virgola fissa, potendo usare solo il tipo di dato Currency, che offre solo 4 cifre decimali, risultando quindi inadeguato in molte situazioni, soprattutto per contenere i tipi di dato che SQL-Server 7.0 è in grado di memorizzare. E' possibile in verità utilizzare un'altra soluzione particolare in Visual Basic, ovvero il tipo di dati Decimal. Questo tipo di dato, pur essendo a virgola fissa, contiene numeri decimali divisi per una potenza di 10. Se il fattore di divisione è zero (numeri senza decimali) l'intervallo valido è +/-79.228.162.514.264.337.593.543.950.335. Per i numeri con 28 decimali, l'intervallo è +/-7,9228162514264337593543950335. Il numero diverso da zero più piccolo che può essere rappresentato è quindi 0,0000000000000000000000000001.
Uno degli inconvenienti derivati dall'uso del tipo Decimal è che non è possibile dichiarare esplicitamente un dato come Decimal, ma è necessario usare il tipo Variant (che può contenenere ogni tipo di dato) e convertirlo a Decimal. Nonostante questo inconveniente si è preferito, in certi casi, il suo utilizzo rispetto al Currency od a un numero a virgola mobile.
In prima analisi l'adozione di numeri a virgola mobile era sembrata la migliore soluzione, dovendo rappresentare delle grandezze fisiche (peso, peso specifico, volume,...) in cui quindi gli errori

eventualmente introdotti dalle operazioni aritmetiche vengono ridotti. Tuttavia essendo valori strettamente legati ad un listino prezzi (a seconda degli articoli di magazzino il prezzo è legato al peso o al volume del contenitore) si è preferito ricorrere a numeri a virgola fissa con il maggior numero di decimali disponibile e controllare via codice che i numeri immessi dall'utente nel corso della definizione di un articolo non eccedessero i limiti imposti. In questo modo è stato possibile evitare il problema dell'overflow e dell'underflow, nonostante si sia dovuto ricorrere a un tipo di dato di dimensioni maggiori, causando quindi uno spreco di memoria.

3.2.2 I Contenitori

Uno dei principali problemi che si incontrano quando si devono maneggiare dimensioni fisiche (peso, peso specifico, volume), oltre a quanto già detto riguardo a overflow e underflow, è che i valori rilevati risultano spesso imprecisi già alla sorgente. Quando si devono quindi registrare questi numeri ci si trova nella condizione di dover introdurre delle approsimazioni. Questo fenomeno è particolarmente di difficile gestione nel momento in cui serve fare delle operazioni sui dati rilevati, causando una somma degli errori invece che una loro diminuizione (secondo la teoria di propagazione dell'errore).
Ancora una volta, essendo valori legati a dei prezzi, si è cercato di avere la maggior precisione possibile, anche se questo ha comportato delle complicazioni nella gestione dei dati numerici.
Oltre alla già citata soluzione di adottare dei numeri con maggiore precisione è stato introdotto il concetto di contenitore, ovvero ogni operazione effettuata su una grandezza fisica di un articolo viene effettuata solo attraverso multipli interi di una quantità nota. Per semplificare l'operazione nella definizione di un articolo viene richiesto all'utente di indicare con quale contenitore il prodotto si presenta, specificando che in futuro potrà essere movimentato solo una quantità multipla di quel contenitore.
Questo tipo di gestione deriva anche da una specifica di programma: l'esigenza era fare in modo che a magazzino non restassero confezioni di prodotto usate a metà, con il rischio di un loro deterioramento e spreco, così facendo invece l'utente è obbligato a richiedere lotti interi di prodotto e a risponderne per tutta la quantità a lui fornita. Un'altro vantaggio è dato dalla possibilità di associare più contenitori per ogni singolo articolo, con relativa fotografia, dimensione e corrispondeza tra contenitore e prodotto. In questo modo è possibile avere a magazzino uno stesso articolo ma, a seconda del contenitore con cui è stato venduto, poterlo movimentare in maniera differente. L'adozione dell'uso dei contenitori risolve in parte il problema degli errori introdotti dall'uso di grandezze fisiche: ogni operazione risulta multipla di un contenitore prefissato e quindi gli errori possibili, dovuti a operazioni sulle quantità, vengono diminuiti. Un'altra considerazione pratica consiste nell'avere un modello che si avvicina il più possibile allo stato reale di un magazzino, con dei prodotti stoccati nei loro contenitori effettivi e con la relativa fotografia, in modo da rendere il lavoro dell'utente più agevole.
Nella strutturazione della gestione del magazzino si è cercato di virtualizzare le operazioni che l'operatore comunemente svolge, modellando il programma sulla reale organizzazione del magazzino. In quest'ottica sono state introdotte, oltre all'uso dei contenitori, le zone di stoccaggio, ovvero ogni materiale deve essere movimentato da o verso una specifica zona, che rispecchia l'effettiva strutturazione del magazzino, suddiviso appunto in zone distinte.
Ogni prodotto quindi non è più un semplice articolo con associata la sua giacenza, ma risulta un struttura più complessa, caratterizzata da uno o più contenitori e dalle zone in cui è stoccato. In questo modo può essere che uno stesso articolo sia registrato in zone differenti con contenitori differenti, così

come effettivamente accade nel magazzino reale.
Nonostante per l'utente ogni movimento da e verso il magazzino viene effettuato con multipli del contenitore definito (quindi numeri interi che rappresentano quanti contenitori sono stati movimentati), in pratica nella base di dati vengono memorrizzate le quantità fisiche di prodotto (espresse quindi in kg o litri). Da qui l'esigenza dell'utilizzo di variabili numeriche appropriate, sia lato applicazione che base di dati. Questa apparente incongruenza, ovvero la differenza tra cosa viene effettivamente memorizzato (quantità fisiche in virgola fissa con cifre decimali) e quanto richiesto all'utente (quantità multiple di un contenitore) è in verità solo apparente, in quanto essendo definiti più contenitori per ogni singolo prodotto è necessario memorizzare l'effetiva quantità movimentata e non il multiplo del contenitore, onde evitare incongruenze.
C'è inoltre da aggiungere che essendo questo il rifacimento di un programma su una base dati già esistente, si è preferito non stravolgere troppo la base dati precedente e dove possibile adattare l'applicativo alla base dati e non viceversa dal momento che questa risulta in condivisione con altri programmi.
La strutturazione dei dati prevede l'uso di un unica base di dati per tutto l'applicativo, anche se è prevista l'adozione di una base di dati separata per il solo magazzino in un prossimo futuro.
Si è voluto memorizzare esplicitamente la giacenza totale e la giacenza per zona, invece di ricorrere al suo calcolo ogni volta facendo la somma dei movimenti per evitare di far lavorare inutilimente la base di dati, calcolando valori che facilmente si possono memorizzare. Questa scelta è stata dettata dall'elevato numero di richieste dei valori di giacenza di ogni articolo, dal momento che dopo ogni movimento da o verso il magazzino l'utente ha la necessità di conoscere la situazione della giacenza, sia totale che per zona. Nonostante le query di somma di valori siano estremamente ottimizzate, in previsione di un notevole sviluppo del numero di movimenti registrati si è preferito non appesantire ulteriormente il lavoro della base di dati, inserendo appunto due tabelle al posto di una.

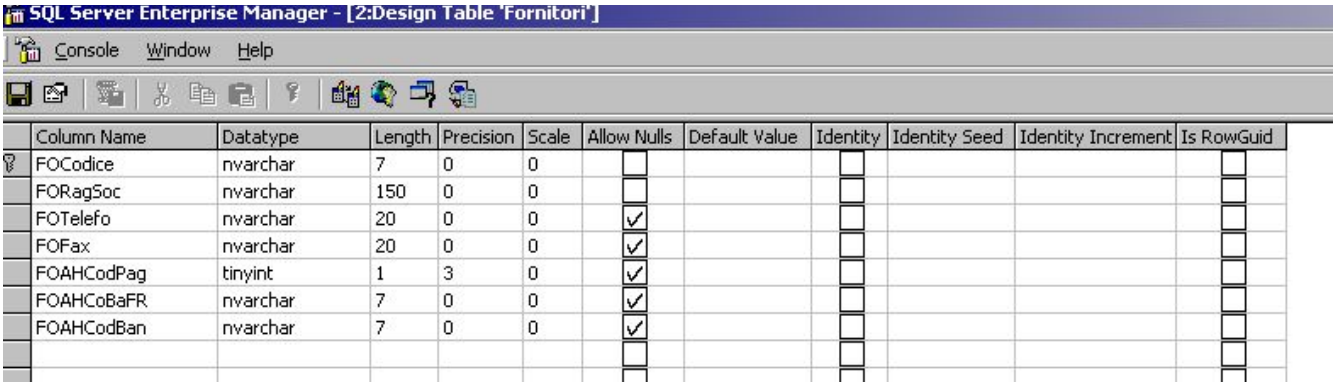
3.2.3 Indici di revisione

Ogni articolo presente a magazzino può essere soggetto a vari cambiamenti, composizione chimica, aspetto esteriore o anche semplicemente di prezzo.
L'utente potrebbe aver la necessità di fare in modo che le modifiche apportate ad un articolo abbiano effetto solo per i prossimi movimenti registrati, ovvero che lo storico già memorizzato non venga modificato (caso tipico è quello di un prezzo appunto, nei vecchi carichi deve continuare a essere registrato il vecchio prezzo, mentre per i prossimo deve essere usato il prezzo aggiornato). La soluzione più semplice è quella di eliminare la tabella degli articoli, riportando cioè di volta in volta le caratteristiche dell'articolo nelle varie tabelle interessate, ad esempio il prezzo nella tabella dei carichi in modo che ad ogni carico è possibile modificarne il valore a seconda del mercato. Questa gestione tuttavia porta ad alcuni problemi e incongruenze. Innanzitutto in questo modo si ha la replicazione dei dati, in molti casi inutilimente, se nel caso del prezzo questo potrebbe, almeno teoricamente, cambiare ogni volta, difficilmente il peso specifico, l'unità di misura cambieranno in continuazione e quindi le dimensioni della base di dati crescerebbero senza motivo. Inoltre come suddividere le varie informazioni? Le caratteristiche tecniche sono necessarie sia nel caso di carico che di scarico di un articolo, servirebbe quindi riportarle in entrambe le tabelle.
Una soluzione potrebbe essere quella di gestire i vari campi dell'articolo in modi differenti, si avrebbe

così ad esempio il prezzo memorizzato nella tabella di carico mentre i dati anagrafici nella tabella degli articoli, che verrebbe così mantenuta. Le difficoltà di questa gestione riguardano il come suddividere i campi tra le varie tabelle e inoltre non risolvono il problema della possibilità di avere uno storico delle modifiche effettuate all'articolo: se ad esempio il prezzo venisse chiesto ogni volta(presentando magari il prezzo dell'ultimo carico come riferimento) non si avrebbe nessun controllo sui carichi precendenti, viceversa se si modificasse un dato anagrafico presente nella tabella articoli questo interesserebbe indistintamente i vecchi e i nuovi movimenti.
Una soluzione intermedia, adottata dal programma, è l'introduzione degli indici di revisione, meccanismo semplice e assolutamente trasparente all'utente che dovrà infatti semplicemente indicare se le modifiche effettuate sull'articolo dovranno aver effetto solo per i prossimi movimenti oppure interessare tutto lo storico. Nel secondo caso, il più semplice da gestire, verrà semplicemente aggiornata la tabella degli articoli in modo che le modifiche verranno visualizzate (e quindi applicate) in ogni parte del programma: quando l'utente chiederà i dettagli di un articolo caricato, qualunque sia la data di carico, vedrà queste modifiche. E' questo il caso tipico di modifiche alla descrizione dell'articolo, magari sostituita con una più completa, oppure delle schede di sicurezza, aggiornate con i nuovi parametri di legge.
Le modifiche dei prezzi ovviamente non possono seguire questa logica, un vecchio carico infatti dovrà avere il prezzo coerente con il listino della data del movimento e non quello attuale. Può quindi servire che le modifiche abbiano effetto solo per i prossimi movimenti, e non solo per il prezzo. In questo caso, in modo del tutto trasparente all'utente, vengono introdotti gli indici di revisione, ovvero l'utente vedrà i vecchi movimenti con i dati prima delle modifiche, mentre i movimenti futuri presenteranno le modifiche apportate.
Per fare questo occorre aggiungere due campi alla tabella articoli, ovvero l'indice di revisione e un bit che indica la revisione corrente. Al momento del salvataggio delle modiche viene creata una copia dell'articolo, con un suo ID di tabella differente da quello dell'articolo originario, l'indice di revisione viene impostato sull'ID dell'articolo originario e il bit di revisione corrente viene impostato a 1. Nell'articolo originario viceversa viene impostato a 0 il bit relativo alla revisione corrente.
Così facendo nei vecchi movimenti, essendo linkati alla tabella articoli tramite l'ID di tabella, continuerà a essere visualizzato il vecchio articolo, mentre in quelli futuri verrà usato quello appena modificato(ovvero nei prossimi movimenti verrà usato l'ID dell'articolo con indice di revisione a 1).
In questo modo è possibile creare più revisioni, in cui quella corrente risulta quella con il bit di revisione a 1 mentre tutte le altre avranno questo bit a 0. In verità il bit di revisione non è necessario, in quanto bastererebbe controllare la data di modifica dell'articolo (che viene comunque memorizzata dal programma) che indica lo storico di tutte le modifiche: l'ultima sarà quella con data più recente. Ciononostante l'uso di un bit aggiuntivo risulta molto più comodo, snellisce le query aggiungendo un overhead davvero piccolo, dovuto ad un solo bit.
Con l'adozione degli indici di revisione in caso di modifica si ricorre sempre alla creazione di un nuovo articolo, causando nel caso di lievi modifiche un certo overhead non trascurabile. Ciò è particolarmente vero per il prezzo che effettivamente è il dato che più di ogni altro è sottoposto a cambiamenti, anche frequenti. Per ovviare a questo inconveniente sono allo studio soluzioni, non ancora implementate, per poter salvare separatamente il prezzo. Probabile soluzione sarà l'adozione di una nuova tabella, linkata a quella degli articoli per la memorizzazione di tutte le fluttuazioni di prezzo dei prodotti, accompagnata dalla possibilità per l'utente di visualizzare tutti i cambiamenti nel corso del tempo. Probabilmente verrà introdotto il concetto di preventivo associato non solo al prodotto ma anche alla quantità acquistata, in quanto ovviamente il prezzo può variare anche in maniera consistente a seconda dell'entità dell'ordine.

3.3 – Tabelle utilizzate

Fornitori



	Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
🔑	FOCodice	nvarchar	7	0	0	☐		☐			☐
	FORagSoc	nvarchar	150	0	0	☐		☐			☐
	FOTelefo	nvarchar	20	0	0	☑		☐			☐
	FOFax	nvarchar	20	0	0	☑		☐			☐
	FOAHCodPag	tinyint	1	3	0	☑		☐			☐
	FOAHCoBaFR	nvarchar	7	0	0	☑		☐			☐
	FOAHCodBan	nvarchar	7	0	0	☑		☐			☐
						☐		☐			☐

Figura 3.1 *Tabella "Fornitori"*

La tabella fornitori (Figura 3.1) contiene l'elenco di tutti i fornitori di materie prime dell'azienda. L'elenco dei fornitori risulta ricavato da una base di dati esterna in cui il codice associato è composto da 7 caratteri numerici, preceduti se necessario da degli “0” (es il codice “23” deve essere memorizzato come “0000023”). E' stata quindi una scelta obbligata quella di utilizzare il tipo di dato nvarchar per il campo “FOCodice” che rappresenta appunto il codice associato al fornitore. Questa tabella risulta quindi l'unica a non avere un identificativo di riga gestito automaticamente dalla base di dati: anche se il campo”FOCodice” è chiave primaria non è autoincrementato automaticamente dalla base di dati ad ogni nuovo inserimento. Il campo “FORagSoc” rappresenta la ragione sociale del fornitore ed è costituito da un campo nvarchar di 150 caratteri. Questi due campi sono gli unici obbligatori per l'utente quando intende inserire un nuovo fornitore. Questa tabella risulta collegata alla tabella ArticoliFornitori tramite il campo “FOCodice”: ogni articolo ha associato un solo fornitore mentre a ciascun fornitore possono essere associati più articoli.
Non sono previsti indici di revisione per questa tabella: ogni modifica ha effetto retroattivo.

Articoli Fornitori

SQL Server Enterprise Manager - [2:Design Table 'ArticoliFornitori']

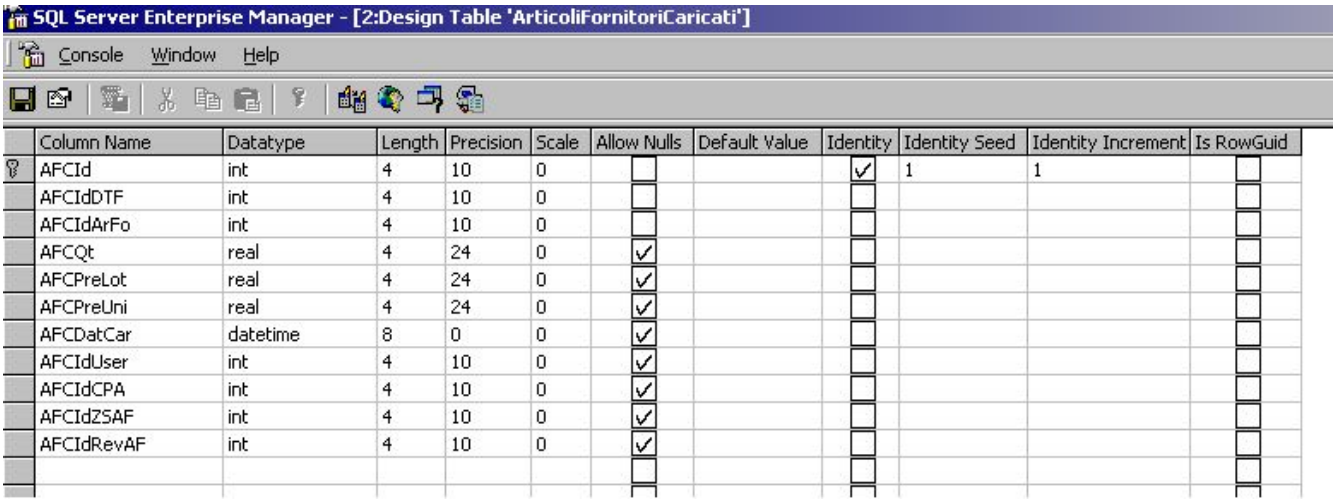
Console Window Help

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
AFId	int	4	10	0	☐		☑	1	1	☐
AFIdForn	nvarchar	7	0	0	☐		☐			☐
AFCodice	nvarchar	40	0	0	☐		☐			☐
AFDescri	nvarchar	150	0	0	☐		☐			☐
AFUniMis	nvarchar	5	0	0	☐		☐			☐
AFUniMis2	nvarchar	5	0	0	☐		☐			☐
AFScoMin	real	4	24	0	☑		☐			☐
AFPreUni	real	4	24	0	☑		☐			☐
AFPesSpe	real	4	24	0	☑		☐			☐
AFDatLis	datetime	8	0	0	☑		☐			☐
AFIdAE	int	4	10	0	☑		☐			☐
AFCorAE	real	4	24	0	☑		☐			☐
AFCorAE2	real	4	24	0	☑		☐			☐
AFImgPth	nvarchar	100	0	0	☐		☐			☐
AFIdMaIn	int	4	10	0	☐		☐			☐
AFIdUser	int	4	10	0	☑		☐			☐
AFRevCor	bit	1	0	0	☑		☐			☐
AFRevId	int	4	10	0	☑		☐			☐
AFRevDat	datetime	8	0	0	☑		☐			☐
AFIdCont	int	4	10	0	☑		☐			☐
AFIdAm	int	4	10	0	☑		☐			☐
AFIdCAF	int	4	10	0	☑		☐			☐
AFIdZSAF	int	4	10	0	☑		☐			☐
AFLstRev	bit	1	0	0	☑		☐			☐
					☐		☐			☐

Figura 3.2 *Tabella ArticoliFornitori*

In Figura 3.2 possiamo vedere la tabella relativa agli articoli associati a un fornitore. Il campo "AFIdForn" risulta associato al campo "FOCodice" e rappresenta il codice del fornitore del prodotto specifico. Rispetto alla tabella Fornitori la tabella ArticoliFornitori possiede il campo"AFId" che oltre ad essere chiave primaria risulta autoincrementato e gestito autonomamente dalla base di dati (è cioè una identità).
Da notare inoltre il campo "AFRevCor" che indica se il record rappresenta la revisione corrente dall'articolo e il campo "AFRevId" che rappresenta l'indice di revisione dell'articolo: se è uguale a "AFId" significa che questo record non ha subito revisioni ed è l'unico relativo a quell'articolo. Tutte le revisioni di un articolo hanno in comune il campo "AFRevCor" che rappresenta l'indice ("AFId") della prima revisione dell'articolo, ovvero quella più vecchia.

Articoli Fornitori Caricati



	Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
🔑	AFCId	int	4	10	0	☐		☑	1	1	☐
	AFCIdDTF	int	4	10	0	☐		☐			☐
	AFCIdArFo	int	4	10	0	☐		☐			☐
	AFCQt	real	4	24	0	☑		☐			☐
	AFCPreLot	real	4	24	0	☑		☐			☐
	AFCPreUni	real	4	24	0	☑		☐			☐
	AFCDatCar	datetime	8	0	0	☑		☐			☐
	AFCIdUser	int	4	10	0	☑		☐			☐
	AFCIdCPA	int	4	10	0	☑		☐			☐
	AFCIdZSAF	int	4	10	0	☑		☐			☐
	AFCIdRevAF	int	4	10	0	☑		☐			☐
						☐		☐			☐

Figura 3.3: *Tabella ArticoliFornitoriCaricati*

In Figura 3.3 possiamo vedere la tabella in cui sono registrati i movimenti di carico di uno specifico articolo. Tramite il campo “AFCIdArFo” è possibile risalire all'articolo movimentato mentre il campo “AFCIdRevAF” indica l'Id della revisione corrente di quello specifico articolo (in questo modo è possibile risalire direttamente alla attuale revisione dell'articolo movimentato senza ulteriori operazioni).
Questa tabella risulta collegata alla tabella relativa ai contenitori tramite il campo “AFCIdCPA”, in modo da avere indicato con quale specifico contenitore è stato movimentato l'articolo. Il campo “AFCIdZSAF” è collegato invece alla tabella relativa alle zone di stoccaggio, in modo da poter risalire alla zona specifica in cui il prodotto è stato immagazzinato. Il campo “AFCQt” rappresenta la quantità caricata nella specifico movimento, espressa in unità di misura principale: sebbene infatti ogni movimento registrato debba essere un multiplo di un contenitore risulterebbe troppo oneroso memorizzare il valore intero dei contenitori. Ogni operazione sulle quantità dovrebbe infatti tenere conto anche del contenitore utilizzato.

Articolo Fornitori Scaricati

SQL Server Enterprise Manager - [2:Design Table 'ArticoliFornitoriScaricati']

Console Window Help

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
AFSID	int	4	10	0	☐		☑	1	1	☐
AFSIdMaga	int	4	10	0	☑		☐			☐
AFSIdArFo	int	4	10	0	☐		☐			☐
AFSQt	real	4	24	0	☐		☐			☐
AFSData	datetime	8	0	0	☑		☐			☐
AFSIdCaSc	int	4	10	0	☑		☐			☐
AFSIdMaEs	int	4	10	0	☑		☐			☐
AFSIdUser	int	4	10	0	☑		☐			☐
AFSCodDIP	varchar	10	0	0	☑		☐			☐
AFSIdCPA	int	4	10	0	☑		☐			☐
AFSIdZSAF	int	4	10	0	☑		☐			☐
AFSDataReg	datetime	8	0	0	☑		☐			☐
AFSIdRevAF	int	4	10	0	☑		☐			☐
					☐		☐			☐

Figura 3.4: *ArticoliFornitoriScaricati*

La tabella in Figura 3.4 risulta speculare alla tabella relativa ai carichi, mentre in quella precedente sono infatti registrati i movimenti di carico a magazzino in questa vengono memorizzati quelli di scarico.
Come per l'altra tabella abbiamo quindi i campi "AFSIdZSAF" e "AFSIdCPA" che indicano rispettivamente l'Id alla tabella della zona dalla quale è stato scaricato il materiale e l'Id alla tabella dei contenitori per indicare il contenitore utilizzato per l'operazione.
Da notare il campo "AFSIdCaSc" che indica la causale dello scarico, ovvero la destinazione del materiale scaricato (es: in quale linea il prodotto verrà utilizzato).

Allineamento Articoli Fornitori

SQL Server Enterprise Manager - [2:Design Table 'AllineamentoGiacenzaAF']

Console Window Help

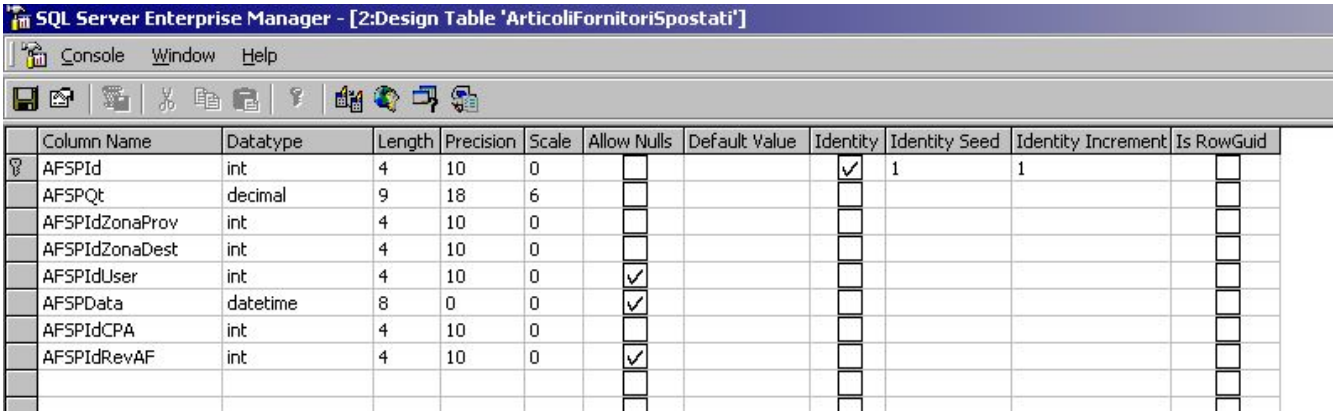
Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
AGAFId	int	4	10	0	☐		☑	1	1	☐
AGAFIdRevAF	real	4	24	0	☐		☐			☐
AGAFDate	datetime	8	0	0	☑		☐			☐
AGAFCausale	nvarchar	150	0	0	☑		☐			☐
AGAFQta	real	4	24	0	☐		☐			☐
AGAFIdUser	tinyint	1	3	0	☑		☐			☐
AGAFIdZSAF	int	4	10	0	☑		☐			☐
AGAFIdCPA	int	4	10	0	☑		☐			☐
					☐		☐			☐

Figura 3.5: *Tabella AllineamentoArticoliFornitori*

La Figura 3.5 rappresenta la tabella degli allineamenti del magazzino, che come indicato più avanti,

sono operazioni del tutto eccezzionali eseguite nel caso di errori nella normale registrazione delle operazioni da e verso il magazzino. Risulta quindi organizzata come le tabelle di carico e scarico, dove oltre alla quantità movimentata è necessario indicare anche la zona e il contenitore utilizzato per effettuare il movimento.

Articoli Fornitori Spostati

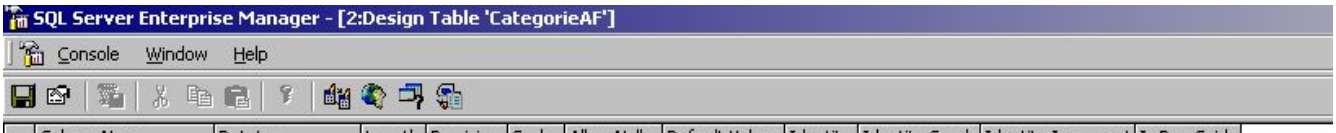


Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
AFSPId	int	4	10	0			✓	1	1	
AFSPQt	decimal	9	18	6						
AFSPIdZonaProv	int	4	10	0						
AFSPIdZonaDest	int	4	10	0						
AFSPIdUser	int	4	10	0	✓					
AFSPData	datetime	8	0	0	✓					
AFSPIdCPA	int	4	10	0						
AFSPIdRevAF	int	4	10	0	✓					

Figura 3.6: *Tabella ArticoliFornitoriSpostati*

In Figura 3.6 è indicata la tabella che contiene i record relativi agli spostamenti di articoli da una zona all'altra del magazzino, per una sua eventuale riorganizzazzione. E' linkata alla tabella ArticoliFornitori tramite il campo "AFSPIdRevAF": il magazzino è infatti organizzato in base agli Id di revisione, non viene fatta differenza tra una revisione e l'altra nel conteggio della giacenza.
Come per tutti i movimenti relativi al magazzino è necessario indicare (tramite il campo "AFSPIdCPA") quale contenitore si vuole spostare. E' ovviamente necessario indicare la zona da cui viene prelevata la quantità e la zona di destinazione, tramite i campi "AFSPIdZonaProv" e "AFSPIdZonaDest".

Categorie Articoli Fornitori



Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
CAFId	int	4	10	0			✓	1	1	
CAFDescri	nvarchar	50	0	0	✓					
CAFNome	nvarchar	50	0	0	✓					
CAFRevCor	bit	1	0	0	✓	(1)				

Figura 3.7*: Tabella CategorieArticoliFornitori*

I prodotti del magazzino risultano suddivisi in categorie di appartenenza, in base a ben precisi standard chimici e di utilizzo del materiale stesso (Figura 3.7). Ogni articolo della tabella ArticoliFornitori risulta collegato a questa tabella tramite il campo "CAFId": ogni prodotto appartiene a una e una sola

categoria. Il campo “CAFRevCor” indica se quella specifica categoria è ancora in uso: se il bit è a “0” significa che l'utente non può usare quella specifica categoria per i nuovi prodotti, ma resta associata soltanto a quelli già in uso.

Destinazione Impiego Personale

SQL Server Enterprise Manager - [2:Design Table 'DestinazioneImpiegoPersonale']

Console Window Help

	Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
🔑	DIPFlag	nvarchar	5	0	0	☐		☐			☐
	DIPDescri	nvarchar	50	0	0	☐		☐			☐
	DIPOpeRic	nvarchar	50	0	0	☑		☐			☐
	DIPLineaLav	bit	1	0	0	☐	(0)	☐			☐
	DIPScaMag	bit	1	0	0	☐	(1)	☐			☐
	DIPScaLinea	bit	1	0	0	☐	(1)	☐			☐
						☐		☐			☐

Figura 3.8: *Tabella DestinazioneImpiegoPersonale*

.

Ogni volta che l'utente registra una operazione di scarico deve essere indicata la destinazione dell'articolo, ovvero dove verrà fisicamente utilizzato il prodotto. Questa tabella (Figura 3.8) contiene tutte le possibili destinazioni di un prodotto e risulta in comune con un'altra sezione del programma, quella relativa alla registrazione degli orari di lavoro dei dipendenti: per principio infatti un prodotto può essere utilizzato unicamente dove vi possa essere un operatore, da qui la possibilità di avere una tabella in comune, in sola consultazione in entrambi i casi. Vi sono poi alcuni bit di supporto che specificano meglio il tipo di destinazione (“DIPLineaLav”, “DIPScaMag”) che indicano rispettivamente se è una linea di lavorazione (la maggior parte delle destinazioni), se è una destinazione che può essere usata solo per lo scarico del magazzino e non come impiego del personale. “DIPScalinea” è invece un campo utilizzato per la registrazione degli orari di lavoro.

Prossimo sviluppo sarà la divisione della tabella in due tabelle distinte, una per la destinazione del materiale e l'altra per quella degli operatori: fin'ora è stata utilizzata una sola tabella per retrocompatibilità.

Contenitori Per Articoli

SQL Server Enterprise Manager - [2:Design Table 'ContenitoriPerArticoli']

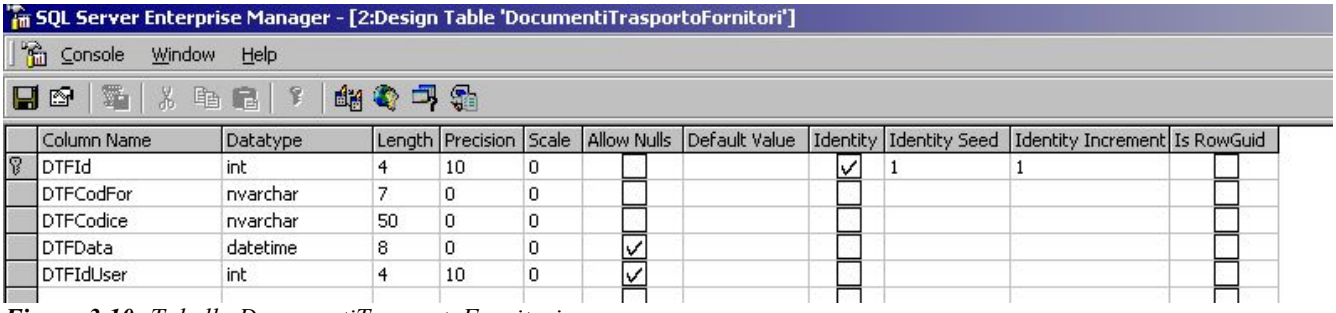
Console Window Help

	Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
	CPAId	int	4	10	0	☐		☑	1	1	☐
	CPAIdRevAF	int	4	10	0	☐		☐			☐
	CPAIdAEAF	int	4	10	0	☐		☐			☐
	CPACorAE	decimal	9	18	6	☐		☐			☐
	CPACorAE2	decimal	9	18	6	☑	(null)	☐			☐
	CPAImgPth	nvarchar	50	0	0	☑	(null)	☐			☐
	CPARevCor	bit	1	0	0	☐	(1)	☐			☐
						☐		☐			☐

Figura 3.9: *Tabella ContenitoriPerArticoli*

Ad ogni articolo risulta associato uno o più contenitori che devono essere specificati al momento della registrazione di un movimento da e verso il magazzino. Ogni operazione deve essere espressa in multipli di contenitori presenti nella tabella in Figura 3.9.
Il campo "CPAIdRevAF" indica l'id di revisione dell'articolo a cui il contenitore è associato: ad ogni contenitore è associato uno e un solo articolo mentre a un articolo possono essere associati più contenitori. Il campo "CPAIdAEAF" è collegato alla tabella AspettoEsterioreAF e indica la tipologia di contenitore: se è un cassetta metallica, un fusto, un sacco, etc.
Tramite il campo "CPACodAE" viene memorizzata la corrispondenza tra l'unità di misura principale e il contenitore stesso, corrispondenza che viene utilizzata per registrare le quantità movimentate.
Se l'articolo possiede anche una seconda unità di misura (come per esempio alcuni liquidi che vengono venduti a peso) viene registrata anche la corrispondenza con quest'ultima.
E' possibile inoltre associare una immagine al contenitore ("CPAImgPth"), una fotografia generalmente, che aiuta gli operatori nella registrazione. E' un percorso relativo a un cartella condivisa sul server, il cui path viene passato all'avvio del programma (in caso di spostamento non serve quindi modificare la tabella "ContenitoriPerArticoli")

Documenti Trasporto Fornitori



	Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
🔑	DTFId	int	4	10	0	☐		☑	1	1	☐
	DTFCodFor	nvarchar	7	0	0	☐		☐			☐
	DTFCodice	nvarchar	50	0	0	☐		☐			☐
	DTFData	datetime	8	0	0	☑		☐			☐
	DTFIdUser	int	4	10	0	☑		☐			☐

***Figura 3.10**: Tabella DocumentiTrasportoFornitori*

Ogni operazione di carico a magazzino deve riportare il riferimento al documento di trasporto relativo (Figura 3.10). In questa tabella è indicato il codice relativo al fornitore ("DTFCodFor") collegato alla tabella Fornitori, la data di registrazione ("DTFData") l'Id dell'utente ("DTFIdUser") e il codice associato all' Documento di Trasporto ("DTFCodice").

Giacenza Articoli Fornitori

SQL Server Enterprise Manager - [2:Design Table 'GiacenzaArticoliFornitori']

Console Window Help

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
GAFQt	real	4	24	0	☑		☐			☐
GAFPreUni	real	4	24	0	☑		☐			☐
GAFDaUlOp	datetime	8	0	0	☑		☐			☐
GAFIdRevAF	int	4	10	0	☐		☐			☐
					☐		☐			☐

***Figura 3.11**: Tabella GiacenzaArticoliFornitori*

In Figura 3.11 è rappresentata la tabella della giacenza totale di ogni articolo (giacenza = carichi – scarichi + allineamenti) indipendentemente dalla zona specifica di stoccaggio e dal contenitore. E' indicata la quantità, espressa in unità di misura principale ("GAFQt") e l'Id di revisione dell'articolo a cui si riferisce. Questa tabella viene aggiornata ogni volta che vengono effettuti dei movimenti da e verso il magazzino.

Giacenza per Zona Articoli Fornitori

SQL Server Enterprise Manager - [2:Design Table 'GiacenzaZoneAF']

Console Window Help

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
GZAFId	int	4	10	0	☐		☑	1	1	☐
GZAFIdZSAF	int	4	10	0	☐		☐			☐
GZAFQt	decimal	9	18	6	☐		☐			☐
GZAFIdRevAF	int	4	10	0	☐		☐			☐
GZAFIdCPA	int	4	10	0	☐		☐			☐
GZAFDaUlOp	datetime	8	0	0	☐		☐			☐
					☐		☐			☐

***Figura 3.12:** Tabella GiacenzaZoneAF*

La tabella GiacenzaZoneAF (Figura 3.12) rappresenta la giacenza di ogni prodotto in ogni singola zona del magazzino e per un particolare contenitore. In una zona quindi uno stesso prodotto può essere stoccato con contenitori differenti e in zone differenti può essere registrato con lo stesso contenitore. La somma di tutte le giacenze nelle varie zone e con i vari contenitori è ovviamente uguale alla giacenza totale. Il campo "GZAFQt" rappresenta la quantità stoccata espressa in unità di misura principale, "GZAFIdZSAF" l'Id della zona in cui è presente fisicamente il prodotto mentre il campo "GZAFIdCPA" il contenitore utilizzato.
Da notare il campo "GZAFDaUlOp" che indica l'ultimo aggiornamento della tabella: incrociato con le date di registrazione delle operazioni di carico e scarico è possibile controllare eventuali problemi.

Aspetto Esteriore

SQL Server Enterprise Manager - [2:Design Table 'AspettoEsterioreAF']

Console Window Help

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
AEId	int	4	10	0	☐		☐			☐
AEDescri	nvarchar	200	0	0	☐		☐			☐
AEIsInt	bit	1	0	0	☐		☐			☐
AENome	nvarchar	50	0	0	☑		☐			☐
					☐		☐			☐
					☐		☐			☐

Figura 3.13: *Tabella AspettoEsterioreAF*

Ogni contenitore associato ad un articolo si presenta imballato in un certo involucro, che può essere una scatola di cartone, un sacco, una cassetta metallica o altro (Figura 3.13). L'involucro esterno rappresenta semplicemente una descrizione esterna del prodotto, e serve all'utente per identificare il prodotto fornendo una descrizione più dettagliata dello stesso.

Scheda Tecnica e Scheda di Sicurezza

SQL Server Enterprise Manager - [2:Design Table 'ArticoliFornitoriSchedeTS']

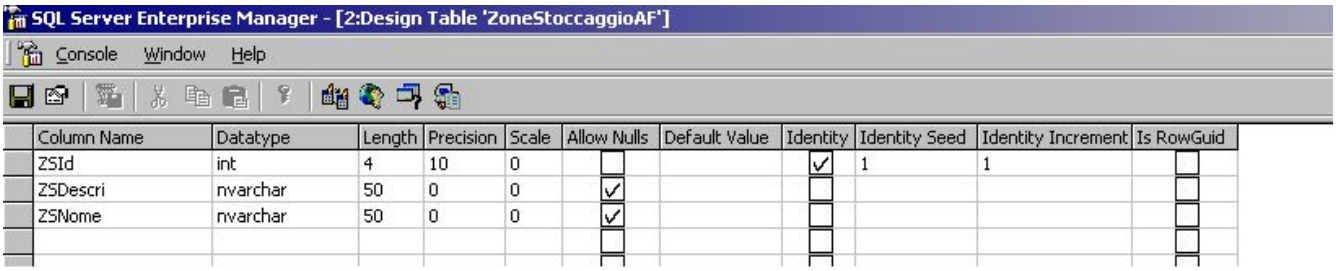
Console Window Help

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
STSIdRevAF	int	4	10	0	☐		☐			☐
STSTecnica	text	16	0	0	☑		☐			☐
STSSicurezza	text	16	0	0	☑		☐			☐
					☐		☐			☐

Figura 3.14: *Tabella ArticoliFornitoriSchedeTS*

Ad ogni articolo sono associate due schede: una di sicurezza e una tecnica (Figura 3.14) che rappresentano rispettivamente le caratteristiche intrinseche del prodotto e le operazioni da svolgere in caso di incidente. Non essendoci la possibilità di definire uno schema preciso da compilare a cura dell'utente si è preferito fornire all'utente la possibilità di creare un Rich Text Format da inserire all'interno della tabella. Il formato RTF è infatti supportato nativamente da SQL-Server e permette una certa libertà di impaginazione (consente anche l'inserimento di alcune tipologie di immagini) nonostante occupi uno spazio non indifferente. Dal momento che entrambe le schede sono raramente richieste in consultazione da parte dell'utente si è preferito memorizzarle su una tabella a parte e visualizzarle all'utente solo su esplicita domanda: in questo modo vengono snellite la maggior parte delle query relative alla anagrafica, che generalmente non richiedono la visualizzazione delle schede, mentre vengono visualizzate su richiesta senza un eccessivo overhead.

Zone stoccaggio



Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment	Is RowGuid
ZSId	int	4	10	0	☐		☑	1	1	☐
ZSDescri	nvarchar	50	0	0	☑		☐			☐
ZSNome	nvarchar	50	0	0	☑		☐			☐
					☐		☐			☐

Figura 3.15*: Tabella ZoneStoccaggioAF*

Il magazzino risulta suddiviso in zone di stoccaggio (Figura 3.15) e ogni operazione di carico scarico e allineamento è relativa a una particolare zona e a un particolare contenitore associato all'articolo stesso.

3.4 – Funzioni: Analisi Applicativo

L'applicativo sviluppato, facendo parte di una applicazione più ampia, si avvale dell'uso di DLL e eseguibili esterni per svolgere alcune funzioni generiche.
In particolare devono essere menzionati l'eseguibile SPPCentralService che attraverso Dcom viene lanciato, dal client sul server, e permette l'autenticazione diretta sul database e la DLL SPPGenericFunction che contiene funzioni generiche di manipolazione di stringhe e di valori numerici.
Per la gestione degli accessi al server e quindi per distribuire la stringa di connessione ai vari client è stato creato l'eseguibile SPPCentralService, che eseguito sul server gestisce gli accessi dei singoli utenti alla base di dati (Figura 3.16). Quando l'utente esegue il programma SPP questo richiede una login, composta da nome utente e password, che vengono passati a SPPCentralService, che verifica consultando la base di dati se l'utente può accedere all'uso del programma (e quindi alla base dati). In caso di risposta affermativa viene ritornata al client una stringa di connesssione tramite la quale può connettersi alla base di dati.
Il primo utente che lancia SPPCentralService ne causa la reale creazione che poi migrerà sul server, ogni utente seccessivo invece creerà si il processo sulla propria macchina ma se è già presente una istanza del processo sul server verrà sfruttata quella senza inutili duplicazioni.

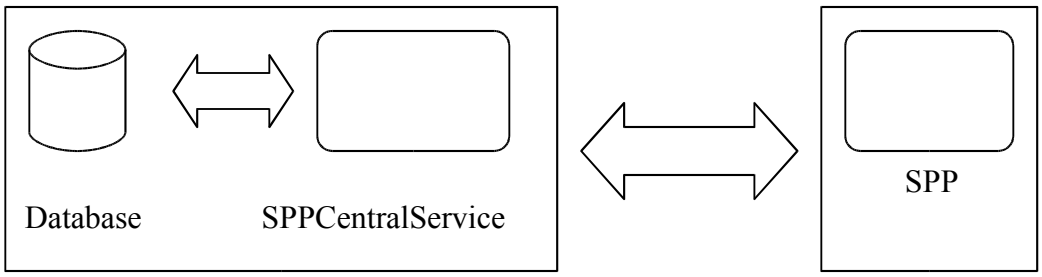


Figura 3.16 *Architettura di S.P.P.*

L'interfaccia utente per la gestione del magazzino è composta da 9 form principali, ovvero maschere tramite cui l'utente può interagire con la base dati (Figura 3.17).
L'idea di fondo è quella di isolare il più possibile la base dati all'utente, formattando i dati in modo da presentarli nel modo più semplice e comprensibile, indipendentemente dall'implementazione effettiva.

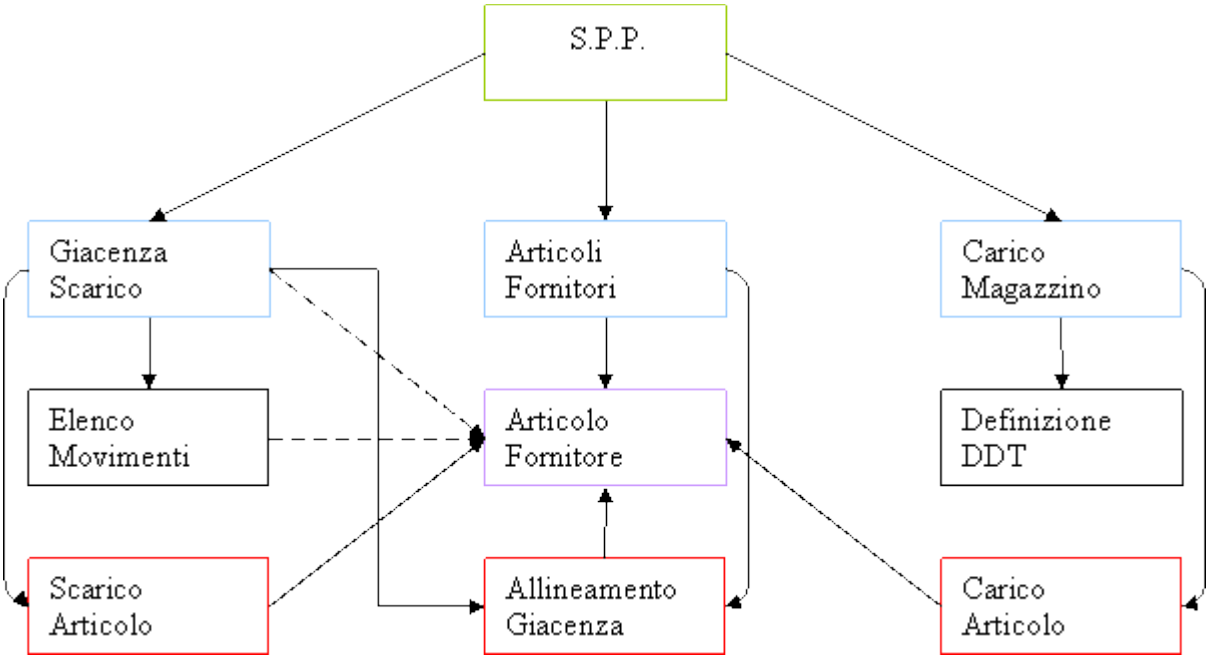


Figura 3.17:*Flusso di gestione del Magazzino*

Facendo parte di un programma più generale è presente un menù comune tramite il quale è possibile accedere a tutte le sezioni del programma, dopo essersi autenticati (Figura 3.18).

Figura 3.18: *Login*

in Figura 3.19 è visibile il menù tramite il quale si può scegliere quali funzionalità usare

Figura 3.19 *Menu iniziale*

Dopo aver selezionato la parte concernente il magazzino è possibile effettuare le seguenti operazioni

- Creazione di un nuovo articolo, relativo a un fornitore precedentemente selezionato. I campi obbligatori sono il codice articolo (generalmente fornito dal fornitore), l'unità di misura, il peso specifico e il prezzo. Non obbligatori sono la descrizione, la fotografia, la composizione del materiale(ovvero se polvere, solido o liquido) la categoria di appartenenza (acido, base...). E' necessario definire almeno un contenitore da associare all'articolo prima di poterlo movimentare(vd. punti successivi)
- Per ogni articolo è possibile associare una scheda di sicurezza (come comportarsi in caso di incidenti) e una scheda tecnica descrittiva del prodotto.
- Modifica di un articolo: tutti i campi sono modificabili se non è stato movimentato l'articolo, altrimenti l'unità di misura e il peso specifico non sono modificabili
- Creazione di un contenitore. Ad ogni articolo deve essere associato almeno un contenitore per poterlo movimentare. I dati obbligatori relativi al contenitore sono l'aspetto esteriore (ovvero il tipo di contenitore, se una cassetta, un fusto o altro) e la capacità del contenitore rispetto alla unità di misura principale. Facoltativa è la fotografia del contenitore.

- Operazioni di carico: deve essere specificato l'articolo da caricare, la zona in cui effettuare il carico, il contenitore usato, la quantità e il Documento di Trasporto.

- Operazione di scarico, deve essere indicato l'articolo, il contenitore e la zona di provenienza, la destinazione di impiego (ovvero in quale linea lavorativa verrà utilizzato) e la quantità.
- Operazione di allineamento: potrebbe capitare che alcune operazioni non siano state registrate per dimenticanza o omissione e quindi in via del tutto eccezionale per poter far coincidere la giacenza reale presente a magazzino con quella registrata l'utente deve poter avere la possibilità di allineare una quantità, ovvero introdurre un movimento che aggiunge o sottrae quantità alla giacenza.

3.4.2 Visualizzazzione Articoli Fornitoril

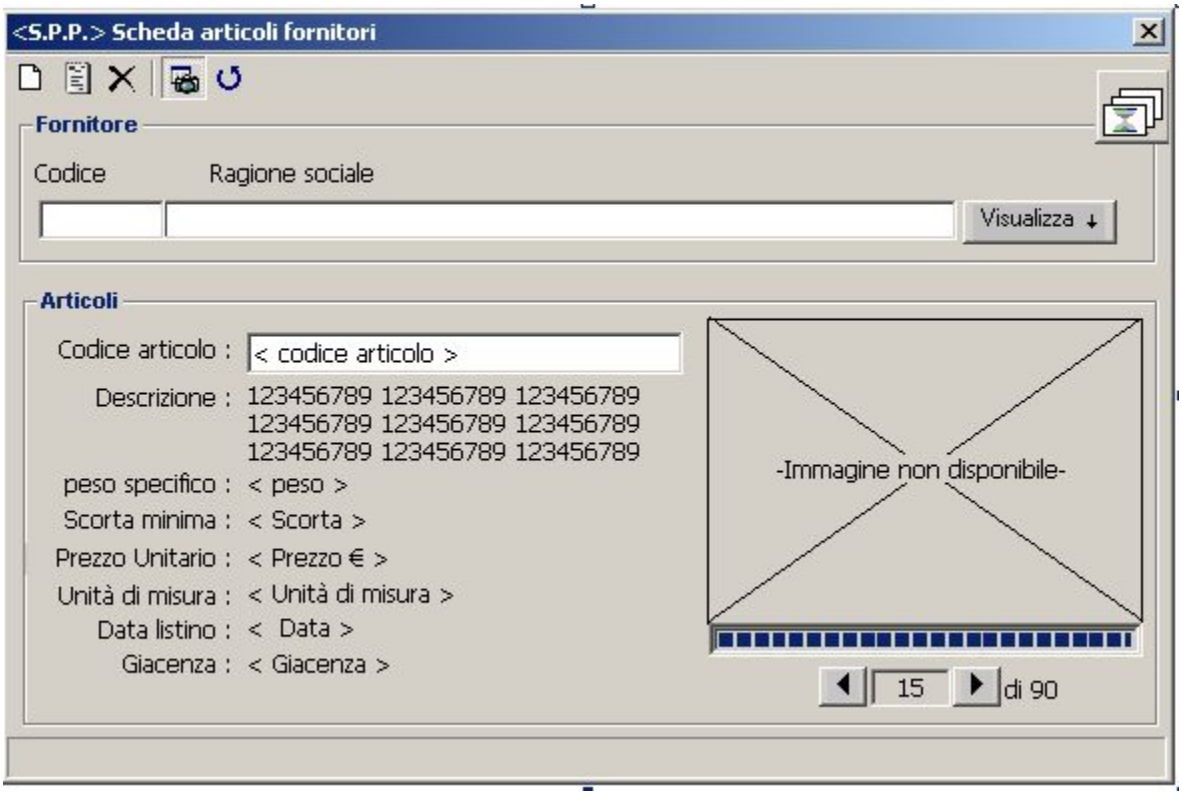


***Figura 3.20**: Visualizzazione Articoli Fornitori*

Funzionalità del form

Per poter effetuare dei movimenti da e verso il magazzino l'utente deve poter visionare gli articoli associati a ciascun fornitore, poterli modificare, eliminare e aggiungere (Figura 3.20).
La prima operazione che quindi si aspetta di dover effettuare è l'indicazione del fornitore a cui sono associati i prodotti desiderati. La selezione del fornitore può avvenire tramite l'indicazione del codice associato oppure tramite l'indicazione della Ragione Sociale (scegliendo da un elenco premendo [F9]). Dopo aver selezionato un fornitore all'utente viene presentata la scheda del primo articolo e tramite due appositi pulsanti può scorrere tutte le schede degli articoli. Tramite la barra delle funzioni in alto (e anche tramite alcune shortcut da tastiera) l'utente può aggiungere, modificare o eliminare un articolo, oppure allinearne la giacenza. Queste funzionalità (eccezzion fatta per l'eliminazione) prevedono l'apertura di un altro form per completare le operazioni.

Gestione delle funzioni

Progettando la gestione della scelta del fornitore si è dovuto tener conto di una duplice esigenza dell'utente: se da un lato è molto probabile che conosca il codice del fornitore a cui vuole fare riferimento (fornitori abituali) e quindi voglia poterlo selezionare inserendo un semplice codice che conosce a memoria, dall'altro è possibile che per molti fornitori non ricordi il codice associato e voglia

quindi scorrere un elenco da cui selezionarlo.
Da subito è stata scartata la presentazione all'utente di un elenco completo, caricato all'apertura del form e visualizzato in una griglia, da cui poter selezionare il record interessato. Questa soluzione infatti offrirebbe una visualizzazzione pesante dal punto di vista grafico, soprattutto se l'utente conoscesse già il codice del fornitore. Sarebbe inoltre poco usabile, essendo l'elenco abbastanza lungo, dovendo quindi scorrere diverse pagine prima di posizionarsi sul fornitore desiderato. Servirebbe quindi una funzione di ricerca, annullando quindi la necessità di avere l'elenco completo.
La soluzione che è stata adottata è quindi, come si può vedere, intermedia, ovvero l'utente ha la possibilità di indicare direttamente il codice del fornitore nell'apposita casella oppure di effettuare una ricerca tramite la casella di testo "Ragione Sociale". In questo caso infatti se viene inserito del testo e viene battuto [INVIO] viene selezionato il primo fornitore la cui ragione sociale contiene le lettere inserite, mentre se viene premuto [F9] viene aperta una finestra con tutti i fornitori che contengono la stringa indicata. In questo modo l'elenco completo dei fornitori è si accessibile all'utente (immettendo una stringa vuota nella casella di ricerca vengono ritornati tutti i fornitori) ma solo su richiesta esplicita, alleggerendo l'interfaccia.

Dovendo offrire questa duplice scelta si è preferito scaricare all'apertura del form la ragione sociale e il codice di tutti i fornitori su un Recordset lato client disconnesso, su cui poi è possibile effettuare ricerche e posizionamenti senza particolari problemi o ritardi. In questo modo infatti a fronte di una certa lentezza all'apertura dovuto al lavoro della base di dati e soprattutto della rete, ogni successiva ricerca risulta estremamente veloce, lavorando in locale, e senza un sovraccarico ulteriore della base di dati.
L'uso previsto di questo form infatti è la selezione di numerosi articoli appartenenti a diversi fornitori, alcuni dei quali conosciuti, altri no. Effettuare quindi ad ogni richiesta dell'utente una query specifica alla base di dati risulterebbe eccessivo e soprattutto ridondante, con infatti la possibilità di rieffettuare una stessa ricerca a breve distanza. Si è quindi cercato di migliorare il caso pessimo (worst case), ovvero la situazione in cui l'utente richiede ripetutamente delle ricerche con stringhe differenti.
Questa scelta, che potrebbe sembrare troppo prudente, si è in verità resa necessaria dato il differente uso che gli utente fanno del programma: alcuni utenti conoscono a memoria il codice del fornitore cercato mentre altri ricordano solo parte della ragione sociale e necessitano quindi di continue e ripetute ricerche.
Per poter effettuare delle ricerche all'interno dei record del Recordset dei fornitori serve controllare attentamente la stringa di ricerca immessa, onde evitare problemi con la base di dati. E' necessario rimuovere gli eventuali spazi introdotti all'inizio o alla fine della stringa e sostituire il carattere Ascii (39)' e Ascii(34)" in Ascii(96)`, per non causare problemi con la stringa SQL di ricerca.
E' stata creata un funzione apposita per effettuare queste operazioni di controllo sulle stringhe ed è stata inserita in una DDL (SPPExtension) assieme ad altre funzioni di uso comune in modo da evitare la replica del codice e da avere un maggiore controllo sulle funzioni di uso generico, per facilitare il debug e un possibile aggiornamento, nonché avere la certezza che il programma esegua azioni simili in modo standardizzato.

Dopo aver scelto il fornitore tramite il tasto "visualizza" vengono mostrati all'utente gli articoli relativi. Dovendo visualizzare tutti gli articoli associati a un fornitore e non sapendo a priori il numero di articoli associati, si è preferito ricorrere a una visualizzazione per schede, ovvero all'utente viene presentato il primo articolo associato e poi tramite dei bottoni può scorrere l'intero elenco oppure ricercare, tramite l'apposita casella il codice ricercato.

Come già per i fornitori si è preferito scaricare su un Recordset lato client tutti gli articoli relativi a uno specifico fornitore, per evitare ancora una volta un eccessivo lavoro della rete e della base di dati in caso di continue ricerche. Uno dei fattori negativi dovuti a questa gestione è che se l'utente richiede gli articoli di un fornitore "A", poi gli articoli di un fornitore "B" e successivamente di nuovo gli articoli del fornitore "A" questi vengono riscaricati interamente dalla base di dati, causando un inutile overhead. Tuttavia questa situazione è statisticamente molto improbabile, difficilmente l'utente ha la necessità di ritornare su un fornitore appena visionato. Nota positiva di questa gestione è che i dati relativi agli articoli sono sempre aggiornati, mentre quelli relativi ai fornitori vengono ricaricati solo alla chiusura e riapertura del form. Le modifiche relative ai fornitori sono infatti estremamente rare, mentre quelle relative agli articoli (seppur comunque non frequenti) sono comunque più usuali.

3.4.3 Visualizzazzione/Creazione/Modifica Articolo Fornitore

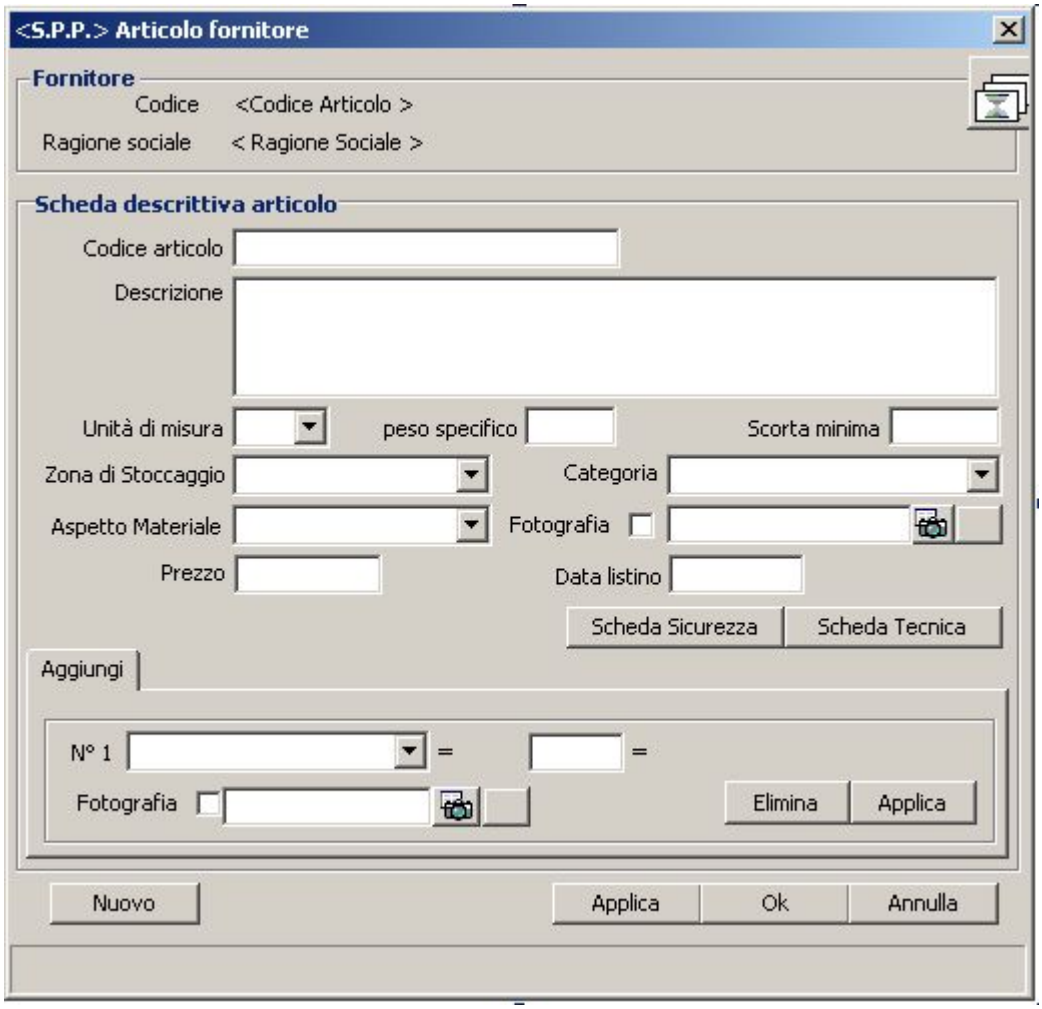


Figura 3.21 *Visualizzazzione/Creazione/Modifica Articoli Fornitori*

Funzionalità del form

Questo form (Figura 3.21), che potrebbe sembrare una parziale ripetizione del form precedente, risulta in verità molto complesso a causa delle numerose funzioni che ricopre. Si è infatti preferito dedicare un form a parte per le funzioni di creazione e modifica di un articolo, quando invece si sarebbe potuto usare, opportunamente modificato, il form precedente. Questa scelta è stata dettata da varie ragioni, anzitutto evitare modifiche non volute agli articoli (la modifica di un articolo deve essere richiesta escplicitamente dall'utente e in caso di errore l'apertura di questo form informa immediatamente l'utente dell'errore, riducendo quindi le modifiche accidentali).
Si è voluto poi mantenere il più separate possibile, anche graficamente, le varie funzioni del programma, evitando quindi di avere un form che contemporaneamente permette la visualizzazione e la immediata modifica degli articoli. Anche a livello di programmazione risulta più semplice gestire form separati, ognuno con la sua funzionalità, piuttosto che un unico form complesso che gestisce tutte le possibili operazioni.
Tramite questo form in particolare l'utente può modificare i dati relativi all'anagrafica di un articolo, nonché crearne uno nuovo. Non è possibile modificare sempre ogni dato, alcune caratteristiche, come già detto precedentemente, sono bloccate in caso l'articolo sia già stato movimentato (es il peso specifico) onde evitare incongruenze. E' possibile, ma non obbligatorio, associare uno o più contenitori, potendone posticipare la definizione al primo movimento (spesso infatti i nuovi articoli vengono inseriti in anagrafica prima di effettuare l'ordine, non sapendo ancora che contenitori sono disponibili e con i quali il prodotto verrà venduto). E' possibile ovviamente eliminare anche i vecchi contenitori.
Questo form (con i bottoni relativi al salvataggio oppurtunamente nascosti) viene utilizzato anche per la sola visualizzazione dei dati di un articolo e può essere richiamato nel caso si vogliano conoscere i dettagli anagrafici di uno specifico prodotto (es. dal form delle giacenza in cui viene presentato un elenco di tutti gli articoli e la relativa giacenza, senza dare ulteriori caratteristiche di ciascun prodotto).

Gestione delle funzioni

Nonostante le funzioni gestite siano abbastaza standard per la gestione di un magazzino, i fattori che si sono dovuti considerare rendono questo form uno dei più complessi.
Anzitutto bisogna considerare che questo form non viene richiamato esclusivamente dalla maschera precedente, in quanto da varie parti del programma è possibile modificare o aggiungere un articolo: se ad esempio si sta effettuando un movimento e ci si accorge che l'articolo desiderato non è presente è possibile aggiungerlo richiamando direttamente questo form. Per fare questo sono state usate delle property tramite le quali è possibile indicare l'articolo che si vuole creare (viene impostato il codice dell'articolo) potendo così richiamare questo form da più parti, semplicemente impostando questo e altri parametri (come ad esempio se l'articolo è in sola visualizzazione oppure in modifica).
All'apertura del form è necessario sapere se l'articolo è già stato movimentato oppure no, in modo da poter bloccare certi campi dalla modifica, come già è stato detto. Per fare questo è stata usata una Stored Procedure, richiamata direttamente dall'oggetto connection di ADO e a cui viene passato il codice relativo all'articolo interessato. Ovviamente non solo bisogna controllare i movimenti di questo articolo, ma anche quelli di ogni sua revisione, onde evitare incongruenze. Bisogna quindi prima risalire all'Id di revisione (tramite l'apposito campo nella tabella ArticoliFornitori) e successivamente controllare tutti i movimenti (tabella ArticoliFornitoriCaricati, ArticoliFornitoriScaricati, AllinementoArticoliFornitori) di questo articolo. La ricerca dei movimenti è relativamente semplice, dal momento che nelle tabelle di carico/scarico/allinemento è riportato per ogni record oltre all'Id

dell'articolo (quindi la specifica revisione che si sta movimentado) anche l'Id di revisione relativo. Questo campo (sebbene non strettamente necessario in quanto è possibile risalire all'Id di revisione tramite la tabella ArticoliFornitori) risulta utile, evitando di complicare troppo le query. Anche in questo caso si è quindi preferito introdurre un leggero overhead, nel caso specifico di spazio, per snellire il programma e rendere le operazioni più veloci.
Se viene ritrovato anche un solo movimento relativo a quell'Id di revisione viene bloccata la possibilità di modificare i valori sensibili indicati, altrimenti l'utente è libero di modificare a suo piacere ogni dato.
Gestione leggermente differente viene invece usata per i contenitori. Mentre modificare il tipo di contenitore (se sacco o cassetta ad esempio) è sempre possibile, anche se sono stati effettuati dei movimenti, non è possibile modificare la corrispondenza tra contenitore e unità di misura per evitare problemi con i vecchi movimenti che potrebbero non risultare più multipli della quantità indicata.
In linea generale si è quindi cercato di imporre all'utente la creazione di un nuovo contenitore qualora il fornitore ne modificasse uno, così come avviene nel magazzino reale.
Ovviamente sia nel caso dei contenitori che degli articoli l'eliminazione effettuata dall'utente può essere sia effettiva che "virtuale". Nel caso infatti che l'articolo non sia mai stato movimentato questo viene fisicamente rimosso dalla tabella, mentre in caso contrario viene solo impostato un flag che lo indica come non più corrente, in modo che non venga visualizzato all'utente quando richiede l'elenco degli articoli di un fornitore. Così facendo è possibile visualizzarne le caratteristiche solo partendo dai form di carico/scarico/allineamento e dal form di giacenza, come già spiegato.

3.4.4 Visualizzazione/Creazione/Modifica Scheda Tecnica e Scheda Sicurezza

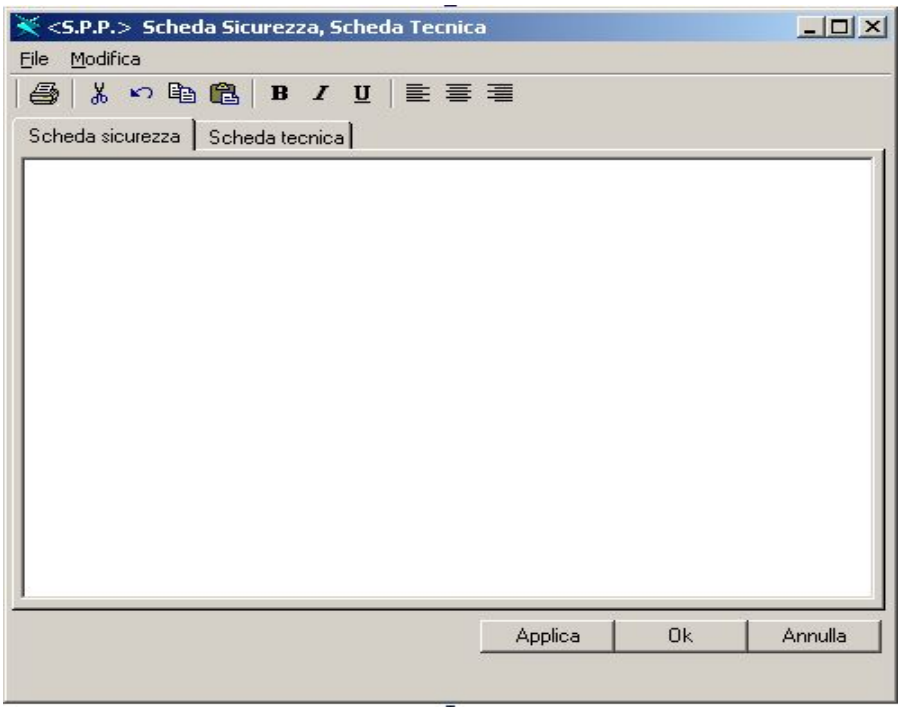


Figura 3.22: *Scheda Tecnica e scheda sicurezza*

Funzionalità del form

Questo form (Figura 3.22) rappresenta una propaggine del form precedente, gestendo funzioni che fanno parte dell'anagrafica di un articolo. Nonostante questo form dovrebbe essere parte integrante di quello precedente, si è preferito ricorrere a un nuovo form per diversi motivi. Innanzitutto queste due schede, sebbene importanti, generalmente non sono molto richieste dall'utente nell'uso normale del programma, e quindi un loro inglobamento nel form precedente avrebbe appesantito inutilmente l'interfaccia occupando spazio, costringendo ad un allargamento del form (sebbene infatti si possa effettuare il resize, un form troppo grande risulta comunque di difficile gestione per l'utente).
Tramite questa maschera l'utente può definire/modificare la scheda tecnica e la scheda di sicurezza, associate a un articolo. Essendo campi rtf (rich text format) è possibile inserire anche delle immagini e impostare una impaginazione. Generalmente queste schede vengono inserite partendo da moduli cartaceci scannerizzati e passati tramite OCR (optical caracter recognition) attraverso programmi esterni: l'utente non deve fare altro che inserire il testo desiderato e salvare. Vengono forniti tutti gli strumenti di editing necessari (copia, incolla, stampa, formattazione...) per dare massima libertà all'utente e non vincolarlo a un modulo specifico da compilare, data la varietà di casistiche presenti e alla facilità d'uso di un simile sistema.

Gestione delle funzioni

Nonostante per l'utente risulti molto comodo poter inserire del testo libero per la definizione di una scheda, questo risulta abbastanza oneroso per la base di dati, che seppure supportando come tipo di dato nativo il rich text format, si ritrova con record di dimensioni abbastanza elevate. Tuttavia avendo utilizzato una tabella a parte si è solo aumentata la dimensione della base di dati senza inficiarne le prestazioni: nella maggior parte degli accessi le schede tecniche e di sicurezza vengono infatti difficilmente consultate.
La possibilità di salvare direttamente in un campo di SQL-Server 7.0 in rtf ha semplificato notevolmente il lavoro: è sufficiente salvare nel campo della tabella ArticoliFornitoriSchedeTS il testo formattato in rtf (supportato nativamente dal controllo RichTextBox di Visual Basic) senza ulteriori complicazioni.

3.4.5 Giacenza e Scarico Magazzino

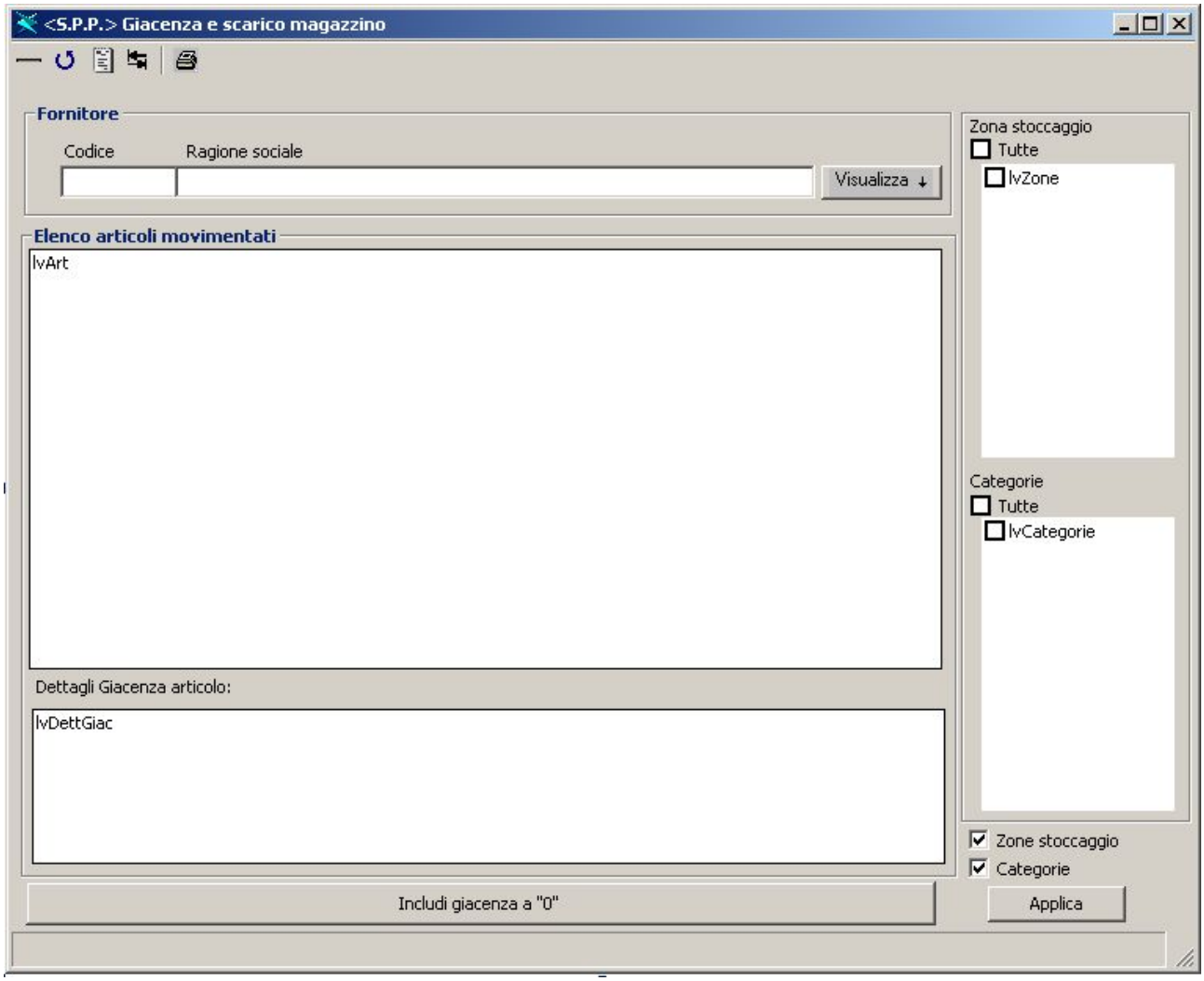


Figura 3.23: *Giacenza e Scarico Magazzino*

Funzionalità del form

Tramite questa maschera (Figura 3.23) l'utente può visionare la giacenza dei prodotti stoccati a magazzino. Come già avveniva per la visualizzazzione dell'anagrafica degli articoli è necessario procedere prima alla selezione del fornitore (il funzionamento è analogo) avendo in più la possibilità di visionare tutti gli articoli di tutti i fornitori, con un'istantanea di tutto il contenuto del magazzino.
La sezione riguardante la scelta del fornitore è volutamente simile a quello del form di visualizzazzione dell'anagrafica, onde offrire all'utente un'interfaccia di lavoro uguale per gestire funzioni equivalenti in modo da facilitare l'uso dell'applicativo. Come ulteriore strumento di ricerca l'utente può filtrare gli articoli in base alla zona di stoccaggio e alla categoria di appartenenza, avendo quindi a disposizione tutti gli strumenti necessari per individuare un articolo specifico o categorie ben distinte di prodotti, così come avviene nella gestione reale del magazzino.

Nel primo controllo vengono listati tutti i prodotti che rispondono alle caratteristiche selezionate nel filtro, con la loro giacenza totale, indipendentemente quindi dal contenitore e dalla zona di stoccaggio. Quando un articolo viene selezionato, nel controllo immediatamente sotto vengono visualizzati i dettagli della giacenza, indicando le singole quantità stoccate, raggruppate per zona e per contenitore, in modo da avere un'istantanea precisa della reale composizione del magazzino.
Tramite il bottone in basso è possibile escludere dall'elenco quei prodotti che hanno una giacenza nulla (ovvero che non sono più presenti fisicamente nel magazzino ma che erano stati movimentati in passato). In questo modo si evita un appesantimento visivo del programma, presentando dati che nella maggior parte dei casi non sono utili all'utente, permettendone però la visualizzazzione in caso questo fosse necessario.
Dopo aver selezionato un'articolo è possibile effettuare, tramite la barra degli strumenti, le operazioni di scarico e allinemento, nonchè visualizzare i dettagli anagrafici o la lista dei movimenti effettuati.

Gestione delle funzioni

La selezione del fornitore è concettualmente uguale a quanto già fatto per il form di visualizzazione, viene quindi scaricato l'elenco dei fornitori su un Recordset lato client e su di esso vengono effettuate le ricerche.
La richiesta della giacenza degli articoli deve tener conto dei parametri che possono essere stati impostati dal filtro (zona di stoccaggio, categoria, giacenza nulla, fornitore) e risulta quindi abbastanza complessa. Dal punto di vista logico si tratta di una semplice query alla base di dati, ma la possibilità di impostare molti parametri complica non poca la sua realizzazione in quanto non tutti i filtri possono essere impostati contemporaneamente, rendendo la gestione abbastanza problematica da controllare. La prima scelta era ricaduta sull'utilizzo di una Stored Procedure dedicata, opportunamente parametrizzata, che avrebbe consentito tempi di accesso ottimali e lo spostamente della logica sul server. E' stata proprio quest'ultima caratteristica che ha pesato molto in sfavore di questa soluzione, sebbene infatti la sintassi di Transact SQL sia molto completa non è certo facile effettuare dei controlli multipli sui parametri in ingresso, avendo come risultato un codice poco leggibile e sicuramente poco mantenibile.
Gestire invece da Visual Basic tutte le possibili combinazioni risulta molto più facile e comprensibile, permettendo soprattuto un debug molto più accurato. L'uso di un Recordset lato server, in sola lettura e a scorrimento in avanti (che crea fisicamente sulla base di dati un semplice cursore) non risulta così oneroso rispetto a una Stored Procedure: non devono infatti essere gestiti join complessi ma un semplice filtro sui dati. Si è quindi preferito avvantaggiare la chiarezza del codice alle prestazioni dal momento che queste non sarebbero migliorate sensibilmente.
Nella visualizzazione della giacenza per zone invece (ovvero il riquadro sottostante quello della giacenza totale) si è preferito ricorrere a una vista opportunamente creata.
Una vista è il risultato di una query SQL, viene usata come se fosse una tabella, ovvero si può dichiarare nella clausola “FROM” di uno statement SQL. Per definire una vista viene usata la sintassi standard di SQL e non quella di Transact SQL.
Dovendo usare una vista come se fosse una tabella, non è possibile specificare una clausola ”where” nella sua definizione, clausola che deve essere impostata nel momento del suo utilizzo. In questo caso specifico la vista rappresenta tutti gli articoli, con le relative giacenze, raggruppati per zone e contenitori. Nella stringa SQL che serve per aprire il Recordset utilizzato (ovviamente lato server in sola lettura e a scorrimento in avanti) viene posto il filtro che seleziona soltanto le giacenze per zona e

contenitori dell'articolo selezionato.
Come già detto precedentemente la memorizzazione esplicita della giacenza (in questo caso della tabella GiacenzaZoneArticoliFornitori) ha semplificato notevolmente il lavoro di ricavo dei dati, dovendo semplicemente interrogare la tabella della giacenza (ovviamente in join con quella degli articoli e dei contenitori per avere dati relativi appunto al contenitore utilizzato).

3.4.6 Elenco Movimenti Articoli Fornitori

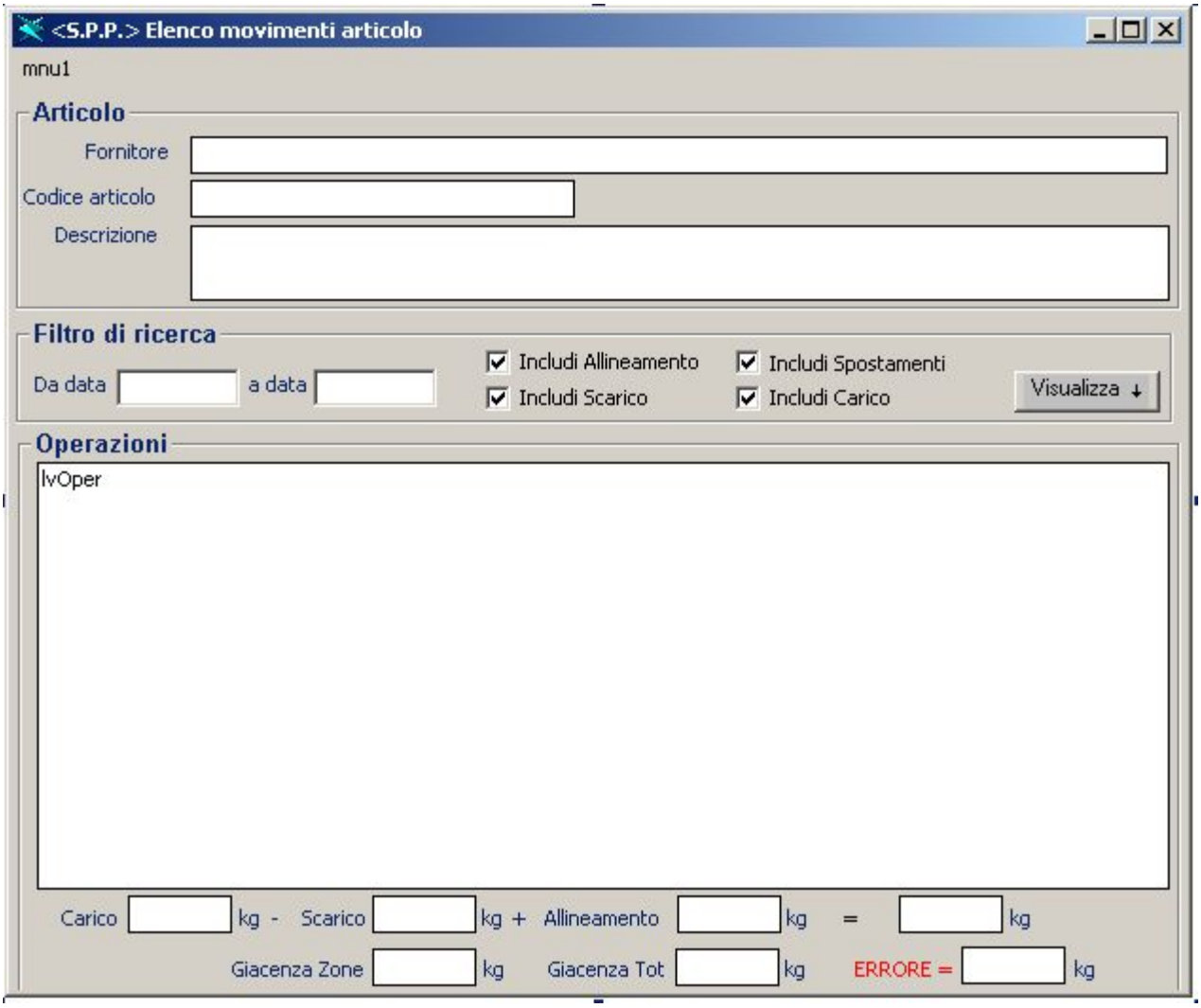


***Figura 3.24**: Elenco movimenti*

Funzionalità del form

Tramite questo form (Figura 3.24) è possibile visualizzare tutti i movimenti, da e verso il magazzino, di uno specifico articolo, selezionato nel form di giacenza. Vengono visualizzati tutti i movimenti relativi all'articolo, indipendentemente dal contenitore e dalla zona interessata, in modo da avere una panoramica generale. Per ogni singolo movimento visualizzato vengono ovviamente indicate la zona interessata e il contenitore utilizzato, mentre al di sotto vi è l'indicazione del totale di ogni singolo

movimento: totale dei carichi, degli scarichi e degli allineamenti.
L'utente ha la possibilità di visualizzare solo alcune specifiche tipologie di movimenti tramite le checkbox apposite. In questo modo si ha una panoramica generale di tutte le operazioni effettuate sull'articolo selezionato, ogni operazione con la specifica zona interessata, la data e il contenitore utilizzato.
Si è scelto di consentire di eliminare solo l'ultima operazione registrata, principalmente per due motivi. Anzitutto per problemi di coerenza: se si elimina un vecchio carico, potrebbe capitare che i successivi scarichi non abbiano sufficiente giacenza per essere effettuati. Di per se non sarebbe troppo difficile da gestire, ma se si aggiunge che è possibile effettuare degli spostamenti di un articolo da una zona all'altra, la gestione si complica enormente, dovendo di fatto controllare tutti i movimenti passati per controllare se l'eliminazione può essere effettuata.
La seconda ragione è di ordine logico: poter eliminare un movimento effettuato magari mesi addietro non dovrebbe essere permesso.

Gestione delle funzioni

Il form di giacenza, da cui questo viene richiamato, imposta tramite una property il codice dell'articolo di cui si vogliono conoscere i movimenti. Dopo aver ricavato alcuni dati anagrafici, indispensabili all'utente per riconoscere l'articolo che sta visualizzando, il form è in attesa che l'utente imposti l'intervallo temporale (di default è impostato per comprendere tra il primo del mese corrente e la data attuale) e che richieda l'elenco dei movimenti desiderati.
Anche in questo caso, come nel form di giacenza, è richiesto alla base di dati un elenco di record da visualizzare, dopo che l'utente ha impostato alcuni parametri. Nello specifico i parametri da impostare sono: un'intervallo di data e dei flag che indicano i tipi di movimenti.
Rispetto al form di giacenza però, a parte l'impostazione della data, l'unica scelta che l'utente può fare è di tipo binaria, ovvero se vuole visualizzare un movimento oppure no. Nella giacenza infatti, l'utente poteva selezionare quale zona o quale categoria di prodotti visualizzare, dovendo quindi controllare via codice le scelte dell'utente. In questo caso invece si tratta di semplici flag facilmente gestibili. Date queste premesse la scelta di usare una Stored Procedure al posto di un Recordset (come era avvenuto nel caso della giacenza) è stata quasi obbligata. Scelta resa necessaria anche dalla mole di lavoro che si trova a dover gestire: non una semplice lettura di una tabella, ma un incrocio di 4 tabelle per fornire un elenco cronologico dei movimenti.
All'utente infatti vengono presentati i movimenti ordinati in base alla data di esecuzione e non suddivisi in base al tipo di operazione. Per realizzare questo è stato necessario effettuare 4 subquery (una per ogni tipo di movimento: carico, scarico, allinemento, spostamento) in relazione tra loro tramite il comando "union all", che permette di combinare il risultato di 2 o più query in un singolo risultato contenente tutte le righe di tutte le query dell'unione. Il risultato di questa query è stato poi ordinato in base alla data, dando come prodotto l'elenco di tutte le operazioni in ordine cronologico, indipendentemente dal tipo di operazione.Ovviamente la Stored Procedure tiene conto dei parametri impostati (intervallo di data e operazioni da visualizzare) nella creazione del risultato da presentare all'utente, oltre che dover ricavare per ogni movimento il contenitore utilizzato
Possiamo notare come la complessità del lavoro che deve svolgere la base di dati è sicuramente superiore a quanto non dovesse fare per la presentazione della giacenza, rendendo quindi necessario l'utilizzo di una Stored Procedure, a fronte di una minima complessità di codice della Stored Procedure

stessa per la gestione delle opzioni impostate dall'utente.
L'eliminazione di un movimento risulta di esecuzione abbastanza agevole, salvo il dover ricavare se l'operazione che si desidera eliminare è l'ultima operazione eseguito su quell'articolo oppure no. Avendo a disposizione l'Id dell'articolo questo controllo risulta abbastanza semplice: si ricerca nelle tabelle carico/scarico/allineamento/movimenti una operazione posteriore a quella da eliminare.
Per eliminare una operazione viene rimosso fisicamente il record della tabella interessata e poi, tramite un'apposita Stored Procedure vengono aggiornate le tabelle di giacenza totale e giacenza per zona. Questa Stored Procedure viene utilizzata anche durante la registrazione dei carichi/scarichi/allineamenti/spostamenti per aggiornare la giacenza con i nuovi valori.
Per aumentare la sicurezza dell'operazione l'intero processo avviene sotto transazione, tramite l'oggetto connection. L'eliminazione del record dalla tabella infatti avviene tramite l'esecuzione di uno statemant SQL sfruttando l'oggetto connection di ADO, mentre l'aggiornamento delle giacenze (totale e per zona) avviene tramite Stored Procedure, come già detto. In questa situazione se dovesse esserci qualche problema tra le due operazioni (caduta della rete, problema al server, etc..) la base di dati si troverebbe in uno stato inconsistente, con cioè la giacenza calcolata e quella memorizzata differenti. Per ovviare a questo problema si imposta una transazione sulla connessione prima dell'eliminazione e si chiude dopo un risultato positivo della Stored Procedure.
Unico neo di questa gestione è che mentre è sotto transazione non è possibile per l'utente utilizzare l'oggetto connection, dovendo di fatto attendere il completamento delle operazioni. In effetti però questo è un falso problema in quanto innanzitutto l'intera operazione, dovendo semplicemente eliminare un record (l'operazione) e aggiornarne due (le due giacenze) è estremamente rapida, in secondo luogo per il tipo di programma: l'utente non può svolgere due operazioni contemporaneamente.

3.4.7 Carico Magazzino

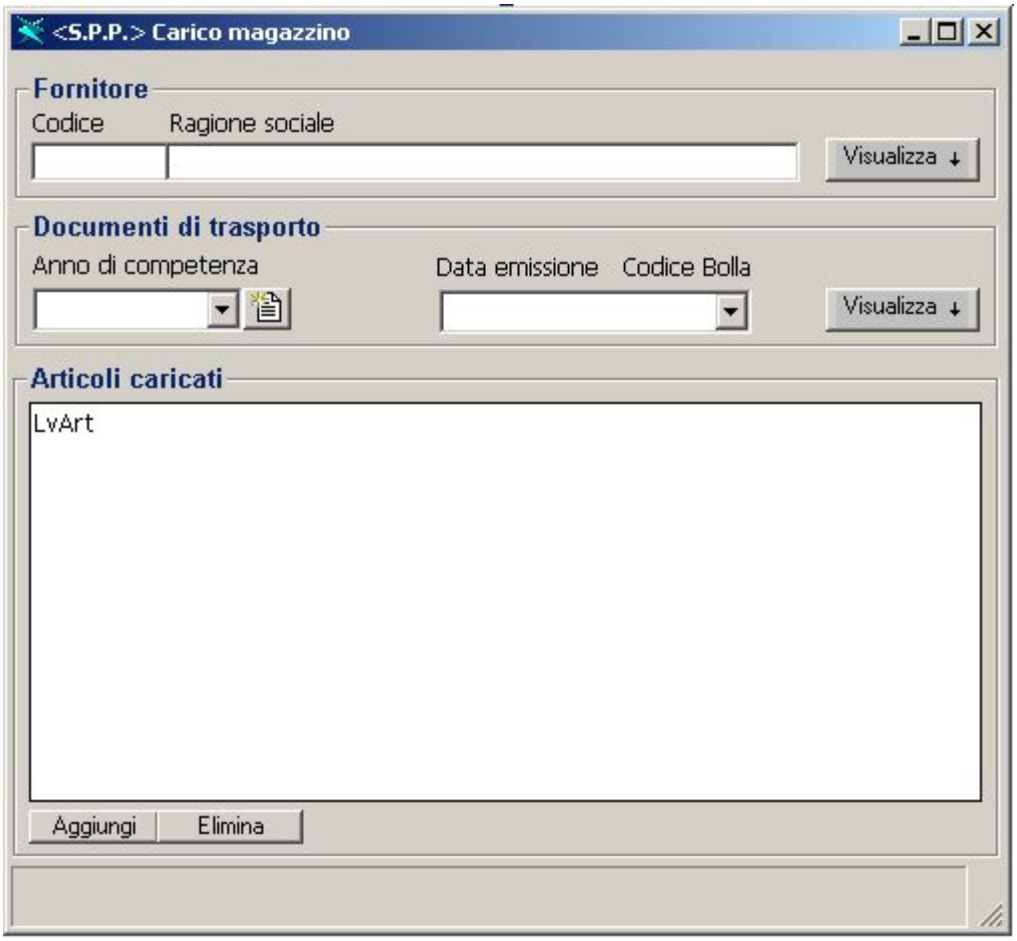


Figura 3.25: *Carico Magazzino*

Funzionalità del form

Questo form (Figura 3.25) permette all'utente di visualizzare ogni singolo DDT (Documento di Trasporto) con i relativi articoli registrati e consente di creare un nuovo DDT e di associarvi degli articoli.
La parte inerente la selezione del fornitore è la stessa di quanto già visto per l'anagrafica e la giacenza, con la possibilità quindi di selezionare direttamente il fornitore oppure di aprire un elenco completo o filtrato.
Dopo aver scelto il fornitore si procede con la selezione del DDT, raggruppati per anno di competenza e ordinati per data. E' possibile aggiungere un nuovo DDT (Figura 3.26) a quelli già presenti tramite una apposita maschera, in cui viene indicata la data e il codice che si vuole assegnare.

***Figura 3.26**: Aggiunta Documenti di Trasporto (DDT)*

Dopo aver selezionato il DDT vengono visualizzati gli articoli già registrati e caricati a magazzino, con la possibilità di aggiungere un nuovo articolo e di eliminare quelli già caricati in precedenza. In questo caso è possibile eliminare un carico (anche se non è l'ultima operazione) purchè la giacenza non risulti negativa.

Gestione delle funzioni

Come già abbiamo visto per il form di visualizzazzione della giacenza e dell'anagrafica degli articoli il filtro inerente i fornitori è gestito tramite un Recordset lato client compilato all'avvio del form, per rendere le ricerche più veloci e efficenti.
Dopo aver selezionato il fornitore e premuto il tasto “visualizza” viene aperto un Recordset lato server che ricava tutti gli anni di competenza relativi a quel fornitore, ovvero tutti gli anni in cui sono stati effettuati dei DDT. Questa gestione, come già era successo con l'elenco degli articoli, permette di ottimizzare l'utilizzo delle risorse della base di dati. In questo caso a maggior ragione la semplice richiesta di un elenco di anni è estremamente veloce e performante e non richiede molto lavoro alla base di dati: una volta ricavati i dati questi vengono caricati nel controllo comboBox relativo. Rispetto al filtro relativo ai fornitori infatti in questo caso è presente un controllo che memorizza al suo interno i dati necessari, senza dover quindi ricorrere a un Recordset disconnesso.
Sempre con la stessa logica sono gestiti anche i numeri di DDT che vengono caricati anch'essi tramite un Recordset lato server all'interno di un controllo comboBox.
L'utilizzo di due controlli separati “in cascata”, uno per i soli anni di competenza, l'altro per i DDT, rispecchia una esigenza dell'utente che ha tutti i DDT suddivisi per anno e che quindi vorrebbe avere la stessa distinzione precisa anche tramite questo programma. Effettuare due query successive (una solo per ottenere gli anni e una per avere i DDT relativi a quell'anno) invece di una sola che ricavava tutti i DDT di tutti gli anni risulta migliore in quanto, nonostante si aggiunge un certo overhead dovuto alla doppia query, si evita di scaricare molti dati inutili. Nella maggior parte dei casi infatti all'utente interessano soltanto i DDT dell'ultimo anno e quindi scaricare completamente tutto l'archivio risulta insensato.
I dettagli di un DDT (ovvero gli articoli caricati) vengono prelevati anch'essi tramite un Recordset lato server e memorizzati all'interno di un controllo listView, con i relativi contenitori utilizzati.Questa gestione è un compromesso di tutti i possibili modi con cui l'utente potrebbe usare questa maschera: se da un lato infatti l'uso di tre Recordset lato server velocizza richieste molto differenti tra loro (fornitori diversi, anni diversi etc) dall'altro penalizza leggemente accessi a dati simili (es DDT in sequenza

temporale) dove sarebbe più performante scaricare direttamente tutto l'archivio inerente un fornitore.

3.4.8 Carico Articolo

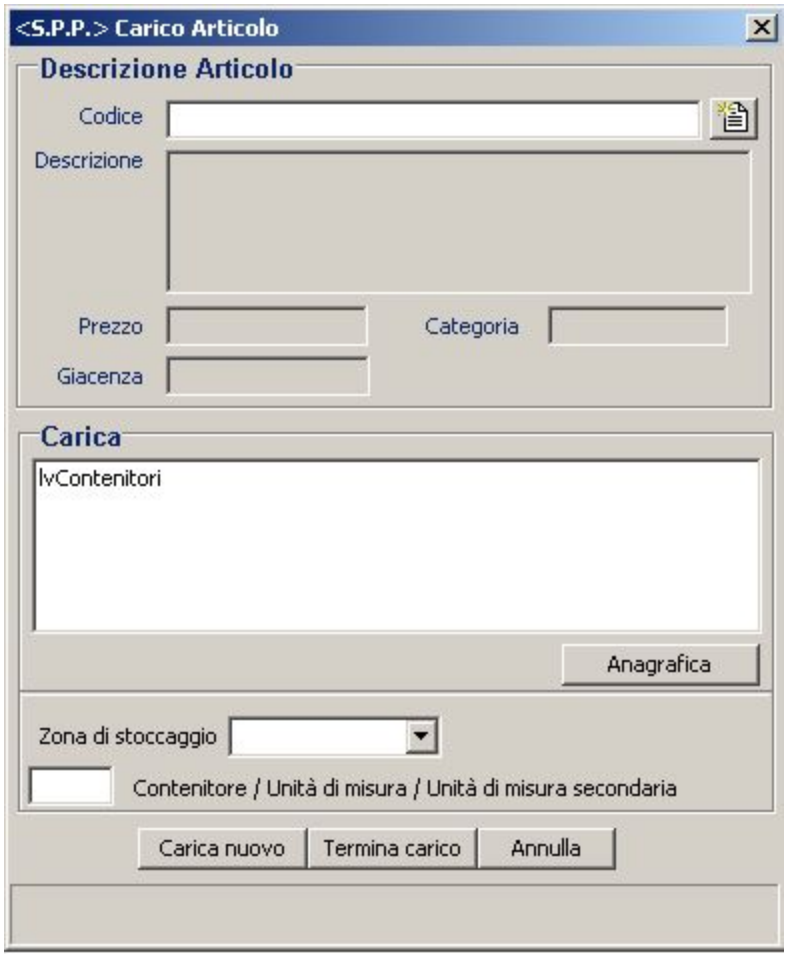


***Figura 3.27**: Carico Articolo*

Funzionalità del form

Assieme al form di scarico e di allineamento questo rappresenta una delle maschere più utilizzate di tutto il programma, permettendo di registrare operazioni di carico a magazzino (Figura 3.27).
Questo form viene richiamato da quello di carico magazzino, dove è stato indicato il fornitore del prodotto e il DDT interessato. A questo punto l'utente deve solo selezionare l'articolo da caricare, il contenitore con il quale si presenta e la quantità (oltre alla data ovviamente).Tramite l'apposita casella si può immettere un parametro di ricerca, con funzionalità simile alla scelta del fornitore visto già altre volte: se viene immessa una stringa e si preme [F9] verrà visualizzato un elenco con tutti i prodotti che rispondono al filtro impostato. Se l'articolo non è presente nell'elenco, si può aggiungerlo in anagrafica tramite l'apposito bottone: verrà visualizzato il form di definizione dell'articolo.
Dopo aver selezionato l'articolo verranno visualizzati nel controllo listView preposto i contenitori

associati (tramite il bottone “anagrafica” viene aperto il form di definizione dell'articolo in modalità “modifica” che permette di modificare l'anagrafica). Dopo aver selezionato la quantità (espressa in multipli di contenitori) e la zona di destinazione, l'utente potrà effettuare un nuovo carico(con un nuovo articolo) oppure terminare l'operazione

Gestione delle funzioni

All'avvio del form viene caricato un Recordset lato client(disconnesso) contenente tutti gli articoli del fornitore selezionato. In questo modo è possibile effettuare delle ricerche, tramite la stringa immessa dall'utente nell'apposita casella di testo, direttamente in locale, senza dover interrogare la base di dati. La query di creazione di questo Recordset ricava solo i dati anagrafici che devono essere visualizzati all'utente (anche la giacenza totale ad esempio) mentre non vengono scaricati i contenitori associati, ricavati solo dopo che l'utente ha confermato l'articolo.
In questo modo il carico della base di dati viene suddiviso in due query successive: prima viene ricavato l'elenco degli articoli, con solo dati anagrafici, che serve per la ricerca in locale del prodotto desiderato e successivamente, dopo aver scelto l'articolo, vengono scaricati i soli contenitori disponibili per quel prodotto. Oltre all'alleggerimento del lavoro della base di dati (evitando di scaricare dati inutili) è possibile avere un miglior controllo sul lavoro che sta svolgendo l'utente: caricando i contenitori in un secondo momento è infatti facilmente controllabile se l'articolo selezionato ha associato un contenitore, in caso contrario l'utente viene avvisato e gli viene data la possibilità, aprendo il form di modifica dell'articolo, di aggiungere un contenitore.
Tramite il passaggio di un valore di ritorno il form di modifica indica se sono state fatte delle modifiche anche ai dati anagrafici, in caso negativo è sufficiente scaricare nuovamente i contenitori (appena aggiunti dall'utente) e visualizzarli tramite l'apposito controllo.
Nella maggior parte dei casi infatti, per gli articoli nuovi, non è presente nella definizione dell'articolo nessun contenitore che di solito viene aggiunto al momento del carico. Si può quindi facilmente notare come questo semplice accorgimento permetta di evitare ogni volta di scaricare tutti gli articoli e i relativi contenitori per ogni modifica apportata.
Come già visto per l'eliminazione di un movimento nel form di elenco dei movimenti, anche in questo caso le operazioni di carico vengono effettuate sotto transazione.
La transazione viene avviata sulla connessione disponibile e subito dopo viene aperto un Recordset lato server che aggiunge il movimento alla tabella dei carichi. L'aggiornamento delle giacenze viene invece effettuato tramite l'apposita Stored Procedure, già discussa precedentemente.
Le operazioni da e verso il magazzino sono state infatti implementate tutte con lo stesso criterio: Recordset lato server per aggiornare la tabella dei movimenti (di volta in volta differente ovviamente) e Stored Procedure per aggiornare le due giacenze (totale e per zona).
La Stored Procedure procedure quindi viene utilizzata in vari punti del programma, consentendo quindi il riutilizzo del codice e la manutenzione del programma in fase di debug.
Le zone di stoccaggio sono ricavate tramite un Recordset lato server che viene chiuso subito dopo aver compilato il combobox relativo.
Dopo aver terminato le operazioni di carico, viene ritornato al form di carico magazzino un flag che indica se sono stati aggiunti articoli, e quindi se devono essere riscaricati i valori relativi a quel DDT oppure se l'operazione è stata annullata.

3.4.9 Scarico Articolo

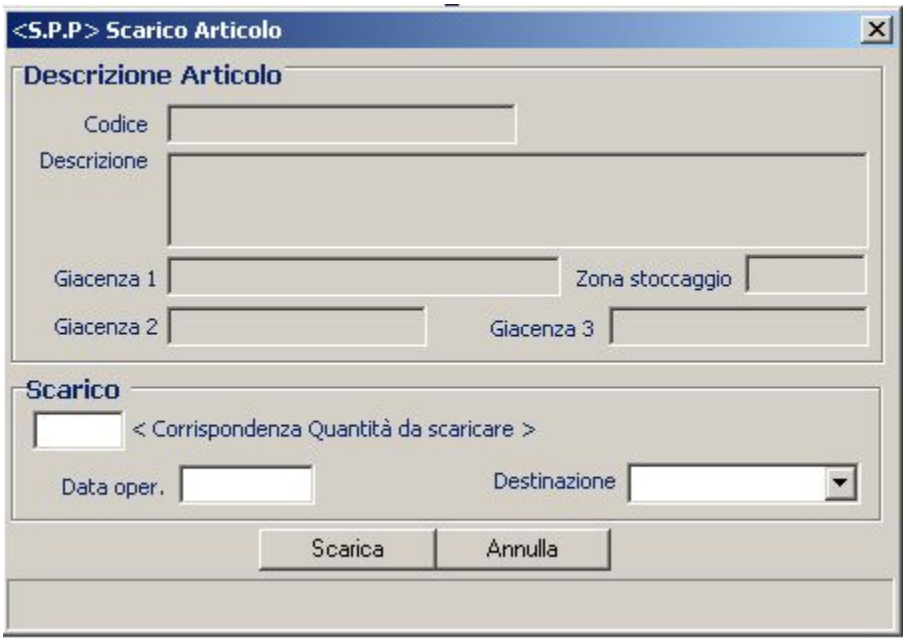


***Figura 3.28**: Scarico Articolo*

Funzionalità del form

Tramite il form di giacenza l'utente ha già selezionato l'articolo che deve essere scaricato con il relativo contenitore (ovvero tramite il controllo listView in basso).
In questo form (Figura 3.28) quindi non resta che indicare la quantità (espressa ovviamente in multipli interi del contenitore selezionato), la data e la destinazione di impiego (dove verrà utilizzato il materiale).
Le tre caselle di testo Giacenza 1, 2 e 3 rappresentano la giacenza totale dell'articolo, nella specifica zona indicata, espressa rispettivamente in contenitori (ovvero il numero intero di contenitori disponibili) in unità di misura principale e in unità di misura secondaria.

Gestione delle funzioni

Così come avviene in tutti i form in cui l'utente deve indicare la quantità da movimentare, anche in questo sorge il problema della gestione dei valori numerici. Come già accennato precedentemente l'utente indica il numero (intero) di contenitori da movimentare, ma le effettive operazioni vengono registrate in unità di misura principale, indipendentemente dal contenitore utilizzato. Questo è necessario in quanto è possibile associare più contenitori ad un solo articolo, contenitori con capacità ovviamente differente di volta in volta: sarebbe quindi impossibile avere una tabella di giacenza totale, ma si avrebbe solo una tabella di giacenza per zona, dovendo calcolare direttamente ad ogni richiesta la giacenza totale.

Molti articoli (ad esempio i liquidi) non possono avere associato un contenitore preciso con il quale essere movimentati (vengono infatti acquistati in cisterne molto grandi o addirittura vengono versati nei silos presenti in azienda) così come anche per molti prodotti in polvere.
In effetti è questo l'unico vero inconveniente dell'uso di contenitori per effettuare i movimenti: non è possibile associare un contenitore ad ogni articolo presente a magazzino. In questi casi si procede quindi alla movimentazione tramite dei contenitori fittizzi, di capacità unitaria per sopperire al problema: sono comunque allo studio (in collaborazione con il cliente) delle soluzioni per risolvere il problema.
L'uso dei contenitori, anche se potrebbe sembrare a prima vista macchinoso sia per l'utente che per il programmatore, è risultato estremamente semplice per l'utente, che dopo aver compilato l'anagrafica si trova a lavorare con quantità intere, così come lavora nella realtà (le richieste al magazziniere da parte degli utilizzatori finali sono generalmente espresse in contenitori).
Anche se di difficile soluzione, i casi di liquidi o polveri senza contenitori sono molto bassi (meno del 2%) e quindi gestire queste eccezioni risulta abbastanza agevole.

3.4.10 Allinemento Giacenza/Modifica zone di Stoccaggio

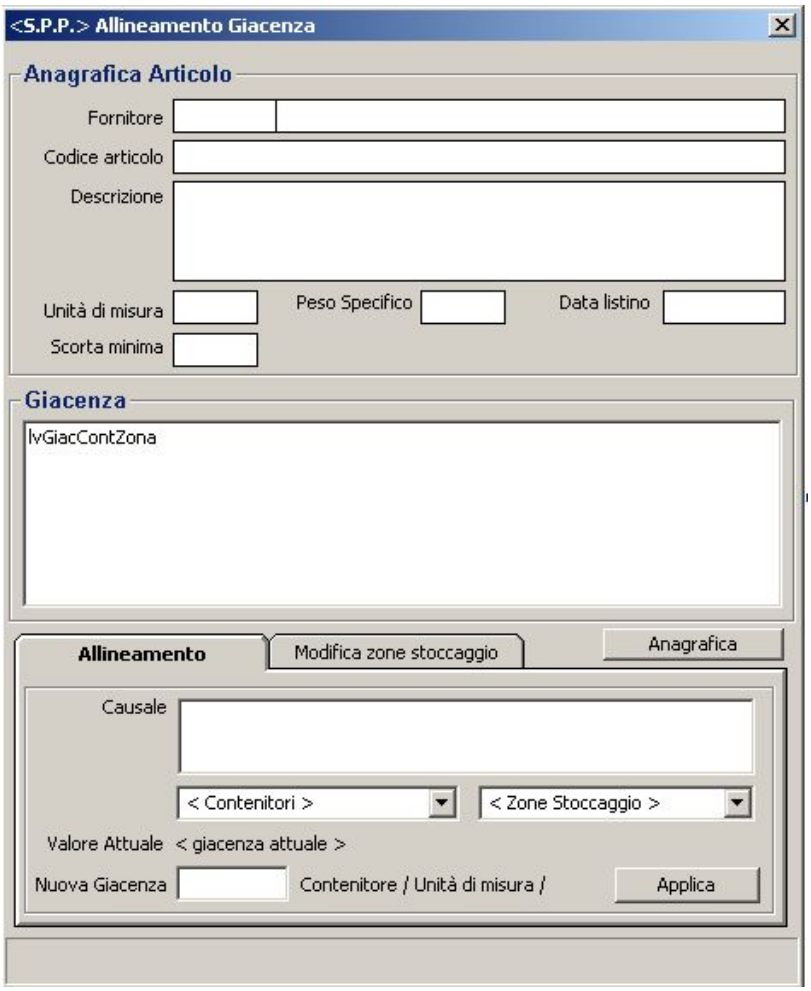


***Figura 3.29**: Allineamento e modifica zone stoccaggio*

Funzionalità del form

In questa maschera (Figura 3.29) vengono raccolte due funzioni di manutenzione del magazzino: l'allineamento della giacenza e lo spostamento di articoli da una zona all'altra del magazzino.
L'operazione di allineamento è del tutto straordinaria e si ha quando, per varie ragioni, si scopre che la quantità di un prodotto effettivamente stoccata a magazzino è differente da quanto registrato. Ciò potrebbe essere causato dalla mancata registrazione di un movimento da parte dell'utente, che si trova con la giacenza reale non allineata con quella registrata, oppure dall'aver registrato una quantità errata. Nella maggior parte dei casi è possibile risalire al movimento mancante e registrarlo normalmente, tuttavia alcune volte ciò non è possibile (soprattutto per i movimenti di scarico) e quindi si rende necessaria questa operazione. L'utente è tenuto a fornire una causale dell'allineamento e in ogni momento è possibile visualizzare questi movimenti.
Sebbene questa operazione risulti molto rara, in alcuni casi è necessaria. E' ancora una volta il caso dei liquidi o delle polveri, dove alcuni errori manuali (es un secchio che si rovescia) difficilmente vengono registrati, rendendo difficile ricostruire correttamente la giacenza.
L'altra operazione registrabile è lo spostamento di quantità da una zona all'altra del magazzino, dovuto alla riorganizzazione delle zone.

Gestione delle funzioni

Questo form può essere aperto sia dall'anagrafica di un articolo sia dalla giacenza. All'avvio, come già visto in altri form, viene caricata l'anagrafica tramite un Recordset lato server, chiuso immediatamente dopo aver recuperato i dati e aver compilato i controlli relativi. All'interno del listView centrale viene visualizzata la situazione a magazzino dell'articolo in modo dettagliato: vengono mostrate tutte le zone in cui l'articolo è stoccato con i relativi contenitori utilizzati, in modo da offrire all'utente una visuale chiara della situazione reale.
Questi dati vengono ricavati tramite un Recordset lato server che interroga la tabella della giacenza per zone e quella dei contenitori, per ricavare le informazioni inerenti al contenitore utilizzato.
Dopo aver selezionato una particolare giacenza all'interno del listView, all'utente viene data la possibilità di modifcare la quantità realmente presente, allineanto così la giacenza reale con quella registrata.
Per fare questo associato al listView della giacenza ci sono due Recordset, uno per memorizzare la quantità in giacenza in quella specifica zona e l'altro per memorizzare il contenitore utilizzato (e quindi la corrispondenza). Anche se sarebbe stato possibile utilizzare un solo Recordset associato al listView delle giacenze, è risultato più comodo usare due Recordset separati, in modo da poter aggiornare anche della giacenze inesistenti. Se infatti in una particolare zona non è presente un certo articolo con una certa giacenza, deve essere possibile aggiungere una quantità in quella zona. Con l'uso di due Recordset l'operazione risulta facilitata.
Ogni volta che l'utente modifica la selezione all'interno del listView delle giacenze devono essere modificati anche i comboBox relativi alle zone di stoccaggio e dei contenitori, in modo da avere tutti i valori presentati all'utente omogenei tra loro. L'utente può ovviamente selezionare una coppia contenitore-zona in cui la giacenza sia nulla (e che quindi non risulta inserita all'interno del listView delle giacenze) in modo da poter allineare una quantità effettivamente nulla.

Ogni volta che l'utente modifica la selezione all'interno del listView viene aggiornata anche la parte inerente lo spostamento di quantità da una zona all'altra.In entrambi il cambiamento della selezione nel listView provoca innanzitutto la ricerca all'interno del Recordset della giacenza e dei contenitori il record scelto(tramite il metodo find di ADO applicato ai Recordset); dopo aver ricavato quindi il contenitore e la zona interessata (sotto forma di indice ovviamente) vengono allineati i comboBox relativi. Risulta ovviamente chiaro come l'intero procedimento sia molto delicato e abbia necessitato di una accurata fase di testing.
L'operazione di allineamento risulta abbastanza semplice da gestire, l'utente infatti indica semplicemente la nuova quantità presente in quella zona e il programma provvede a modificare la giacenza totale e quella di zona tramite l'apposita Stored Procedure già vista altre volte. Per la registrazione del movimento effettuato (da registrare sulla tabella AllineamentoArticoliFornitori) è sufficiente eseguire una semplice sottrazione tra il valore inserito dall'utente e quello realmente presente a magazzino. Quest'ultima operazione viene eseguita tramite un Recordset lato server, nella stessa transazione aperta precedentemente per modificare i dati di giacenza.
Per quanto riguarda la registrazione di uno spostamento tra una zona e l'altra è necessario introdurre un controllo aggiuntivo sui dati: non è possibile spostare da una zona una quantità superiore a quella in giacenza in quella zona specifica. L'intera operazione di spostamento viene effettuata tramite un'apposita Stored Procedure che provvede a modificare la tabella di giacenza per zona e a registrare il movimento. L'uso di una Stored Procedure unica, e non come negli altri casi dell'accoppiamento Stored Procedure e Recordset lato server sotto transazione, si è reso necessario per la complessità delle operazioni da svolgere. Per modificare la giacenza nella tabella giacenza per zona infatti è necessaria la modifica di due record all'interno di una stessa tabella, operazione certamente semplice ma che prevede una complessità maggiore a livello di codice se espressa in SQL standard mentre risulta più agevole se scritta in T-SQL. Questa Stored Procedure inoltre viene utilizzata anche per l'eliminazione dei movimenti (form dettagli movimenti) e quindi risulta più economico adottare un'unica Stored Procedure da richiamare in due punti

3.5 Stored Procedure

Le stored procedure sono utili nello sviluppo di una applicazione che si deve interfacciare a una base di dati. Esse aiutano a separare l'applicazione client dalla sottostante struttura dati e a semplificare la scrittura del codice aumentando la stabilità e la scalabilità dell'applicazione. Attraverso le stored procedure è possibile creare codice in grado di essere veramente riutilizzabile.
A livello applicativo i punti a favore dell'uso delle stored procedure nello sviluppo dei programmi data-driven sono due:

1. Sicurezza dei dati
2. Velocità di esecuzione

Le stored procedure facilitano l'implementazione della sicurezza dei dati della base di dati. Se infatti vengono assegnati dei diritti di esecuzione su una procedura ad utenti o gruppi non è necessario assegnare gli stessi privilegi a tutti gli oggetti (tabelle, viste, ecc…) chiamati all'interno della procedura. Per questo motivo è molto efficace includere i report e le query che interessano all'interno delle stored procedures ed assegnare i privilegi di accesso solamente agli utenti interessati alla loro visualizzazione.
Sempre attraverso le stored procedures è possibile aggiungere, modificare o eliminare i dati.
Queste operazioni così delicate possono essere blindate includendo nelle stored procedures le istruzioni per aggiungere, modificare o eliminare i dati ed assegnando agli utenti e ai gruppi autorizzati i diritti di esecuzione esclusivamente sulle stored procedures e non direttamente sugli oggetti che vengono richiamati al loro interno.

In particolare sono state utilizzate Stored Procedure, per ragioni di sicurezza, nel momento in cui era necessario effettuare operazioni su più tabelle in sequenza, potendo quindi racchiudere una transazione, gestita completamente dalla base di dati, in un'unica Stored Procedure.

Le stored procedures aumentano enormemente le performance dei programmi perchè sono pre-compilate e quindi eseguite più rapidamente. Per ognuna di esse SQL Server genera un query plan contenente il metodo più efficiente di esecuzione della procedura il quale si basa su differenti informazioni come indici disponibili, costi I/O ed altri parametri ambientali. Una volta calcolato il miglior query plan possibile SQL Server lo salva nella memoria cache e lo riutilizzerà ogni qualvolta verrà richiamata la procedura.
Altro beneficio importante derivante dall'impiego delle stored procedure è il tempo minimo del lock dei dati durante l'esecuzione rispetto all'equivalente tempo se le istruzioni SQL venissero inviate dall'applicazione client. Questa caratteristica è particolarmente utile nel caso di trasferimento di grandi quantità di dati (es: la richiesta di tutti i movimenti inclusi in un dato periodo) in cui generalmente si fanno più ricerche mirate in sequenza, riducendo i tempi medi di risposta.

3.5.10 Stored Procedure utilizzate

Introduzione

All'interno del programma le Stored procedure sono state utilizzate, come già detto in precedenza, per ragioni di sicurezza e stabilità in associazione con le transazioni, attivate tramite l'oggetto connection alla base di dati.

Transazione di Carico

Le operazioni di carico a magazzino vengono effettuate all'interno di una transazione, iniziata all'interno del programma scritto in Visual Basic, continuata tramite una Stored Procedure e terminata di nuovo tramite Visul Basic. L'oggetto “connection”, che gestisce la connessione alla base di dati permette di iniziare una transazione tramite il comando “BeginTrans” e di terminarla tramite il comando ”CommitTrans”. Vediamo nel dettaglio il codice

```
'transazione che gestisce il carico di un articolo su un DDT
Private Function TransazioneCarico() As Boolean

   Dim s As String
   Dim rs As ADODB.Recordset
```

In caso di errori viene richiamata la procedure preposta alla loro gestione

```
   On Error GoTo gestErr

   s = "SELECT * FROM ArticoliFornitoriCaricati WHERE 1 = 0"
```

Viene aperta la transazione

```
   '//// INIZIO TRANSAZIONE \\\\
   m_Conn.BeginTrans

   Set rs = New ADODB.Recordset
   rs.CursorLocation = adUseClient
   rs.Open s, m_Conn, adOpenStatic, adLockOptimistic, adCmdText
   rs.AddNew
   rs("AFCIdDTF").value = m_idDDT
   rs("AFCIdRevAF").value = RecordsetArt("AFRevId").value
   rs("AFCIdArFo").value = RecordsetArt("AFId").value
```

```
rs("AFCQt").value = CaricaUMP
rs("AFCDatCar").value = Format(Now, GenF.DateFormat + " hh.mm.ss")
rs("AFCPreUni").value = RecordsetArt("AFPreUni").value
rs("AFCIdCPA").value = Mid(lvContenitori.SelectedItem.key, 2)
rs("AFCIdZSAF").value = cbZona.ItemData(cbZona.ListIndex)
```

Viene aggiornata la tabella relativa i carichi, inserendo il movimento appena effettuato tramite il RecordSet preposto

```
rs.Update
rs.Close: Set rs = Nothing

If CmdGiac Is Nothing Then
   Set CmdGiac = New ADODB.Command
```

Viene richiamata la Stored Procedure preposta all'aggiornamento delle tabella "GiacenzaPerZona" e "GiacenzaTotale"

```
   With CmdGiac
      .ActiveConnection = m_Conn
      .CommandText = "sp_Update_GiacenzaArticoliFornitori"
      .CommandType = adCmdStoredProc
      .Parameters.Append .CreateParameter("IdArt", adInteger, adParamInput)
      .Parameters.Append .CreateParameter("IdCPA", adInteger, adParamInput)
      .Parameters.Append .CreateParameter("IdZSAF", adInteger, adParamInput)
      .Parameters.Append .CreateParameter("Qt", adDecimal, adParamInput)
      .Parameters(3).Precision = 18
      .Parameters(3).NumericScale = 6
      .Parameters.Append .CreateParameter("Oper", adSmallInt, adParamInput)
      .Parameters.Append .CreateParameter("NuovaGiacenzaTotale", adDecimal, adParamOutput)
      .Parameters(5).Precision = 18
      .Parameters(5).NumericScale = 6
      .Parameters.Append .CreateParameter("NuovaGiacenzaZona", adDecimal, adParamOutput)
      .Parameters(6).Precision = 18
      .Parameters(6).NumericScale = 6
   End With
End If

'Tramite l'apposita SP aggiorna la tabella GiacenzaArticoliFornitori
'Nella tabella della giacenza gli articoli sono raggruppati secondo lo
'stesso id di revisione, in modo da avere la giacenza complessiva di
'tutte le revisioni dell'articolo.
'L'Id di revisione dell'articolo viene ricavato dalla SP.
'serve la quantità caricata espressa in unità di misura principale
With CmdGiac
   .Parameters("IdArt").value = RecordsetArt("AFId").value
```

```
      .Parameters("IdCPA").value = Mid(lvContenitori.SelectedItem.key, 2)
      .Parameters("IdZSAF").value = cbZona.ItemData(cbZona.ListIndex)
      .Parameters("Qt").value = Abs(CaricaUMP)
      .Parameters("Oper").value = 1
      .Execute
   End With
```

Se tutto è andato a buon fine, viene effettuato il “Commit”

```
   m_Conn.CommitTrans
   '\\\\FINE TRANSAZIONE////

   TransazioneCarico = True



Exit Function
```

In caso di errori viene mostrato un avviso all'utente e viene effettuato il “RollBack” della transazione.

```
gestErr:
   m_Conn.RollbackTrans
   MsgBox "Errore durante il salvataggio dei dati sul DDT!" & vbNewLine & _
         err.Description, vbCritical, GenF.AT3
   If GenF.IDEMode Then Stop: Resume Else Unload Me

End Function
```

Vediamo ora il codice della transazione che aggiorna le due tabelle relative alla giacenza:

```
/*
 In base ai parametri in ingresso aggiunge o toglie la quantità specificata dalla giacenza del singolo articolo;
se l'articolo non esiste allora viene aggiunto alla tabella GiacenzaArticoliFornitori, mentre se non esiste e l'operazione è uno scarico allora restituisce -1; se lo
scarico è maggiore della giacenza allora restituisce -2
 */

CREATE Procedure dbo.sp_Update_GiagenzaArticoliFornitori
                  @IdArt Int,
                  @Qt Real,
                  @Oper smallInt,
                  @NuovaGiacenza Real OUTPUT
As
Set nocount on
```

```
DECLARE @IdRev as INT, @PreUni as REAL

/*
Viene ricavato l'Id di revisione dell'articolo
*/
SELECT TOP 1 @IdRev = AFRevId, @PreUni = AFPreUni FROM ArticoliFornitori
WHERE AFId = @IdArt

/*
Nel caso in cui l'indice di revisione non sia stato associato alla riga, viene utilizzato l'Id di riga stesso
Questa condizione non si deve verificare e provocherebbe un errore nella SP; la riga di codice seguente
ripara l'errore
*/
SET @IdRev = ISNULL(@IdRev,@IdArt)
SET @PreUni = ISNULL(@PreUni,0)

/*giacenza attuale*/
Declare @NowQt Real

SELECT @NowQt = -1


Tag1:
SELECT TOP 1 @NowQt = GAFQt FROM GiacenzaArticoliFornitori
WHERE GAFIdRevAF = @IdRev

/* se l'articolo non esiste */
if @NowQt = -1
  begin
  /* se l'operazione è uno scarico */
  if @Oper = -1
    Begin
    /* esce avvertendo dell'errore */
    SELECT @NuovaGiacenza = -1
    RETURN
    end
   else
    begin
    /* se l'operazione è un carico crea il nuovo articolo con i prezzi di listino */
    INSERT INTO GiacenzaArticoliFornitori  (GAFIdRevAF, GAFQt, GAFPreUni, GAFDaUlOp)
    VALUES (@IdRev, 0,  0, GetDate())
    /* ora che l'articolo è in giacenza è possibile ri-eseguire la ricerca dell'articolo */
    goto tag1
    end
  end
```

```
if (@NowQt < @Qt) AND (@Oper < 0 )
  begin
  SELECT @NuovaGiacenza = -2
  RETURN
  end

/*
 A questo punto una giacenza deve per forza esistere nella tabella;
viene quindi aggiornata la quantità e il prezzo
*/

UPDATE GiacenzaArticoliFornitori
SET GAFQt = GAFQt + (@Qt * @Oper),
     GAFPreUni = GAFPreUni + (@PreUni * @Qt * @Oper),
     GAFDaUlOp = GetDate()
WHERE GAFIdRevAF = @IdRev


/*
Restituisce il valore della nuova giacenza
*/
SELECT TOP 1 @NuovaGiacenza = GAFQt FROM GiacenzaArticoliFornitori
WHERE GAFIdRevAF = @IdRev
```

Transazione di Scarico

Specularmente alla gestione dei carichi a magazzino anche gli scarichi vengono effettuati registrando i movimenti sulla tabella ArticoliFornitoriScaricati tramite un RecordSet e modificando la giacenza tramite un'apposita Stored Procedure.

```
Private Sub TransazioneScarico()

   Dim s As String
   Dim rs As New ADODB.Recordset
   rs.CursorLocation = adUseServer
   'indica se l'errore è stato scatenato dalla SP
   Dim ErrSP As Boolean

   On Error GoTo gestErr
```

Viene attivata la transazione

```
   '///INIZIO TRANSAZIONE\\\
   m_Conn.BeginTrans
```

```
'l'errore è nella SP
ErrSP = True

'aggiornamento della tabella GiacenzaArticolifornitori e GiacenzaPerZoneAF
Dim Cmd As New ADODB.Command
With Cmd
   .ActiveConnection = m_Conn
   .CommandText = "sp_Update_GiacenzaArticoliFornitori"
   .CommandType = adCmdStoredProc
   'l'Id di revisione dell'articolo viene ricavato dalla SP
   .Parameters.Append .CreateParameter("IdArt", adInteger, adParamInput, , m_IdArt)
   .Parameters.Append .CreateParameter("IdCPA", adInteger, adParamInput, , m_IdCont)
   .Parameters.Append .CreateParameter("IdZSAF", adInteger, adParamInput, , m_idZSAF)
   .Parameters.Append .CreateParameter("Qt", adDecimal, adParamInput, , Abs(ScaricaUMP))
   .Parameters(3).Precision = 18
   .Parameters(3).NumericScale = 6
   .Parameters.Append .CreateParameter("Oper", adSmallInt, adParamInput, , -1)
   .Parameters.Append .CreateParameter("NuovaGiacenzaTotale", adDecimal, adParamOutput)
   .Parameters(5).Precision = 18
   .Parameters(5).NumericScale = 6
   .Parameters.Append .CreateParameter("NuovaGiacenzaZona", adDecimal, adParamOutput)
   .Parameters(6).Precision = 18
   .Parameters(6).NumericScale = 6
```

Viene eseguita la Stored Procedure che aggiorna entrambe le tabelle relative alla giacenza, dopo aver impostato i parametri necessari e aver specificato le variabili.

```
   .Execute
   If Not IsNull(.Parameters("NuovaGiacenzaTotale").value) Then
      NewTotGiac = .Parameters("NuovaGiacenzaTotale").value
   Else
      'se viene ritornato un valore null significa che si è verificato un errore
      GoTo gestErr
   End If
   If Not IsNull(.Parameters("NuovaGiacenzaZona").value) Then
      NewGiacZona = .Parameters("NuovaGiacenzaZona").value
   Else
      'se viene ritornato un valore null significa che si è verificato un errore
      GoTo gestErr
   End If
End With
Set Cmd = Nothing

'se la giacenza è un valore negativo, c'è stato un problema e l'operazione viene annullata
```

```
If (NewTotGiac < 0) Or (NewGiacZona < 0) Then GoTo gestErr

'da adesso l'errore non è più della SP
ErrSP = False

s = "SELECT * FROM ArticoliFornitoriScaricati WHERE 1 = 0"
rs.Open s, m_Conn, adOpenDynamic, adLockOptimistic, adCmdText
rs.AddNew
rs("AFSIdRevAF").value = m_IdRevArt
rs("AFSQt").value = Abs(ScaricaUMP)
rs("AFSData").value = CDate(txSDataOp.Text)
rs("AFSDataReg").value = Format(Now, GenF.DateFormat + " hh.mm.ss")
'il rs viene posizionato sulla causale correntemente visualizzata
'è già stato effettuato il controllo sulla correttezza del contenuto del
'comboBox delle causali di scarico
s = "DIPDescri = '" & CbDesSca.Text & "'"
RsDesSca.Find s, 0, adSearchForward, adBookmarkFirst
rs("AFSCodDIP").value = RsDesSca("DIPFlag").value
rs("AFSIdCPA").value = m_IdCont
rs("AFSIdMaga").value = m_ClsMagInt.p_IdMagazzinoInt
rs("AFSIdZSAF").value = m_idZSAF
```

Viene inserito il movimento all'interno della tabella degli scarichi

```
rs.Update
rs.Close: Set rs = Nothing

m_Conn.CommitTrans
'\\\FINE TRANSAZIONE///

Exit Sub
```

In caso di errori viene avvisato l'utente e viene effettuato il RollBack di tutta la transazione. Da notare il differente messaggio di errore nel caso l'errore sia stato generato all'interno della Stored Procedure oppure sia stato causato dal codice eseguito direttamente da Visual Basic.

```
gestErr:
    m_Conn.RollbackTrans
    If GenF.IDEMode Then
        'se è un errore della Sp
        If ErrSP Then
            MsgBox "Errore nella SP di aggiornamento della giacenza", vbCritical
        Else
            MsgBox "Errore nel Rs di inserimento di valori nella tabella ArticoliFornitoriScaricati"
        End If
```

```
      Stop
    Else
      MsgBox "Errore nel salvataggio dei dati" & vbNewLine & err.Description & _
            vbNewLine & "ripetere l'operazione"
      Unload Me
   End If
End Sub
```

Articolo Fornitore Movimentato

Questa Stored Procedure controlla all'interno delle tabelle di carico scarico e allineamento se un determinato articolo è stato movimentato oppure no. Viene utilizzata all'interno del form di anagrafica degli articoli per consentire o meno all'utente di modificare certe caratteristiche dell'articolo che renderebbero inconsistente il sistema e per permettere l'eliminazione fisica del record: nel caso sia stato movimentato l'articolo deve essere segnato come non più corrente, in modo che non sia più visibile all'utente in anagrafica ma sia ancora presente nella base di dati per visualizzare i dettagli dei movimenti.

```
CREATE Procedure dbo.sp_ArticoloFornitore_Movimentato

        (
                @IdArt Int,
                @Movim bit OUTPUT
        )

As

DECLARE @IdRev as INT

/*
Viene ricavato l'Id di revisione dell'articolo che si vuole movimentare:
*/
SELECT TOP 1 @IdRev = AFRevId FROM ArticoliFornitori
WHERE AFId = @IdArt

/*
Nel caso @IdRev fosse nullo viene ritornato @IdArt, altrimenti @IdRev
*/
SET @IdRev = ISNULL(@IdRev,@IdArt)

/*
Viene controllata la presenza dell'articolo nella tabella ArticoliFornitoriCaricati
ricercandolo tramite l'Id di revisione
*/
```

```
SELECT TOP 1 AFCId FROM ArticoliFornitoriCaricati INNER JOIN
      ArticoliFornitori  ON AFCIdArFo = AFId
      WHERE AFRevId = @IdRev

SET @Movim = @@RowCount

IF @Movim <> 0 Return

/*
Viene controllata la presenza dell'articolo nella tabella ArticoliFornitoriScaricati
ricercandolo tramite l'Id di revisione
*/
SELECT TOP 1 AFSId FROM ArticoliFornitoriScaricati INNER JOIN
      ArticoliFornitori  ON AFSIdArFo = AFId
      WHERE AFRevId = @IdRev
SET @Movim = @@RowCount
```

Implementazione indici di revisione

Tramite questa Stored Procedure vengono impostati gli indici di revisione quando l'utente effettua una modifica ad un articolo che non deve essere retroattiva. Questo record diventerà quindi la revisione corrente dell'articolo, assumento il valore "1" nel campo "AFRevCor" mentre la vecchia revisione corrente avrà questo bit settato a "0" (come tutte le altre revisioni dello stesso articolo). In pratica ogni nuova revisione di un articolo diventa quella corrente e quella vecchia viene impostata come non più corrente. Il campo "AFRevId" della nuova revisione viene invece impostato come l'Id della prima revisione dell'articolo: tutte le revisioni di uno stesso articolo hanno questo campo impostato allo stesso valore, che corrisponde all'Id di tabella della prima revsione dell'articolo, in modo che sia possibile accomunare tutte le revisioni di uno stesso articolo.

```
/*
 Crea l'indice di revisione per la tabella ArticoliFornitori
- OldId è l'id articolo Fornitore del 'vecchio' articolo (quello revisionato)
- NewId è l'Id del nuovo articolo (quello che andrà a sostituirsi)

*/


CREATE Procedure sp_IndiceRev_ArticoliFornitori
      (
            @New_Id as Int,
            @Old_Id as Int
      )

As
```

```
DECLARE @RevId  Int

UPDATE ArticoliFornitori SET AFRevCor = 0
WHERE AFId = @Old_Id

SELECT TOP 1 @RevId = AFRevId
FROM ArticoliFornitori
WHERE AFId = @Old_Id

/* Restituisce @Old_Id solo se @RevId è Null */
SET @RevId = ISNULL(@RevId, @Old_Id)

/* E' una ripetizione nel caso in cui @RevId non era Null */
UPDATE ArticoliFornitori SET AFRevId = @RevId WHERE AFId = @Old_Id

UPDATE ArticoliFornitori
SET AFRevCor = 1,            /* imposta l'articolo come corrente  */
    AFRevId = @RevId         /* associa l'Id di revisione */
WHERE AFId = @New_Id
```

3.6 Note su Transact SQL (T-SQL)

SQL è un linguaggio **non procedurale** con cui gli utenti possono:

1. creare e modificare i databases e gli oggetti contenuti nei databases
2. recuperare e manipolare le informazioni contenuti sui databases. SQL è un linguaggio di alto livello molto semplice da utilizzare e permette di interrogare la base di dati con “domande" in linguaggio naturale (SELECT …FROM), ma possiede un limite: non è stato concepito per la programmazione.

La programmazione richiede funzionalità aggiuntive che vanno dalle istruzioni per il controllo del flusso alla modularità. Per ovviare a questo limite Microsoft ha creato per SQL Server Transact-SQL che a differenza di SQL è un linguaggio procedurale. T-SQL non è standard come SQL anche se è conforme alle specifiche dell’ANSI-92 SQL ed è un linguaggio proprietario che è possibile utilizzare solo con SQL Server. Non è supportato da altre base di dati (come Oracle o MySQL). T-SQL oltre ad ereditare i vantaggi di semplicità e immediatezza di SQL nell’interrogazione della base di datis include una vasta gamma di comandi e istruzioni di controllo che permettono all’utente di lavorare su ogni oggetto e su ogni informazione contenuta in SQL Server (tabelle, indici, login, jobs, alert, backup, ecc…). T-SQL è possibile rappresentarlo con questa formula:

T-SQL = SQL + Estensioni Microsoft

In Microsoft's SQL Server è possibile usare indifferentemente sia istruzioni SQL che T-SQL, ad esempio la SELECT di SQL Standard o quella più sofisticata di T-SQL. Ecco una panoramica delle componenti aggiunte da Microsoft's:

1. Batch
2. Variabili (locali e globali)
3. Cursori scrollabili
4. Procedure memorizzate
5. Tipi di dati
6. Comandi T-SQL
7. Gestione degli errori

Capitolo 4

Conclusioni

Questo lavoro, facendo parte di un progetto più grande, ha contribuito allo sviluppo di una applicazione client server per la gestione amministrativa di una azienda. Dovendosi inserire all'interno di un programma più ampio è stato necessario interfacciarsi con altri moduli e componenti già implementate. Il lavoro, svolto in stretto contatto con il cliente finale, ha permesso lo sviluppo di un prodotto molto vicino alle reali esigenze, ottimizzando di fatto la gestione amministrativa dell'azienda.
Partendo da richieste precise si è portato avanti un progetto enterprise rilasciato in varie versioni successive, ottimizzando di volta in volta le funzionalità disponibili per l'utente su specifiche indicazioni da parte del cliente.
Il progetto è tuttora in sviluppo e sono molte le possibili aggiunte disponibili, alcune delle quali sono già in fase di progettazione. Queste riguardano il programma nella sua interezza e non solo nella gestione del magazzino. In particolare:

- Introduzione di una interfaccia semplificata da utilizzarsi tramite le postazioni installate presso le linee lavorative, per poter gestire agevolmente le attività più frequenti. Sarà possibile per l'operatore ad esempio registrare direttamente ogni aggiunta di prodotto all'interno delle vasche di lavorazione.
- Introduzione di lettori a codici a barre per effettuare le operazioni da e verso il magazzino: sarà sufficiente leggere il codice associato a ogni prodotto e indicarne la quantità, minimizzando quindi i possibili errori e rendendo l'operazione più veloce
- Pubblicazione su di un sito dinamico, accessibile anche dall'esterno previa autenticazione, dello stato di lavorazione degli articoli dei clienti: ogni cliente potrà così direttamente visionare lo stato del proprio ordine in tempo reale.
- Possibilità di utilizzare l'applicazione da remoto, connettedosi al server e alla base di dati tramite la rete internet. E' necessaria ovviamente una gestione più approfondita della sicurezza.
- Possibilità di rendere l'applicativo

Tutti questi possibili sviluppi sono al momento allo studio per una loro reale implementazione per rispondere maggiormente alle esigenze del cliente e per una migliore ottimizzazione delle risorse disponibili.

Bibliografia

[1] – G. Cornell - “Visual Basic 6” - Mc Graw Hill, 1998

[2] – F. Balena - “Programmare in Visual Basic 6.0” - Mondadori Informatica, 1999

[3] - I.Ben-Gan, T. Moreau - “TransactSQL Programmazione avanzata” - Mondadori Informatica, 2001

[4] – it.comp.lang.visual-basic

[5] – http://support.microsoft.com

[6] – http://msdn.microsoft.com

[7] – http://www.html.it

[8] – Microsoft SQL Server Books Online

[9] – Microsoft MSDN Library