

UNIVERSITÀ DEGLI STUDI DI PADOVA



Facoltà di Scienze Statistiche

Corso di Laurea in

Statistica e Tecnologie Informatiche
Sistemi Informativi Aziendali

Relazione finale:

UNA ESPERIENZA DI STAGE: STUDIO E
COMPARAZIONE DI RDBMS IN EASY SOFT S.R.L.

Relatore: Prof.ssa SUSI DULLI

Laureando: ATTILIO DALLA LIBERA

Anno Accademico
2006/07

*“L’Informatica è complessa,
nella sua semplicità”*

– A. Stocchero

Indice

Introduzione	1
1. Caratteristiche del gestionale e struttura del database in EASY	3
1.1 Tecnologia base del gestionale	3
1.2 Interfaccia utente	4
1.3 Caratteristiche salienti del gestionale	4
1.4 Integrazione tra applicazione e database	8
1.4.1 EJB	8
1.4.1.1 JBoss	8
1.4.2 JDBC	9
1.5 Struttura del database in EASY	10
1.5.1 Tabelle	10
1.5.2 Tipi degli attributi	10
1.5.3 Vincoli d'integrità referenziale	11
1.5.4 Vincoli di unicità	12
1.5.5 Indici definiti dall'utente	12
2. Architetture dei database candidati al funzionamento in EASY	13
2.1 Firebird 1.0.x (Firebird 1 – luglio 2000)	13
2.1.1 Accesso alla rete	14
2.1.2 Interclient e connettività JCA – JDBC	14
2.1.3 Architettura multi – generazionale (MGA)	15
2.1.4 Transazioni	15
2.1.5 Stored procedure e trigger	16
2.1.6 Integrità referenziale e indicizzazione	16
2.1.7 Shadowing del database	17
2.1.8 Sicurezza dell'host	17
2.1.9 Privilegi e ruoli	17
2.1.10 Modalità operative	18
2.1.10.1 Classic Server	18
2.1.10.2 Superserver	19
2.1.11 Comparazione tra Classic Server e Superserver	19
2.1.11.1 Eseguibili e processi	19
2.1.11.2 Lock manager	20
2.1.11.3 Utilizzo delle risorse	20
2.1.11.4 Metodo di accesso locale	21
2.2 Firebird 1.5.x (Firebird 2 – febbraio 2004)	21

2.2.1 Migliore ottimizzazione delle query	21
2.2.2 Accesso alla rete	22
2.2.3 Transazioni e indicizzazione	22
2.2.4 Modalità operative	22
2.2.4.1 Embedded Server	23
2.2.4.2 Metodo di accesso locale	23
2.3 Firebird 2.0 Release Candidate 2 (Firebird 2 – maggio 2006)	24
2.3.1 Metodo di accesso locale	24
2.4 SQL Server 2005 Express Edition (aprile 2005)	25
2.4.1 Specifiche del motore di database	25
2.4.1.1 Utilizzo della CPU	26
2.4.1.2 Memoria disponibile	26
2.4.1.3 Dimensioni del file di dati	26
2.4.1.4 Indicizzazione	27
2.4.2 Installazioni del database e relative istanze	28
2.4.3 Application Programming Interface	28
2.4.4 Supporto della connettività di rete	29
2.4.5 Supporto di accesso ai dati	30
2.4.5.1 Multiplexing	30
2.4.5.2 MDAC e SQL Native Client	30
2.4.5.3 Interazioni tra MDAC e SQL Native Client	31
2.4.6 Supporto all'I/O	31
2.4.7 Protezione	32
2.4.7.1 Account sa	32
2.4.8 Supporto per la replica dei dati	33
2.4.9 XCopy di applicazioni	33
2.4.9.1 Opzione AttachDBFilename	34
2.4.9.2 Nome logico del database	35
2.4.9.3 Funzionalità di chiusura automatica	35
2.5 Oracle Database 10g Express Edition (dicembre 2005)	36
2.5.1 Struttura logica del database	36
2.5.1.1 Block	37
2.5.1.2 Extent	37
2.5.1.3 Segment	38
2.5.1.4 Tablespace	38
2.5.1.5 Schema	39
2.5.1.6 Indici	39
2.5.2 Struttura fisica del database	39
2.5.2.1 Data file	40
2.5.2.2 Redo Log file	40
2.5.2.3 Control file	41

2.5.3 Gestione della memoria	42
2.5.3.1 Memoria SGA	42
2.5.3.2 Memoria PGA	44
2.5.4 Gestione dei processi	45
2.5.5 Oracle Net	46
2.5.5.1 Oracle Net listener	46
2.5.5.2 Configurazione del client	46
2.5.6 Sicurezza	47
2.5.6.1 Privilegi e ruoli	48
3. Tool per il trasferimento del database originario	49
3.1 Interfaccia utente (package IO)	50
3.1.1 Classe Input	50
3.1.2 Classe Output	52
3.2 Connessioni ai database (package Connessioni)	54
3.3 Routines trasferimento dati (package Routines)	55
3.3.1 Routine Controlla	55
3.3.2 Routine Create	57
3.3.2.1 Tipi degli attributi	59
3.3.3 Routine Insert	61
3.3.3.1 Tabelle con chiave primaria standard	62
3.3.3.2 Tabelle con chiave primaria non standard	64
3.3.3.3 Inserimenti con tecnica mista	65
3.3.3.4 Gestione valori nulli	65
3.3.4 Routine Alter	66
3.3.5 Routine Index	68
3.3.6 Classe Routines	69
3.4 Corpo dell'applicazione (package Main)	70
4. Implementazione in EASY delle architetture degli RDBMS candidati	75
4.1 Classe SysDatiDitteBean	75
4.2 Classe ConnectionManager	77
4.3 Differenze tra i tipi degli attributi dei vari RDBMS utilizzati	78
4.3.1 Tipo stringa di caratteri a lunghezza variabile	78
4.3.2 Tipo data	79
4.3.3 Tipo intero	80
4.3.4 Tipo decimale	80
4.4 Classe DataConversion	81
4.5 Classe Bridge	82

4.5.1 CAST a run-time e tipo numerico	85
4.5.2 Concatenazione di stringhe	86
4.6 Classe QueryBuild	87
4.7 Classi BeanInput e SchLogin	89
4.8 Query con STARTING WITH	90
4.9 Altre modifiche	91
5. Studio comparativo dei DBMS candidati al funzionamento in EASY	93
5.1 Convenzioni per i nomi dei database	93
5.2 Configurazioni dei pc utilizzati	93
5.2.1 Pc server	94
5.2.2 Pc client	94
5.3 Task	95
5.4 Tempo e complessità	95
5.5 Distribuzione ponderata del lavoro nell'amministrazione del database	96
5.5.1 Dettaglio delle aree di amministrazione	97
5.6 Risultati della comparazione degli RDBMS nelle aree di amministrazione	98
5.6.1 Osservazioni sui risultati precedenti	103
5.7 Distribuzione ponderata del lavoro nei casi d'uso concreti di EASY	104
5.7.1 Dettaglio dei gruppi di operazioni	105
5.8 Risultati della comparazione dei database nei casi d'uso concreti di EASY	106
5.8.1 Osservazioni sui risultati precedenti	117
5.9 Limitazioni e costi delle licenze degli RDBMS utilizzati	117
6. Progetto operativo per l'eventuale transizione del database	119
6.1 Valutazione dell'infrastruttura esistente	119
6.2 Individuazione degli obiettivi del consolidamento del server	120
6.3 Progettazione del nuovo ambiente	120
6.4 Implementazione di un nuovo ambiente pilota	121
6.5 Finalizzazione del piano di migrazione di dati e utenti	122
6.6 Analisi e revisione del progetto	123
7. Conclusioni	125
Bibliografia	127
Ringraziamenti	129

Introduzione

Il presente lavoro è il risultato dell'esperienza di stage presso Easy Soft S.r.l., una software house di Grisignano di Zocco (Vicenza) che produce e commercia EASY, un applicativo gestionale per piccole e medie imprese, oltre che provvedere alla fornitura e riparazione del materiale informatico necessario ai clienti finali.

La maggior parte delle aziende clienti di Easy Soft sono grossisti e dettaglianti di ferramenta, oltre che di altri generi, tra i quali ricambi auto, abbigliamento e alimentari. I punti di forza del software gestionale, riconosciuti da queste aziende, sono l'ottima gestione del magazzino e delle vendite, soprattutto al dettaglio, nonché una completa e sofisticata gestione della contabilità aziendale.

Gli anni di esperienza in questo tipo di mercato, nel quale opera 1981, e le problematiche sollevate da alcuni tra i più importanti clienti della Easy Soft, inducono la software house a provare soluzioni alternative al DBMS gratuito che attualmente immagazzina i dati elaborati dal gestionale, Firebird 1.5. Esigenze quali il miglioramento delle prestazioni, in termini di velocità di esecuzione delle operazioni del gestionale, di irrobustimento del sistema e di limitazione dei danni in caso di grave inconsistenza del database hanno danno vita a questo progetto, che diviene l'obiettivo primario del lavoro svolto durante lo stage.

Dopo un'accurata selezione, gli RDBMS candidati a sostituire Firebird 1.5, sono individuati nelle versioni gratuite distribuite da Microsoft® e Oracle® dei propri database engine, rispettivamente SQL Server 2005 Express e Oracle Database 10g Express, nonché in altre versioni di Firebird, precisamente la 1.0 e la 2.0.

Inizialmente, si analizza la struttura del gestionale e del database esistenti (primo capitolo) e si confrontano le varie architetture degli RDBMS oggetti

dello studio (secondo capitolo), in modo da esaminare le caratteristiche essenziali e peculiari di ogni motore di database.

Dopo l'analisi di queste informazioni preliminari, è necessario progettare un applicativo che trasferisca i metadati e i dati dal database originario verso le nuove strutture prese in esame (terzo capitolo), oltre che implementare tali strutture nel codice sorgente del gestionale (quarto capitolo). Successivamente, si passa all'analisi e alla valutazione comparata delle prestazioni relative a ciascuno dei DBMS candidati (quinto capitolo), il vero e proprio fulcro dell'intero lavoro, e si analizzano i punti necessari alla predisposizione di un progetto per l'eventuale migrazione del database (sesto capitolo), giungendo infine alle conclusioni relative al lavoro svolto (settimo capitolo).

Personalmente, ringrazio tutto lo staff di Easy Soft per avermi permesso di realizzare questa esperienza, in particolare la titolare, sig.ra Anna Maria Polato, e l'ing. Claudio Panzera, senza dimenticare l'apporto, l'aiuto e la pazienza della mia relatrice, la prof.ssa Susi Dulli.

1. Caratteristiche del gestionale e struttura del database in EASY

In questo capitolo sono sommariamente descritte la tecnologia e le principali caratteristiche dell'applicativo EASY, nonché la struttura del database, cuore del programma, con le relative scelte architettureali compiute nella fase di progettazione.

1.1 Tecnologia base del gestionale

Il codice del nuovo gestionale è scritto nel linguaggio Java, precisamente è stata utilizzata la versione 1.4.0-b92. È il nuovo linguaggio internazionalmente riconosciuto come lo strumento di programmazione del futuro, ed inoltre Java significa grande flessibilità di piattaforme: il gestionale può lavorare in ambiente Windows, ed anche in Linux e Mac OS. L'ambiente di sviluppo utilizzato in tutto l'arco del progetto è JBuilder™ 6 Enterprise Edition, un Java editor distribuito da Borland®, particolarmente adatto alla creazione e alla gestione degli EJB, descritti nei prossimi paragrafi. Un ulteriore punto di forza di questo editor è la gestione in CVS (Concurrent Versioning System), che implementa un sistema di controllo versione: tiene al corrente il software in via di sviluppo di tutti i cambiamenti in un insieme di file e permette a molti sviluppatori, potenzialmente distanti tra loro, di collaborare.

È stato scelto come RDBMS (Relational Database Management System) la versione gratuita di InterBase, Firebird, un database molto affidabile e disponibile senza alcun pagamento di licenze. Attualmente, la versione del motore di database utilizzata presso la Easy Soft e i relativi clienti, è la 1.5.3.

1.2 Interfaccia utente

L'accesso alle informazioni presenta il tipico aspetto di un programma a finestre, pensato sia per quegli utenti già abituati all'uso del mouse sia per quelli più tradizionalmente legati al solo uso della tastiera.

Le informazioni si presentano, nell'ambiente grafico, raccolte ed evidenziate per ottenere la massima facilità di lettura, con un aspetto dei quadri presentati semplice e intuitivo. Per l'utenza più esigente tali quadri risultano altamente configurabili, in modo diverso per ciascuna postazione di lavoro.

L'utente ha immediatamente la sensazione su quale campo sta lavorando, perché questo è evidenziato con un colore diverso: un uso ragionato dei colori e della grafica rende immediatamente consapevoli degli aspetti essenziali tra tutte le informazioni visibili e, di conseguenza, guida l'utente nell'operare.

Sono inoltre previste maschere di informazione quando per errore vengono inseriti dati incongruenti.

1.3 Caratteristiche salienti del gestionale

Il gestionale EASY si propone come evoluzione del software già disponibile sul mercato da anni, sviluppato con la tecnologia SuperDos®, non solo in termini di grafica e flessibilità. L'obiettivo è quello di mantenere le funzioni disponibili ed integrarle con nuove idee atte a completare e migliorare il software, sia dal punto di vista prestazionale e funzionale, sia dal punto di vista giuridico e amministrativo.

Le principali caratteristiche dell'applicativo EASY sono:

- Utilizzo in rete del programma, anche su macchine dotate di sistemi operativi diversi: alta configurabilità del sistema e facile inserimento di nuove stazioni di lavoro al programma.
- Operatività in ambiente multi-tasking: quando EASY è attivo l'utente può, in caso di necessità, passare a lavorare un testo con un editor, un foglio elettronico o qualunque altro programma senza dover interrompere la propria sessione di lavoro nel gestionale.
- Nel contempo, è possibile attivare più sessioni di lavoro sulla stessa macchina (con lo stesso operatore o con operatori diversi, se la licenza lo permette), consentendo dunque all'utente operazioni multiple differenti.
- Figura di amministratore di sistema (SUPERVISOR), in grado di configurare i privilegi di accesso per tutti gli utenti abilitati nel programma: in questo modo ogni utente potrà accedere solo a quelle parti del programma a lui riservate, ed eseguire solo le operazioni cui è stato abilitato. In Fig. 1.1 appare la schermata iniziale di login al gestionale.



Fig. 1.1

- Possibilità da parte del SUPERVISOR di verificare la sequenza delle operazioni eseguite sui dati, con conoscenza dell'operatore, data/ora

dell'operazione compiuta, ed altro: possibilità di ripristinare tutti i dati cancellati in precedenza.

- Gestione documenti: piena configurabilità dei passaggi, esempio, di un documento da ordine a bolla, e di qui a fattura. Ogni meccanismo di movimentazione può essere deciso dal SUPERVISOR e realizzato automaticamente in fase di registrazione.
- Gestione semplificata delle anagrafiche: una stessa anagrafica può appartenere ad un cliente, ad un fornitore oppure ad un vettore (Fig. 1.2). Diventa così possibile ritrovare le informazioni connesse alla stessa persona o ditta in qualunque punto del programma questa sia coinvolta.

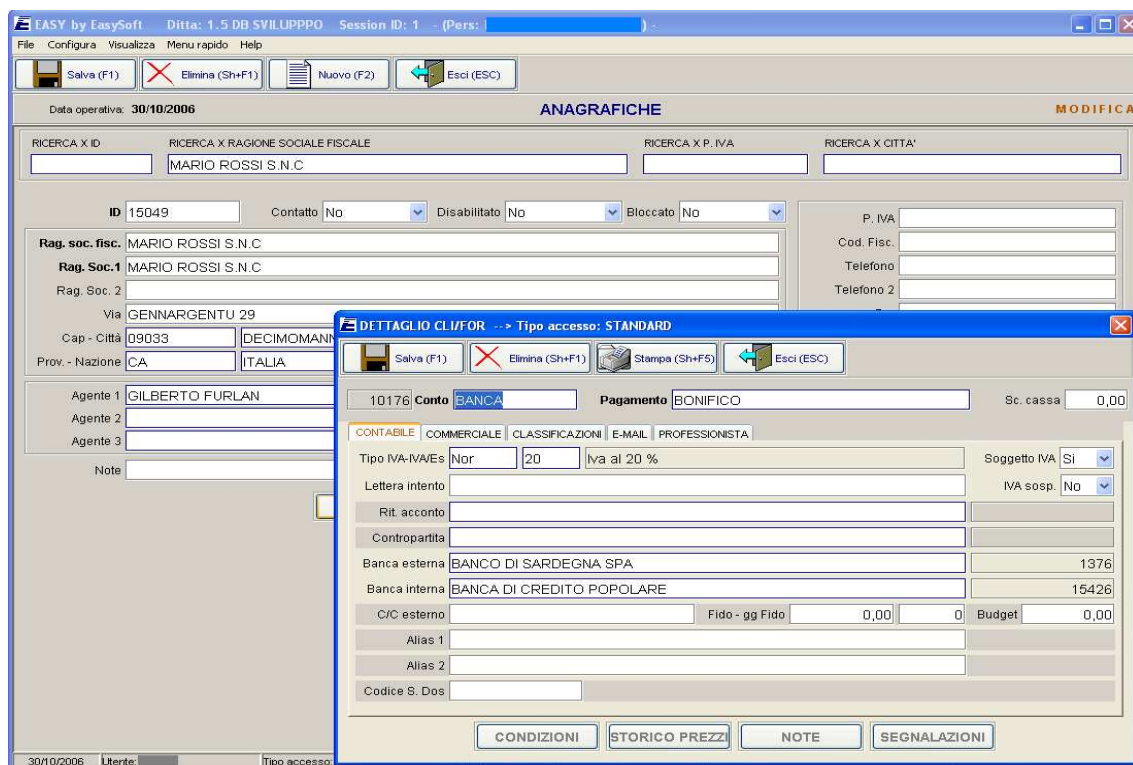


Fig. 1.2

- Scheda articoli: denominazioni illimitate dello stesso articolo presso i diversi fornitori e clienti.
- Pieno sostegno all'operatore in fase di inserimento dati. Un HELP in linea è disponibile per ricordare le funzionalità base del programma e

consente, inoltre, di inserire in maniera autonoma dei commenti o delle annotazioni al fine di aiutare l'utente stesso nell'eseguire le varie operazioni. La ricerca dei codici e descrizioni (di un articolo, un cliente, una banca, una causale od altro) è facilitato dalla pressione dei tasti F5 (Fig 1.3) o F9 a seconda se si preferisce visualizzare l'elenco delle voci richieste, rispettivamente, con la stringa immessa situata all'inizio o contenuta nel codice o descrizione articolo. In fase di inserimento di un ordine (o l'emissione di una fattura o DDT) l'operatore può avere a video il quadro completo di tutte le condizioni di vendita disponibili per il cliente/articolo o fornitore/articolo in questione, ottenendo così un supporto nella scelta della quantità più conveniente da proporre.

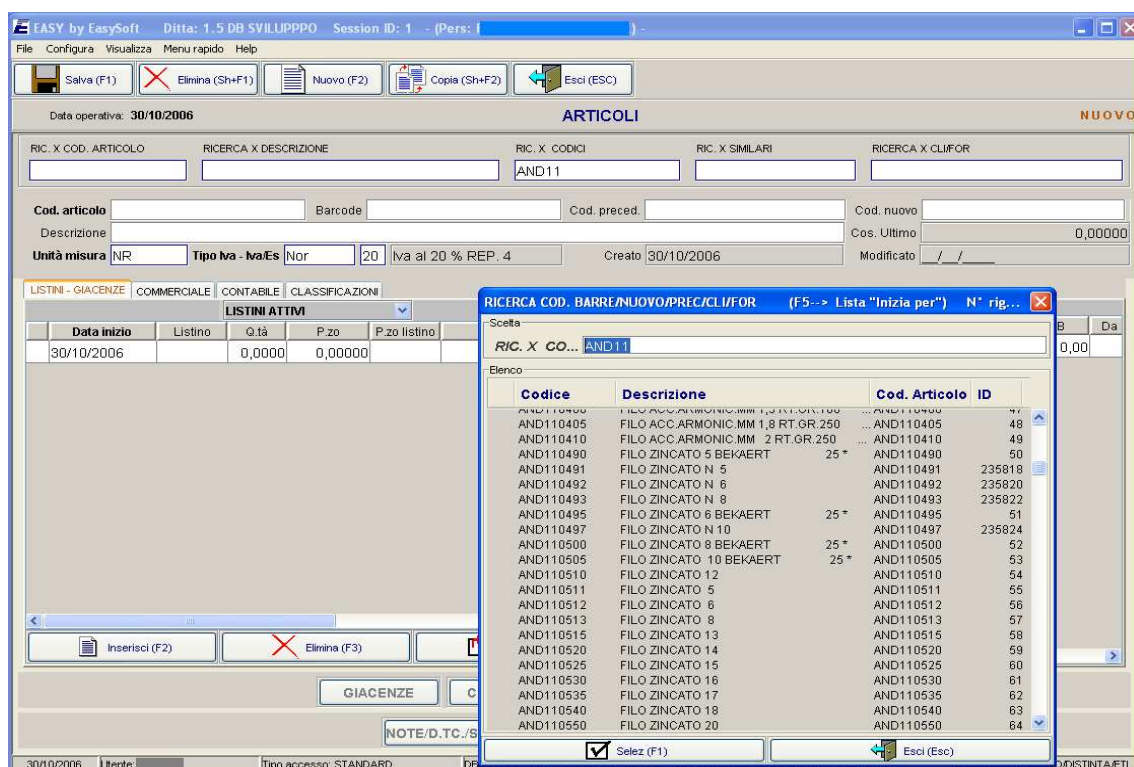


Fig. 1.3

- Gestione multi-ditta e multi-esercizio, anche esercizi fiscali non solari.
- Gestione condizioni di vendita: EASY prevede diverse procedure guidate per aiutare l'operatore ad inserire prezzi e sconti su prodotti, categorie prodotti, clienti o categorie clienti, sia a livello di condizioni standard che

di offerta temporanea. Le offerte vengono salvate e conservate in un archivio storico.

1.4 Integrazione tra applicazione e database

Il gestionale si interfaccia al database tramite due metodi: gli EJB (Enterprise Java Beans), derivati dalla tecnologia J2EE, e JDBC.

1.4.1 EJB

Gli EJB sono i componenti che implementano, lato server, la logica di business all'interno dell'architettura J2EE (Java 2 Enterprise Edition), cioè la versione della piattaforma Java costituita da un insieme di specifiche che definiscono le caratteristiche e le interfacce di un insieme di tecnologie pensate per la realizzazione di applicazioni aziendali. Le specifiche per gli EJB definiscono diverse proprietà che questi devono rispettare, tra cui la persistenza, il supporto alle transazioni, la gestione della concorrenza e della sicurezza.

In EASY, si è scelta la tecnologia EJB per poter meglio controllare i continui sviluppi del software: quando, ad esempio, un cliente necessita di una personalizzazione su uno specifico componente, o urgono degli aggiornamenti dell'applicativo stesso, basta modificare le poche classi relative a quel *bean* e ricompilarle nello spazio dedicato al *deploy* dell'application server.

1.4.1.1 JBoss

Gli EJB in EASY sono gestiti tramite JBoss 3.0.5, un application server open-source e compatibile con gli standard EJB, completamente sviluppato in Java.

La particolarità di JBoss deriva dalla metodologia di programmazione con la quale è stato pensato, basata su JMX (Java Management eXtension) che è il mezzo migliore per ottenere software integrabile. JMX fornisce un'interfaccia per mezzo della quale poter inserire moduli, contenitori e *plugin* (aggiunte) che costituiscono JBoss, ma permette anche di estendere questi moduli da parte dell'utente.

1.4.2 JDBC

JDBC (Java Database Connectivity) è un *middleware*, cioè un programma che funge da intermediario tra diverse applicazioni, che consente l'accesso al database a qualsiasi applicazione scritta in Java, indipendentemente dal tipo di DBMS utilizzato. È costituita da una API (Application Programming Interface), raggruppata nel package *java.sql*, che serve ai client per connettersi al database. Fornisce metodi per interrogare e modificare i dati. La piattaforma J2EE contiene le JDBC API, insieme all'implementazione di un *bridge JDBC-ODBC*, che permette di connettersi a database relazionali che supportino ODBC (Open Database Connectivity).

L'architettura di JDBC, così come quella di ODBC, prevede l'utilizzo di un *driver manager*, che espone alle applicazioni un insieme di interfacce standard e si occupa di caricare a run-time i driver opportuni per gli specifici DBMS. Le applicazioni Java utilizzano le JDBC API per comunicare con il JDBC driver manager, mentre il driver manager usa le JDBC driver API per comunicare con i singoli driver dei DBMS specifici.

Tutte le connessioni e le query di EASY al database sono gestite tramite le chiamate JDBC.

1.5 Struttura del database in EASY

Come già illustrato, il database EASY è gestito tramite l'RDMBS Firebird 1.5.3. Il programma gestionale si interfaccia al server del database tramite il driver JDBC, *Interclient*. L'aspetto delle connettività di rete sarà approfondito nel secondo capitolo, riguardante le architetture dei database.

1.5.1 Tabelle

Il database originario di EASY, che quindi non tiene conto di eventuali personalizzazioni specifiche per ogni azienda cliente e delle eventuali tabelle temporanee create dal gestionale, conta 94 tabelle definite dai progettisti, più ovviamente le tabelle di sistema create dall'RDBMS, che contengono informazioni quali dipendenze tra tabelle, indici, set di caratteri e quant'altro sia utile al database per funzionare correttamente.

Le dimensioni delle tabelle sono molto variabili, sia come numero di attributi (*colonne*) che le costituisce, sia come numero di informazioni (*record* o *righe*) salvate in esse. Ad esempio, la tabella ANNI è formata da una sola colonna e conta, attualmente, solamente 17 record, mentre, ad esempio, per le tabelle *ARTICOLI*, *CONDIZIONI* o *DOCUMENTITESTATE* sono specificati, rispettivamente, 62, 19 e 74 attributi; inoltre, data la loro continua espansione, esse possono raggiungere il milione di righe e addirittura superarlo presso i database delle aziende di maggiori dimensioni.

1.5.2 Tipi degli attributi

Durante la progettazione del database di EASY, si è preferito utilizzare una minima parte dei tipi di dati messi a disposizione da Firebird, in modo da

ottenere un'elevata omogeneità tra i dati contenuti nelle tabelle. I principali tipi utilizzati sono:

- VARCHAR per i dati di tipo testo; la dimensione massima associata a questo tipo è 5.000 caratteri nei campi *note* del database, ma mediamente non supera mai i 1.000 caratteri.
- DATE per le date dei documenti e delle modifiche; sono salvate nel formato *mm-gg-aaaa*.
- INTEGER per i campi identificativi delle tabelle.
- SMALLINT solitamente per i campi *flag*, come ad esempio *attivo* o *bloccato*.
- DOUBLE PRECISION per i dati numerici o riguardanti le valute (Euro e USD).

1.5.3 Vincoli d'integrità referenziale

Escludendone un esiguo numero, in quasi tutte le tabelle i progettisti hanno introdotto un campo identificativo intero progressivo, *id<nome tabella>*, che funge da chiave primaria artificiale. Così facendo, anche per le tabelle di collegamento per le identità multi-a-molti, ogni record è univoco e distinguibile dagli altri.

Le chiavi esterne che collegano le tabelle della versione originaria del database sono 186. Molte tabelle, come ad esempio *DOCUMENTITESTATE*, hanno più campi, in questo caso 18, collegati ad altrettanti campi chiave primaria di altre tabelle del database.

Ovviamente, un maggior numero di chiavi esterne, con relative clausole di UPDATE CASCADE e DELETE CASCADE, rallenta le prestazioni del motore del database utilizzato, ma più in generale di un qualsiasi RDBMS.

1.5.4 Vincoli di unicità

Un vincolo di unicità, proprio come una chiave primaria, assicura che due righe non abbiano mai lo stesso valore in una colonna o un gruppo di colonne.

Alcuni campi delle tabelle di EASY, come ad esempio *idarticolo* di *ARTICOLI*, sono dichiarati con la clausola *UNIQUE* per non permettere repliche dello stesso codice, che porterebbero inevitabilmente, ad esempio, l'utente finale ad associare un identico barcode a due articoli differenti.

1.5.5 Indici definiti dall'utente

Un indice è un puntatore logico ad un *indirizzo* (locazione fisica) di righe di una tabella. Nella maggior parte dei casi, se il motore del database può ricercare le righe richieste dall'utente direttamente scorrendo la lista dei puntatori contenuta in un indice, anziché scorrere tutte le righe della tabella, la richiesta è completata in un tempo minore. Un piano di indicizzazione ben ottimizzato gioca un ruolo fondamentale sulle prestazioni finali di un database.

Oltre all'indicizzazione automatica creata dall'RDBMS sui vincoli d'integrità referenziale, i progettisti del database EASY hanno aggiunto degli indici, precisamente 19, su alcuni campi chiave di un ristretto numero di tabelle. Questa decisione è stata presa per due motivi:

- I campi soggetti a frequenti ricerche da parte dell'utente finale, ad esempio il campo *descrizione* di *ARTICOLI*, hanno bisogno di tempi di accesso più rapidi rispetto ad altre informazioni delle medesime tabelle.
- Il numero di campi indicizzato è giustamente ridotto al minimo indispensabile, poiché quando viene aggiunto o aggiornato un valore in un campo indice, il database rallenta per integrare o modificare la relativa lista di puntatori.

2. Architetture dei database candidati al funzionamento in EASY

Questo capitolo è dedicato all'analisi dei cinque RDBMS confrontati nello studio, focalizzando le peculiarità specifiche della struttura di ciascuno. I nomi, per esteso, dei DBMS esaminati sono i seguenti:

- Firebird® 1.0.3 Version WI-V6.2.972
- Firebird® 1.5 Version 1.5.2.4870
- Firebird® 2.0 Release Candidate Version 2.0.0.12654
- Microsoft® SQL Server 2005 Express Edition Service Pack 1
- Oracle® Database 10g Express Edition Release 2 (10.2.0.1.0)

In seguito, per rendere più scorrevole la lettura, i nomi dei database saranno abbreviati. In particolare, per la famiglia Firebird si esamineranno soltanto le versioni maggiori e non si terranno conto delle release minori, poiché lo scopo dell'analisi è descrivere le tre architetture nel modo più generale possibile, evidenziando soltanto i principali cambiamenti tra una versione e le relative precedenti.

2.1 Firebird 1.0.x (Firebird 1 – luglio 2000)

Il *database engine* (motore di database) di Firebird 1.0 è stato programmato da un team indipendente di sviluppatori partendo dal codice di InterBase che fu rilasciato da Borland sotto la "InterBase Public License v.1.0" del 25 luglio 2000.

Questa release è stata costruita da un codice, scritto in C, ampiamente ripulito con la correzione dei bug originari e grazie alla fase di beta testing, denominato Firebird 1.

2.1.1 Accesso alla rete

Un server Firebird, operativo su qualsiasi piattaforma, accetta connessioni con protocollo TCP/IP (Transmission Control Protocol) da parte di un qualsiasi client che implementi le API (Application Programming Interface) di Firebird. Il protocollo TCP permette alle applicazioni eseguite in una rete di computer di connettersi tra loro e scambiarsi dati in *pacchetti*; inoltre, tale protocollo riconosce le connessioni multiple ad una stessa macchina e regola l'accesso concorrente ad essa. Un client si deve connettere al server attraverso un *path* (percorso) fisico assoluto, cioè specificando l'indirizzo IP (Internet Protocol) della macchina su cui risiede.

Generalmente, in ambiente Windows, l'istanza predefinita all'installazione di Firebird è denominata *gds_db* e va ad occupare la porta di rete 3050.

2.1.2 Interclient e connettività JCA – JDBC

Il team Firebird-Java ha implementato un driver Java, *Interclient*. Tale driver è basato sia sulla tecnologia JCA (Java Connector Architecture), sia sulla più ben nota JDBC, descritta nel primo capitolo. Lo standard JCA specifica un'architettura in cui un'applicazione server può cooperare con un driver JDBC, in modo tale che il server possa gestire le risorse, le transazioni e la sicurezza, mentre il driver si occupa soltanto della gestione delle connessioni al database. Poiché è molto simile al concetto di JDBC 2 XADataSource, la specifica JCA è considerevolmente più chiara per quanto riguarda la divisione dei compiti tra applicazione server e driver JDBC.

2.1.3 Architettura multi – generazionale (MGA)

Il modello di Firebird per isolare e controllare il lavoro di più utenti si dipana su un'architettura capace di conservare, in maniera concorrente, più di una versione di un singolo record nel database. Molteplici generazioni, cioè versioni, di un record possono esistere simultaneamente: ogni *task* (operazione) utente che interessi il cambiamento di stato di un record, ha una propria visione dello stato del database e, quando è pronto a farlo, scrive la propria versione della tupla sul disco del server. A quel punto, alla nuova versione (o cancellazione) non possono accedere altri processi utente.

La più recente versione del record su cui è stato eseguito il commit, è l'unica visibile dagli utenti che ne hanno demandato con successo una nuova versione, e continua ad essere la versione vista da tutti gli altri utenti. Questi ultimi si accorgeranno che qualcosa è accaduto alla tupla, poiché saranno a loro impediti aggiornamenti e cancellazioni sul medesimo record, finché questo non sarà diventato definitivo dopo lo statement di commit.

2.1.4 Transazioni

Tutti i processi utente in Firebird sono delimitati dalle transazioni. Un processo inizia con uno statement di `START TRANSACTION`, e termina quando sul lavoro dell'utente demandato al database viene eseguito lo statement di commit o roll-back. Un task può includere più operazioni per essere completato in una singola transazione, ed esse possono riguardare più di un database.

Il lavoro è salvato in due tempi nel database. In un primo tempo, le modifiche sono archiviate nel disco, senza compiere alterazioni al database; in seguito, alle modifiche è applicato il commit o il roll-back.

Le transazioni in Firebird sono atomiche, significa cioè che tutte le operazioni contenute in un task vanno tutte a buon fine, oppure fallisce tutto il processo.

2.1.5 Stored procedure e trigger

Firebird possiede un ricco linguaggio per estensioni procedurali, il PSQL, per programmare stored procedure e trigger. È un linguaggio strutturato per il ciclo FOR, attraverso salti condizionati, gestione delle eccezioni e triggering degli eventi. Il codice scritto in PSQL è compilato durante la creazione e salvato in formato binario.

Importante è il supporto del database ai trigger, con una fase di BEFORE e una di AFTER per ogni istruzione DML (INSERT, UPDATE e DELETE).

I trigger multipli possono essere salvati per ogni fase o evento, e opportunamente numerati per un'esecuzione in sequenza.

2.1.6 Integrità referenziale e indicizzazione

Firebird supporta l'integrità referenziale SQL standard, includendo le opzioni di aggiornamento e cancellazione a cascata.

In Firebird 1.0, per ogni tabella è permesso creare al massimo 64 indici. Ogni pagina radice dell'indice occupa 31 byte per il suo identificatore, spazio per la descrizione di ogni colonna compresa nell'indice e alcuni byte per salvare un puntatore alla prima pagina dell'indice. Una radice di un indice non può eccedere, in ampiezza, i 253 byte.

2.1.7 Shadowing del database

Il server Firebird può opzionalmente mantenere più copie dello stesso database. Una *shadow* (letteralmente “ombra”) è una copia in tempo reale del database, con alcuni attributi aggiuntivi che ne rendono inutilizzabile la lettura, finché non è resa possibile dal server. Le shadow possono essere “promosse” a database primario qualora l’utente lo ritenesse opportuno, in maniera automatica o manuale, nel caso in cui il database primario subisse dei danni non dovuti alla volontà dell’operatore, ad esempio dovute all’improvvisa inconsistenza dell’hardware. Infatti, l’obiettivo principale dello shadowing è rendere utilizzabile il database in tempi molto rapidi nel caso occorresse un errore del genere.

2.1.8 Sicurezza dell’host

Firebird provvede all’accesso sicuro degli utenti al server grazie all’utilizzo di account utenti e password criptate. Il server può immagazzinare dati criptati, ma, escludendo la password, non ha la possibilità di criptare autonomamente tali informazioni.

2.1.9 Privilegi e ruoli

Nessun altro utente, eccetto l’utente SYSDBA e il proprietario del database, possiede i privilegi (*SQL privileges*) necessari per accedere ad un database individuale. Quindi, ad ogni utente, devono essere garantiti esplicitamente i diritti di accesso ad un determinato oggetto.

I ruoli degli utenti (*SQL roles*) sono insiemi di privilegi che possono essere assegnati e garantiti in “pacchetti” agli utenti del database. Un singolo

operatore può possedere privilegi contenuti in diversi ruoli, sebbene solamente un ruolo possa essere selezionato al momento della connessione del database.

2.1.10 Modalità operative

Il server di Firebird 1.0 può essere installato in due modi: *Classic Server* e *Superserver*. La distinzione è meramente una questione di architettura, nel senso che ogni applicazione client progettata per connettersi in modalità Superserver, ha la possibilità di connettersi esattamente allo stesso modo nell'altra modalità. Anche il viceversa è ammesso, con l'eccezione che il Superserver possiede dei requisiti per la sicurezza dei processi per le funzioni esterne (funzioni definite dall'utente, librerie di caratteri e filtri BLOB) più rigidi.

2.1.10.1 Classic Server

L'architettura Classic Server ha preceduto, cronologicamente, quella Superserver. È stata progettata nel tardo 1980, quando le risorse delle macchine server e i programmi le usavano con molta parsimonia. Tale modello seguì nei sistemi operativi che non supportavano il multi-threading e le cui capacità erano troppo limitate per supportare la versione Superserver.

Il Classic Server rimane la migliore alternativa in quegli ambienti dove contano principalmente le prestazioni elevate, e le risorse del sistema sono adeguate per affiancare una crescita lineare dell'utilizzo del server, ogni volta che a questo si aggiunge una nuova connessione.

2.1.10.2 Superserver

Nel 1996, l'antenato di Firebird, InterBase 4.1, introdusse l'architettura Superserver multi-thread per le nuove piattaforme Windows a 32 bit: lo scopo era fare miglior uso delle capacità avanzate delle macchine server e delle connessioni di rete. L'efficienza del Superserver di avvicinare i processi utente e allocare memoria dinamicamente, lo fa preferire al Classic Server laddove vi siano limitate risorse di sistema e un numero consistente di utenti concorrenti in lettura e scrittura.

Con l'espandersi del sistema operativo GNU/Linux sui processori Intel alla fine degli anni '90, il Superserver divenne un obiettivo primario per molte piattaforme POSIX. Un primo scheletro di questa architettura per Linux fu rilasciato con la licenza open-source di InterBase 6.0, e fu realizzato definitivamente con Firebird 1.0.

2.1.11 Comparazione tra Classic Server e Superserver

Anche se le architetture Classic Server e Superserver si basano su caratteristiche comuni, essendo costruite sullo stesso codice di base, esse presentano tratti operativi distintivi, analizzati nei prossimi paragrafi.

2.1.11.1 Eseguibili e processi

Il Classic Server avvia, quando l'utente lo richiede, un solo processo server per connessione. Quando un client tenta la connessione ad un database Firebird, viene creata un'istanza dell'eseguibile *fb_inet_server* e rimane dedicata a quella specifica connessione client per tutta la durata della connessione stessa. L'istanza del processo termina quando il client si disconnette dal database.

Il Superserver esegue una singola invocazione dell'eseguibile *fbserver*: esso è avviato una sola volta, tramite uno script del sistema operativo o dell'utente amministratore, e rimane attivo in attesa delle richieste di connessione al server da parte dei client. Il processo termina soltanto con una chiamata di chiusura esplicita.

2.1.11.2 Lock manager

Nel Classic Server, ogni processo client ha la propria memoria dedicata sul database, e i processi multipli si contendono l'accesso al database server. Un sottosistema *lock manager* (gestione concorrenziale), *fb_lock_mgr*, utilizza i metodi di ICP (comunicazione interprocessuale) per sincronizzare e dare priorità agli accessi dei diversi processi attivi.

Nel Superserver, il lock manager è implementato come un thread nel processo *fbserver* e anch'esso utilizza i meccanismi ICP.

2.1.11.3 Utilizzo delle risorse

Il Classic Server alloca pagine di memoria cache statica del database per ogni istanza di *fb_inet_server*. La crescita della richiesta per ogni connessione client aggiuntiva è quindi lineare: quando il numero di client concorrenti è relativamente basso, il Classic Server utilizza poche risorse globali.

Il Superserver impiega un unico spazio di memoria che viene condiviso tra le connessioni client, fornendo migliori utilizzo e gestione della memoria cache quando il numero di connessioni simultanee cresce maggiormente.

2.1.11.4 Metodo di accesso locale

L'architettura Classic Server permette ai processi che avanzano sulla stessa macchina del server e del database di compiere operazioni di I/O, direttamente sul file di dati.

Il Superserver richiede alle applicazioni di utilizzare un metodo di rete per le richieste di I/O. Sulle piattaforme non-Windows le connessioni locali al Superserver sono realizzate tramite l'host locale del server con IP 127.0.0.1. Nei server Windows, server e client possono simulare una connessione di rete usando i metodi ICP.

2.2 Firebird 1.5.x (Firebird 2 – febbraio 2004)

Lo sviluppo del codice Firebird 2 iniziò presto dopo quello di Firebird 1, con il porting di Firebird in C++ e un forte *refactoring* (rivisitazione) del codice. Firebird 1.5 è la prima release del codice Firebird 2: è la pietra miliare per gli sviluppatori e l'intero progetto Firebird, ma non è fine a sé stessa. Contemporaneamente allo sviluppo della versione 1.5, infatti, si procede con il cammino che porta alla prossima release nel percorso del codice Firebird 2. Firebird 1.0 continua ad essere mantenuto attivo, con importanti correzioni di bug e aggiunte importati.

In seguito sono riportati le differenze sostanziali del codice Firebird 2 dal suo predecessore, nonché le particolarità della release 1.5.

2.2.1 Migliore ottimizzazione delle query

Durante il processo di sviluppo della versione 1.5, l'*SQL query optimizer*, ossia il processo del database engine che ottimizza il piano d'esecuzione delle query,

subisce numerosi aggiornamenti e miglioramenti che si palesano con una tangibile diminuzione della velocità di esecuzione dal 30 al 60%.

2.2.2 Accesso alla rete

Ha le stesse caratteristiche della versione 1.0; inoltre, da Firebird 1.5 in poi, esiste una componente che permette alle applicazioni client di connettersi attraverso gli alias degli indirizzi delle macchine, i cui path sono configurati in ogni server attivo.

Un server Firebird funzionante in una rete Windows, può accettare connessioni dai client attraverso il protocollo di rete Named Pipes.

2.2.3 Transazioni e indicizzazione

In aggiunta a quanto detto per la versione 1.0, da Firebird 1.5, per gli utenti è possibile eliminare parte del lavoro demandato al database immettendo dei *savepoint*, che, in caso di roll-back, anziché perdere l'intera transazione, viene effettuato soltanto fino al savepoint desiderato.

Per quanto riguarda gli indici, rispetto alla versione 1.0, Firebird 1.5 permette all'utente di creare fino a 256 indici per tabella.

2.2.4 Modalità operative

Con l'avvento di Firebird 1.5, oltre alle architetture Classic server e Superserver, si è introdotta una terza modalità, l'*Embedded Server*, che va incontro alle esigenze dei database utilizzati esclusivamente all'interno di una rete aziendale, velocizzandone letture e scritture.

2.2.4.1 Embedded Server

Firebird 1.5 introduce la variante *embedded* (dedicata) del Superserver solamente per le piattaforme Windows. In questo modello, un normale Superserver è compilato con un client dedicato che si connette direttamente ed esclusivamente al proprio database. Un'unica libreria dinamica (*fembed.dll*) usa la comunicazione interprocessuale di Windows per coordinare le richieste del client e le risposte del server. L'API dell'Embedded è identica a quella della modalità Classic o Superserver, e non è richiesta nessuna implementazione aggiuntiva al codice per creare un'applicazione Embedded.

Un'applicazione dedicata può usare soltanto l'accesso da una rete locale (e non worldwide) e supporta un unico processo utente. Si possono sviluppare diverse applicazioni dedicate sulla stessa macchina, ma un singolo database può avere una sola applicazione connessa per volta. Un'applicazione sviluppata in modalità Embedded Server può essere eseguita in modo concorrente nella macchina, come in una qualsiasi altra architettura di Firebird. L'applicazione, comunque, non può essere utilizzata dal Superserver e dall'Embedded Server contemporaneamente.

La modalità Embedded provvede a fornire un'applicazione autonoma, performante e mono-macchina per il database, che può essere sviluppata in pochi passi. Dal momento che il database permette l'accesso ad un normale server quando quello dedicato è offline, l'Embedded Server è particolarmente utile per lo sviluppo mobile, ad esempio su portatili, notebook o dischi removibili.

2.2.4.2 Metodo di accesso locale

L'architettura Embedded Server può utilizzare soltanto il metodo di accesso locale di Windows, e supporta una e solamente una connessione ad ogni

database locale. Il client dedicato è in grado di connettersi a più database locali in modo mutualmente esclusivo, e vi accede tramite il server. Il client dedicato può, inoltre, connettersi come un normale client ai database di altri server.

2.3 Firebird 2.0 Release Candidate 2 (Firebird 2 – maggio 2006)

La release 2.0 di Firebird apporta un largo numero di miglioramenti a lungo attesi, che incrementano notevolmente le prestazioni, la sicurezza e il supporto dei linguaggi internazionali. Il codice di base rimane sempre quello di Firebird 1.5, cioè Firebird 2, ma, essendo ancora una release candidate, Firebird 2.0 è ancora in beta testing.

L'OSD (struttura fisica sul disco) del database creata per le precedenti versioni di Firebird è cambiata: sebbene la 2.0 abbia la possibilità di connettersi all'OSD delle precedenti, le nuove funzionalità non sono supportate, invece, da quest'ultima.

2.3.1 Metodo di accesso locale

Firebird 2.0 ha rimpiazzato la precedente implementazione del protocollo di rete locale IPC con uno nuovo, denominato XNET, esclusivamente per piattaforme Windows.

Lo scopo è identico al precedente, poter connettere in maniera efficiente il client al database localizzato nel server. Anche l'implementazione XNET utilizza memoria condivisa per le comunicazioni interprocessuali, ma modifica la logica di sincronizzazione tra i task. Inoltre, i vantaggi portati dalla tecnologia XNET sono che funziona con la modalità Classic Server ed elimina i blocchi nel caso in

cui fosse in attesa un notevole numero di connessioni simultanee. Infine, ci si aspetta che il protocollo XNET sia, anche se leggermente, più veloce del IPC.

2.4 SQL Server 2005 Express Edition (aprile 2005)

SQL Server 2005 Express Edition, che in seguito sarà abbreviato in SQL Server Express, è un prodotto gratuito basato sulla tecnologia SQL Server 2005. Rispetto alle altre edizioni di SQL Server 2005 offre caratteristiche di rete e di protezione differenti ed è l'unico a includere la funzionalità XCopy per le applicazioni.

Oltre a utilizzare lo stesso motore di database delle altre versioni di SQL Server 2005, SQL Server Express utilizza anche le stesse API di accesso ai dati. Questo prodotto differisce, in questa release gratuita, dalle altre edizioni di SQL Server 2005 per gli aspetti seguenti:

- mancanza del supporto per le funzionalità aziendali
- supporto di una sola CPU
- limite di 1 GB di memoria per il *pool di buffer*
- dimensione massima dei database limitata a 4 GB

Funzionalità quali la chiusura automatica e la possibilità di copiare database come semplici file sono attivate per impostazione predefinita, mentre sono assenti le funzionalità di BI (Business Intelligence).

2.4.1 Specifiche del motore di database

Come già illustrato, a causa delle limitazioni della release, il motore SQL supporta una CPU, 1 GB di RAM e database con dimensione massima di 4 GB. Per quanto riguarda gli altri aspetti, invece, rispetto alle precedenti versioni di

SQL Server 2005, non sono previsti limiti per il carico di lavoro e il motore offre le stesse prestazioni.

2.4.1.1 Utilizzo della CPU

SQL Server Express può essere installato ed eseguito anche su computer multiprocessore, ma è possibile utilizzare una singola CPU alla volta. Internamente, il motore limita ad uno il numero dei thread dell'utilità di pianificazione utente ed è pertanto possibile utilizzare un solo processore. A causa di questa limitazione, funzionalità quali l'esecuzione parallela delle query non sono supportate.

2.4.1.2 Memoria disponibile

Il limite di 1 GB di RAM limita la quantità di memoria disponibile per il *pool di buffer*: esso è utilizzato per archiviare pagine di dati ed altre informazioni. La memoria necessaria per tenere traccia di connessioni, blocchi e così via non rientra invece nel conteggio relativo al limite per il pool di buffer.

2.4.1.3 Dimensioni del file di dati

Il limite di 4 GB relativo alla dimensione dei database si riferisce solo ai file di dati e non ai file di log delle transazioni. Non è tuttavia previsto alcun limite al numero dei database che è possibile collegare al server.

2.4.1.4 Indicizzazione

Il metodo di indicizzazione in SQL Server Express è la ricerca *full-text*. Un indice full-text è un tipo speciale di indice che viene creato e gestito dal servizio *msftesql* (Microsoft Full-Text Engine for SQL Server). Il processo di creazione è piuttosto diverso da quello di altri tipi di indici. Anziché creare una struttura *b-tree* (albero binario) basata su un valore archiviato in una riga specifica, *msftesql* crea un indice invertito, compresso e in pila, basato sulle singole porzioni del testo che si sta indicizzando. Il processo di creazione e gestione di un indice full-text è definito da tre tipi di popolamento degli indici.

- *Popolamento completo*: si verifica, in genere, quando un indice full-text viene popolato per la prima volta. Gli indici possono quindi essere gestiti utilizzando popolamenti incrementali o basati sul rilevamento delle modifiche.
- *Popolamento basato sul rilevamento delle modifiche*: SQL Server Express registra le righe che sono state modificate in una tabella impostata per l'indicizzazione full-text. Tali modifiche vengono propagate all'indice full-text.
- *Popolamento incrementale basato su timestamp*: aggiorna l'indice full-text relativamente alle righe aggiunte, eliminate o modificate dopo l'ultimo popolamento o durante la sua esecuzione. Per eseguire il popolamento incrementale è necessario che la tabella indicizzata contenga una colonna del tipo di dati timestamp. Se non è disponibile una colonna timestamp, non è possibile eseguire il popolamento incrementale. Se viene richiesto un popolamento incrementale su una tabella senza una colonna timestamp, verrà eseguito un popolamento completo.

2.4.2 Installazioni del database e relative istanze

Su uno stesso computer possono coesistere più installazioni di SQL Server Express, insieme ad altre installazioni di SQL Server 2000, SQL Server 2005 o MSDE (Microsoft Desktop Engine). È previsto un limite massimo di 50 istanze SQL Server Express sullo stesso computer, a ciascuna delle quali deve essere attribuito un nome univoco per consentirne l'identificazione.

Per impostazione predefinita, SQL Server Express installa l'istanza denominata *sqlexpress*, che può essere condivisa tra più applicazioni. Alcuni requisiti di installazione, ad esempio l'autenticazione SSL (Secure Socket Layer), interessano l'installazione nel suo complesso e richiedono pertanto l'installazione di un'istanza denominata a parte. In tutti gli altri casi è in genere sufficiente l'istanza condivisa. L'utilizzo dell'istanza condivisa offre anche un altro vantaggio, perché evita di dover installare SQL Server Express insieme alle applicazioni di altri fornitori di software, semplificando l'installazione di queste ultime.

2.4.3 Application Programming Interface

Le API disponibili per la realizzazione di applicazioni basate su SQL Server Express sono le stesse di SQL Server 2005 e questo consente agli utenti di passare in modo completamente trasparente ad altre edizioni previste nelle licenze di SQL Server 2005. Sono supportate anche tutte le altre nuove caratteristiche di SQL Server 2005, come l'integrazione con CLR (Common Language Runtime), i nuovi tipi di dati quali VARCHAR(MAX) e XML, i tipi definiti dall'utente e le aggregazioni definite dall'utente. Inoltre, i database di SQL Server Express possono essere collegati a SQL Server 2005, e le applicazioni realizzate, utilizzando un'istanza di SQL Server Express, funzionano perfettamente anche con un'istanza di SQL Server 2005.

2.4.4 Supporto della connettività di rete

Per impostazione predefinita, SQL Server Express è in grado di accedere solo alla memoria condivisa del computer locale, ma l'utente può attivare in modo esplicito anche altri protocolli supportati, ad esempio TCP/IP e Named Pipes. I protocolli VIA e HTTP non sono supportati in SQL Server Express. Poiché per impostazione predefinita è disponibile solo la memoria condivisa, per consentire ai computer remoti di connettersi a SQL Server Express è necessario attivare la connettività di rete. Per attivare la connettività di rete è possibile:

- Utilizzare l'applicativo SQL Server Management Studio per attivare i protocolli necessari e avviare SQL Browser, entrambe utilità fornite nel pacchetto SQL Server Express.
- Se si sa fin dall'inizio che è necessario il supporto della connettività di rete, utilizzare `DISABLENETWORKPROTOCOLS=0` nella riga di comando del programma di installazione.

SQL Browser è servizio di SQL Server 2005 che consente di identificare le porte su cui restano in attesa le istanze denominate. Poiché non viene utilizzato dalla memoria condivisa, in SQL Server Express questo servizio è disattivato per impostazione predefinita. Per consentire l'accesso di rete è pertanto necessario avviare tale servizio.

Infine, è importante sottolineare che SQL Browser rimane in ascolto sulla porta UDP (User Datagram Protocol) 1434. Grazie al protocollo UDP, le applicazioni residenti in una rete interna di computer posso inviare ad altre piccoli pacchetti di informazioni, detti *datagrammi*, in modo molto veloce.

2.4.5 Supporto di accesso ai dati

SQL Server Express supporta gli stessi provider nativi e gestiti delle altre edizioni di SQL Server 2005. Ciò costituisce un grosso vantaggio, poiché le applicazioni scritte per il primo sono in grado di interagire in modo completamente trasparente con le altre edizioni di SQL Server.

2.4.5.1 Multiplexing

A partire da SQL Server 2005, le sessioni logiche nel server sono scollegate dalle connessioni fisiche. I livelli di trasporto di client e server sono stati infatti aggiornati in modo da offrire funzionalità di *multiplexing*, consentendo la gestione di più sessioni logiche su una singola connessione fisica. In questo modo i client possono disporre di più MARS (Multiple Active Result Set) associati alla stessa connessione. Si noti tuttavia che la tecnologia MARS non ha lo scopo di eliminare la necessità di utilizzare più connessioni in generale. In SQL Server Express il servizio MARS è attivo per impostazione predefinita e consente di intercalare le operazioni SQL. È ad esempio possibile utilizzare un set di risultati ed eseguire un'istruzione sul database mentre è in corso l'elaborazione, senza aprire una nuova connessione. MARS è in grado di sostituire efficacemente i cursori nella maggior parte degli scenari e risulta particolarmente utile quando tutte le operazioni di recupero dei dati e aggiornamento avvengono nell'ambito della stessa transazione.

2.4.5.2 MDAC e SQL Native Client

I componenti di accesso ai dati sono suddivisi in due gruppi: lo stack MDAC, che fa parte del sistema operativo, e il provider SQL Native Client, che fornisce librerie di accesso ai dati specifiche di SQL Server Express per l'accesso ai dati

nativo. Alcune caratteristiche e funzionalità di SQL Server 2005, quali il servizio MARS, i tipi definiti dall'utente, il tipo di dati XML e così via, sono disponibili tramite SQL Native Client solo per la programmazione in C o C++. SQL Native Client è costituito da una singola DLL (*sqlncli.dll*). Per i componenti che non fanno parte del sistema operativo la manutenzione è più semplice, così come la redistribuzione e lo sviluppo delle applicazioni.

2.4.5.3 Interazioni tra MDAC e SQL Native Client

Esistono alcune dinamiche interessanti basate sulle interazioni tra i provider MDAC e i provider SQL Native Client. Ad esempio, MDAC non è in grado di connettersi a SQL Server Express tramite memoria condivisa. Questo problema interessa tutte le applicazioni native che utilizzano OLE DB o ODBC, oltre alle applicazioni native esistenti.

OLE DB è una API sviluppata da Microsoft che permette di accedere a gruppi di dati diversi in maniera uniforme.

ODBC (Open Database Connectivity) è una API che permette di utilizzare un DBMS su un qualsiasi tipo di file di database.

2.4.6 Supporto all'I/O

SQL Server Express supporta le operazioni di input/output asincrone, pertanto i thread delle applicazioni non vengono più bloccati dalle operazioni di trasferimento dei dati e le applicazioni client possono rispondere con la massima rapidità. In questo modello le chiamate di I/O vengono completate immediatamente e il completamento è notificato in modo asincrono all'applicazione, che può così continuare ad eseguire altre attività di

elaborazione e verificare il completamento dell'operazione di I/O in un momento successivo.

2.4.7 Protezione

Tra gli obiettivi di progettazione di SQL Server Express è incluso quello di fornire impostazioni predefinite sicure per i diversi componenti. Ad esempio, protocolli di rete quali TCP/IP e Named Pipes sono disattivati per impostazione predefinita. Il servizio SQL Browser viene avviato solo se l'utente lo richiede esplicitamente nella riga di comando del programma di installazione.

Se viene utilizzata l'autenticazione di Windows, l'account *sa*, ovvero l'account dell'amministratore di sistema, è disattivato per impostazione predefinita. I normali utenti del computer dispongono solo di privilegi minimi per l'istanza di SQL Server Express. Per consentire a tali utenti di utilizzare le funzionalità SQL, è necessario che un amministratore locale sul server conceda esplicitamente le autorizzazioni appropriate.

2.4.7.1 Account sa

In SQL Server, *sa* è un account di accesso speciale, membro del ruolo *sysadmin* (amministratore del sistema). Viene utilizzato soprattutto nelle configurazioni che utilizzano l'autenticazione di SQL Server, mentre non viene utilizzato con l'autenticazione di Windows. Per motivi di sicurezza, SQL Server richiede una password complessa per l'account *sa*, che deve essere specificata sia quando si esegue l'installazione utilizzando l'interfaccia grafica, sia quando si esegue un'installazione automatica che specifica la modalità di autenticazione di SQL Server. Se invece si esegue un'installazione automatica che specifica la modalità di autenticazione di Windows, la password per l'account *sa* non è necessaria. Quando si utilizza l'autenticazione di Windows, infatti, se l'utente non specifica

una password per l'account *sa*, la procedura di installazione automatica di SQL Server Express genera una password complessa casuale. In questo caso, inoltre, il programma di installazione disattiva l'account *sa* e per utilizzarlo è necessario attivarlo in modo esplicito in un momento successivo.

2.4.8 Supporto per la replica dei dati

Grazie alla replica è possibile conservare copie dei dati in più postazioni, tramite un modello server di pubblicazione-sottoscrittore, con sincronizzazione delle copie a intervalli stabiliti dall'utente. SQL Server Express supporta la sottoscrizione a pubblicazioni di tipo *merge* (integrazione dei dati), *snapshot* (copia identica dei dati) e *transazionali* (utilizzando il file di log), ma non può essere utilizzato per eseguire operazioni di pubblicazione. In SQL Server Express le sottoscrizioni di replica sono dotate di funzionalità complete.

2.4.9 XCopy di applicazioni

SQL Server Express include una nuova funzionalità *XCopy* per le applicazioni, che consente di gestire i database come semplici file. Grazie a questa funzionalità, il database locale può essere spostato, copiato o inviato per posta elettronica insieme all'applicazione e, nel nuovo sistema, può essere utilizzato senza eseguire alcuna operazione di configurazione aggiuntiva. Il supporto della funzionalità. *XCopy* si basa principalmente sulle tre funzionalità e caratteristiche seguenti: l'opzione *AttachDBFilename* della stringa di connessione, il fatto che non è necessario specificare il nome logico del database e la funzionalità di chiusura automatica.

La funzionalità *XCopy* si basa su alcuni presupposti impliciti. Infatti, viene presupposto che nel computer sia presente un'istanza di SQL Server Express

con il nome *sqlexpress*. Inoltre, poiché si tratta di una soluzione basata sullo stack gestito, per sviluppare applicazioni di cui sia possibile eseguire la distribuzione tramite *XCopy*, è necessario utilizzare il provider di dati .NET per SQL Server. In altre parole, non è possibile utilizzare SQL Native Client o MDAC per sviluppare applicazioni che possano essere utilizzate dalla funzionalità *XCopy*. Lo sviluppatore dell'applicazione si limita in genere a copiare il database utente e il file di log insieme all'applicazione. In SQL Server è tuttavia presente uno speciale database di sistema, detto MASTER, che contiene alcune impostazioni di configurazione. Le funzionalità che si basano sulle voci del database MASTER includono l'autenticazione di SQL Server. Se l'applicazione da copiare si basa sulle voci del database MASTER, lo sviluppatore deve verificare che nel sistema di destinazione vengano replicate anche tali voci di configurazione. Per la maggior parte delle applicazioni che utilizzano l'autenticazione di Windows, invece, la replica delle informazioni di tale database non costituisce un problema.

2.4.9.1 Opzione AttachDBFilename

Nell'opzione *AttachDBFilename* della stringa di connessione è possibile specificare un percorso di file relativo o assoluto. Il file di database specificato viene collegato all'apertura della connessione e tale database viene utilizzato come database predefinito per la connessione. Questa parola chiave supporta la stringa speciale, che in fase di esecuzione punta alla directory dati dell'applicazione in cui sono archiviati i file di database. Questa stringa speciale deve trovarsi all'inizio del percorso di file e può essere utilizzata solo con un file system locale. L'opzione *AttachDBFilename* richiede alcune modifiche nell'utilizzo dei file di log. Il nome del file di log deve avere formato *<nome file del database>_log.ldf* e, quando si utilizza *AttachDBFilename*, non sono disponibili

opzioni per specificare il nome del file di log: non sono pertanto supportati i nomi definiti dall'utente.

2.4.9.2 Nome logico del database

Se nella stringa di connessione non è specificato il nome logico del database, viene automaticamente generato un nome per il database da collegare. Il nome generato è basato sul percorso relativo del file con estensione *.mdf*, e viene ottenuto richiamando la funzione intrinseca incorporata *fn_GenDBNameFromPath*.

Questa caratteristica è utile quando si sposta o si copia un database nello stesso computer, perché il nome logico basato sul percorso del file è univoco. Senza questa caratteristica, se si utilizza lo stesso nome logico per aprire database in due directory diverse, si verificano conflitti di denominazione in SQL Server.

2.4.9.3 Funzionalità di chiusura automatica

La funzionalità di chiusura automatica era disponibile già in SQL 2000, e in SQL Server Express è attivata per impostazione predefinita. Tale funzionalità rilascia i blocchi a livello di file sui database utente, quando non sono presenti connessioni attive a questi ultimi. Il database è pertanto pronto per lo spostamento o per la copia subito dopo la chiusura dell'applicazione che lo utilizza.

Dal punto di vista dell'utente, tuttavia, la chiusura automatica non funziona sempre, specialmente in caso di arresto imprevisto del client oppure quando si utilizza un pool di connessioni. In caso di arresto imprevisto del client, dopo qualche tempo le connessioni vengono eliminate automaticamente dal meccanismo di timeout. Se invece è attivato il pooling delle connessioni,

quando l'utente richiama la chiusura di una connessione, quest'ultima non viene effettivamente chiusa, ma viene restituita al pool. Per chiudere tutte le connessioni è in genere necessario chiudere il dominio applicazione oppure tutte le connessioni aperte.

2.5 Oracle Database 10g Express Edition (dicembre 2005)

Oracle Database 10g Express Edition, abbreviato in Oracle Database XE, è la versione gratuita del già provato Oracle Database 10g. Come nel caso di SQL Server Express, anche in questo prodotto sono imposte delle limitazioni:

- supporto di una sola CPU
- limite di 1 GB di memoria RAM utilizzabile
- dimensione massima dei file di dati limitata a 4 GB

Oracle Database XE è stato costruito sulla stessa tecnologia del codice di Oracle Database 10g Release 2. L'amministrazione del database è demandata ad un'interfaccia browser, e le applicazioni possono essere sviluppate tramite interfacce standard quali SQL, JDBC, ODP.NET (Oracle Data Provider per .NET). Le utilità di BI sono completamente assenti.

2.5.1 Struttura logica del database

Il database Oracle è composto da due principali componenti: il database e l'istanza. Il database è l'insieme dei file che fisicamente contengono i dati. L'istanza è invece l'insieme dei processi attivi in background e delle relative aree di memoria utilizzate che permettono l'accesso ai dati contenuti nel database. Ogni istanza è associata ad un solo database (modalità tipica), mentre un database può essere associato a più istanze (in Oracle Database XE, modalità

Parallel Server o Real Cluster Application). Su un server possono essere presenti più coppie istanza-database. Sul medesimo server ogni istanza deve avere un nome univoco. È quindi importante sottolineare la distinzione esistente nella terminologia Oracle: il database è esclusivamente l'insieme dei file dati sul server. Spesso erroneamente si fa riferimento ad un database Oracle nell'accezione dell'applicativo utilizzato: in realtà si fa riferimento al software Oracle Server .

Esiste una struttura logica (rappresentata dai componenti del database: le tabelle, gli indici e altri elementi) che si sovrappone ad una struttura fisica (i file fisici). Questa distinzione permette la portabilità logica di un database da una piattaforma ad un'altra con hardware e/o sistemi operativi differenti senza difficoltà alcuna.

2.5.1.1 Block

Un *block* (blocco) è l'unità base di storage di Oracle Database XE. È raccomandato che un blocco sia un multiplo intero della dimensione di un blocco fisico del file-system del sistema operativo al fine di aumentare le performance di lettura/scrittura. La dimensione in byte di un block viene impostata alla creazione del database (parametro DB_BLOCK_SIZE del file *init.ora*) ed è uno dei pochi parametri di Oracle che non può essere mai modificato una volta creato il database.

2.5.1.2 Extent

Un *extent* è un gruppo di blocchi consecutivi. Gli extent sono associati agli oggetti (per esempio le tabelle) al momento della loro creazione e quando gli

oggetti crescono di dimensione. Di norma un extent non può essere condiviso da più oggetti.

2.5.1.3 Segment

Un *segment* o segmento è un insieme di più extent, non necessariamente contigui. Gli oggetti logici, che allocano i dati del database, sono associati ad un segmento. Una volta che un extent del segmento è saturo di dati, ne viene automaticamente allocato un altro. Esistono quattro tipi di segment, definiti come tali alla creazione degli oggetti logici cui sono associati:

- *Data segment*, relativi alle tabelle
- *Index segment*, relativi agli indici
- *Temporary segment*, creati ed eliminati dinamicamente, in base alle necessità delle operazioni eseguite, ad esempio query particolarmente onerose
- *Roll-back segment*, utilizzati per l'operazione di roll-back e per mantenere la consistenza dei dati nelle successive sessioni.

2.5.1.4 Tablespace

Un *tablespace* è costituito da più segmenti. Esiste sempre almeno il tablespace SYSTEM, che contiene il *Data Dictionary* e tutte il codice immagazzinato nel database, ad esempio procedure, funzioni, packages e altre applicazioni scritte in PL/SQL oppure altri linguaggi, di tutti gli utenti o schemi. Un database di Oracle è formato da uno o più tablespace.

2.5.1.5 Schema

Uno *schema* è un'entità logica che possiede oggetti del database: tabelle, indici, procedure e molti altri oggetti. Gli oggetti appartenenti ad uno schema possono essere locati in più tablespaces. Un tablespace può contenere oggetti di schemi diversi. Ad ogni schema è associato un *user*, cioè l'utente che si connette al database.

2.5.1.6 Indici

Quando l'utente crea un indice, Oracle alloca automaticamente un segmento per salvare i dati dell'indice in un tablespace.

Oracle Database XE utilizza la tecnica degli alberi binari per velocizzare le ricerche di dati tramite l'indice. Esso è organizzato in *branch block* (blocchi superiori) e *leaf block* (blocchi inferiori): i primi contengono i puntatori ai leaf block, mentre i secondi contengono tutti i valori delle colonne indicizzate e un corrispondente identificativo di riga che localizza la tupla corrispondente nel database.

2.5.2 Struttura fisica del database

Nello standard OFA (Optimal Flexible Architecture), cioè lo standard ottimale per l'amministrazione dei file e dei task di Oracle Database XE, è assunto come *init.ora* il file che contiene i parametri di inizializzazione dell'istanza. Quasi tutti i parametri potrebbero, in realtà, essere trascurati (persino il nome del database ha un valore specifico di default). Va detto però che il tuning del database parte proprio da un corretto dimensionamento dei parametri elencati in questo file. Questi valori dimensionano la memoria allocata dai processi in background, e il numero di processi stessi che saranno avviati: infatti, diversi tipi di processi

possono operare in parallelo allo scopo di migliorare le prestazioni e la sicurezza del database. Tale file è salvato in formato ASCII ed è semplicemente modificabile per mezzo di un editor di testo. Qualora si cambiasse uno qualsiasi dei suoi valori, diviene necessario riavviare l'istanza al fine di poter rendere i cambiamenti attivi.

Diverso è invece lo *spfile* (file di spool) introdotto dalla versione 9i di Oracle. Il file *spfile_<nomeistanza>.ora* è un file binario che viene modificato tramite comandi SQLPlus. Le modifiche apportate all'istanza sono immediate e non necessitano di una chiusura – riavvio del server Oracle Database XE. Da un *init.ora* è possibile creare uno *spfile* e viceversa tramite comandi dedicati di SQLPLUS.

2.5.2.1 Data file

I *Data file* contengono i dati del database (normalmente hanno estensione *.dbf*). In un database esiste almeno un data file. Un data file è associato ad un solo tablespace, mentre un tablespace può essere costituito da più data file.

2.5.2.2 Redo Log file

I *Redo Log file* registrano tutti i comandi DML e DDL che vengono eseguiti dagli utenti connessi al database.

Il loro funzionamento si spiega con il seguente esempio: si supponga di eseguire una query di update con relativo commit. I dati interessati vengono prima recuperati dal data file, leggendo gli *n* blocchi necessari, portati in memoria (precisamente la SGA) e qui, finalmente, modificati. A questo punto i Redo Log file sono stati già scritti, mentre nel Data file relativo ci sono ancora i dati originali. Infatti, Oracle Database XE per velocizzare le operazioni scrive

sui Data file solo al verificarsi di determinate condizioni. Quindi, in caso di arresto del sistema, prima ancora che i dati vengano registrati sul data file, si perderebbe l'update: il database non sarebbe più consistente. Al riavvio, Oracle autodiagnostica una chiusura del server non convenzionale, ed allora legge i Redo Log file per eseguire un ripristino del database. I dati vengono scritti ora nei Data file interessati assicurando la consistenza dei dati.

I Redo Log sono file sequenziali: la loro scrittura-lettura è quindi enormemente più veloce rispetto ai Data file, che hanno un accesso casuale e possono essere frammentati. Questo è il motivo per il quale l'OFA ritiene importante riservare un disco per la loro allocazione: essendo costantemente interessati da operazioni di scrittura è bene che la risorsa preziosa del processo (la testina dell'hard disk) sia perennemente a loro dedicata.

Ogni database deve avere almeno due gruppi di Redo Log. Un *Redo Log Group* è formato da uno o più *Redo Log Member*, file identici possibilmente posizionati su supporti differenti. Quando un Redo Log è completo il relativo processo (*lgwr*) inizia a scrivere sul successivo e così via fino a che non inizia a sovrascrivere i dati del primo Redo Log: questa sequenza è necessaria al fine di archiviare i Redo Log file col processo *arch* nel caso in cui il server operi in modalità *archive mode*.

2.5.2.3 Control file

I *Control file* sono file binari generati alla creazione del database. Vengono aggiornati costantemente e contengono la struttura fisica del database quali la locazione dei Data file e dei Redo Log file, il nome del database, la data di creazione, i nomi dei tablespaces, il numero unico di sequenza dei Redo Log, le informazioni sull'ultimo stato dei dati insieme all'SCN (System Change

Number) e, i dati necessari al funzionamento dell'*rman* (Recovery Manager) di Oracle Database XE.

Ogni database ha almeno un Control file, ma l'OFA raccomanda di duplicarlo su almeno un secondo supporto: la perdita dei Control file comprometterebbe il funzionamento dell'intero database.

2.5.3 Gestione della memoria

Oracle Database XE gestisce la memoria secondo due strutture principali: la SGA (System Global Area), memoria condivisa da tutti gli utenti, e la PGA (Program Global Area), memoria riservata per ogni singolo processo utente.

2.5.3.1 Memoria SGA

La SGA è allocata all'avvio dell'istanza tramite il comando STARTUP NOMOUNT, e deallocata alla chiusura dell'istanza stessa. Ha una struttura relativamente complessa, suddivisa in tre componenti principali.

La prima componente è il *Database Buffer Cache*, che memorizza i blocchi di dati utilizzati più di recente dalle varie istruzioni SQL. Avendo una dimensione finita, ovviamente limitata superiormente dalla dimensione della RAM installata nella macchina server, è gestita da appositi algoritmi LRU (Least Recently Used) che permettono di eliminare gli oggetti creati e usati meno di recente, e di mantenere in memoria, invece, i dati recenti o comunque di frequente lettura: statisticamente, infatti, sarà più probabile un loro successivo riutilizzo. Il mantenere in memoria questi dati evita al sistema le continue letture su disco, collo di bottiglia delle performance del sistema, e per questo motivo è importantissimo il tuning di questa grandezza. Esistono tre tipi Database Buffer Cache:

- *Dirty buffer*, blocchi di dati con dati modificati ma non ancora scritti sui tablespace, mentre il relativo log è già stato salvato sui Redo Log file
- *Free buffer*, blocchi liberi o con dati che possono essere sovrascritti
- *Pinned buffer*, blocchi di dati attualmente usati o riservati per particolari transazioni

La seconda componente è lo *Shared Pool* che, a differenza della precedente che memorizza i dati contenuti nelle tabelle, memorizza il codice SQL e PL/SQL utilizzato più di recente. E' suddivisa in due sezioni:

- *Library Cache*, gestita da un algoritmo LRU, è suddivisa ulteriormente in tre parti: una contiene il codice delle istruzioni SQL, una le procedure, le funzioni e i package PL/SQL eseguiti più di recente, e l'ultima viene utilizzata per memorizzare i lock ed altri dati utilizzati dall'istanza per gestire le sessioni concorrenti. In merito alla cache di un package, è importante rilevare quanto continuo le dimensioni dello stesso. Infatti, se si richiama una procedura oppure una funzione di un package di dimensioni considerevoli, quest'ultimo è caricato totalmente nella Library Cache, obbligando il processo di gestione a liberare spazio nell'area di pool. Inoltre, si obbliga il sistema a perdere così il riferimento a numerosi altri oggetti qui contenuti: tutto questo influisce negativamente sulle performance dell'architettura Oracle.
- *Data Dictionary Cache*, memorizza dati inerenti gli oggetti del database, la loro struttura e i privilegi associati agli utenti.

La terza componente è il *Redo Log Buffer*, che mantiene traccia delle modifiche effettuate ai dati del database ed alimenta la scrittura dei Redo Log file.

2.5.3.2 Memoria PGA

La memoria PGA non viene cancellata fino a quando rimane attiva la connessione dell'utente (questo almeno in modalità *Dedicated Server*). Essa è gestita diversamente a seconda della modalità di funzionamento di Oracle Database XE: *Dedicated Server* o *MultiThreaded Server*.

Quando il sistema opera in modalità *Dedicated Server*, il server alloca un processo per ogni connessione che viene stabilita con il database, sia che la connessione abbia origine da un'utente allocato sul server stesso, sia che la connessione venga richiesta tramite Oracle Net da un'applicazione client. Al processo è associata un'area di memoria non condivisa, detta appunto PGA. Sono presenti due aree diverse:

- *Stack Space*, dove sono allocate le informazioni relative alle variabili, ai cursori, e a tutti i componenti indispensabili alla sessione corrente
- *Sort Area*, dove sono eseguiti gli ordinamenti richiesti in fase di formulazione dell'istruzione SQL (GROUP BY, DISTINCT e UNION, MINUS o INTERSECT).

Quando, invece, il sistema opera in modalità *MultiThreaded Server*, l'utente si connette ad un processo condiviso con altri operatori al fine di ottimizzare l'utilizzo delle risorse. Infatti, un processo dedicato è utilizzato per una percentuale temporale ridotta, soprattutto nel caso in cui il database supporti applicazioni OLTP e la memoria allocata dalla PGA non venga quindi utilizzata al meglio, comportando così uno spreco di risorse hardware. In questa seconda modalità la PGA mantiene lo *Stack Space*, condiviso fra le varie sessioni connesse; la *Sort Area* è invece allocata nella SGA e, più precisamente, nella *Large Pool*.

2.5.4 Gestione dei processi

In generale, un processo è utilizzato dal sistema operativo per effettuare una serie di operazioni. Oracle Database XE avvia e controlla diversi processi ogni qualvolta viene avviata un'istanza. Esistono due gruppi di processi:

- *User Process*, avviato sul client non appena un'applicazione richiede un collegamento al database, caratterizzato da una connessione e da una sessione.
- *Oracle Process*, suddividono in *Server process* e *Background process*.

I *Server process* si prendono in carico le richieste degli *User process*, elaborando le seguenti principali operazioni:

- parsing ed esecuzione di codice SQL
- trasferimento dei data block dai data file alla cache del database, nel caso in cui i dati necessari in output non siano già presenti in memoria
- rilascio dati di output

I *Background process* sono numerosi: alcuni sono sempre presenti, alcuni sono opzionali, altri ancora dipendono dalla configurazione prescelta (come nel caso del Dedicated server). Tutti questi processi hanno la possibilità di scrivere, in caso di errori o di dump, dei trace file e di alimentare l>alert file, log cronologico degli errori più significativi e degli eventi più importanti che si avvicinano nel tempo di vita dell'istanza. Questi file sono fisicamente situati nel percorso specificato dal parametro BACKGROUND_DUMP_DEST.

2.5.5 Oracle Net

Un client è un'applicazione che si connette al server di Oracle Database XE, per spedire e ottenere informazioni. Un'applicazione client può risiedere in ogni macchina a patto che il software Oracle sia correttamente installato.

Oracle Net è un componente software che risiede nel server del database. Stabilisce e mantiene le connessioni tra applicazioni client e server, e provvede allo scambio di informazioni usando i protocolli di rete standard, quali TCP/IP e Named Pipes.

Nella versione 10g, se non viene configurato manualmente, un *listener* è avviato automaticamente sulla porta 1521, e utilizza il protocollo TCP/IP.

2.5.5.1 Oracle Net listener

Nel server di database, l'*Oracle Net listener*, meglio noto come *listener*, è un processo che "ascolta" le richieste della connessione client. La configurazione predefinita del listener è detta *listener.ora*. Questo file contiene un indirizzo che identifica il database: esso definisce il protocollo del listener stesso, l'host della macchina su cui risiede, e la porta di comunicazione. Per valori predefiniti, il protocollo standard del listener è il TCP/IP, e lo stesso listener viene avviato automaticamente con l'istanza del server.

2.5.5.2 Configurazione del client

Quando un client richiede una connessione, il listener del server cattura la richiesta e la indirizza al database Oracle.

Il client utilizza un descrittore di connessione per specificare a quale database è richiesta la connessione. Tale descrittore contiene il nome del protocollo

utilizzato e il nome del servizio del database, poiché un database può avere definiti più servizi.

I descrittori sono definiti nel file *tnsnames.ora* nella macchina del client; per configurarlo, è possibile usare Oracle Net Manager.

Un client richiede la connessione al server tramite una stringa di connessione: essa deve contenere un nome utente, una password e un identificativo di connessione. L'identificativo può essere lo stesso descrittore, o un nome che comunque riconduca al descrittore.

2.5.6 Sicurezza

Per gli utenti che accedono al database, l'amministratore dello stesso deve creare gli account e garantire ad ognuno privilegi di accesso appropriati. Ciascun account è identificato da un nome utente e da alcuni attributi di sicurezza, inclusi i seguenti:

- metodo di autenticazione
- password criptata per l'autenticazione al database
- accesso ai tablespace
- spazio utilizzabile sui tablespace
- attributo che specifica se l'account è bloccato oppure no.

Gli account sono generalmente creati da un amministratore predisposto alla sicurezza del sistema, che deve assegnare i privilegi base per permettere all'utente di connettersi al database e creare degli oggetti, detti *oggetti dello schema*.

2.5.6.1 Privilegi e ruoli

Privilegi di sistema, privilegi di oggetto e ruoli provvedono a fornire un livello di sicurezza minimo del database. Sono designati a controllare l'accesso ai dati e limitare i tipi di istruzioni SQL eseguibili dagli utenti. I ruoli, come già descritto nei precedenti paragrafi, sono gruppi di privilegi.

Per assegnare privilegi e ruoli ad altri account, l'amministratore deve possedere a sua volta le autorizzazioni di Oracle Database XE per farlo. Infatti, quando si crea un database Oracle, automaticamente al DBA (amministratore di sistema) vengono assegnati due super account, SYS e SYSTEM, con privilegi di sistema molto simili.

3. Tool per il trasferimento del database originario

La varietà di architetture di ciascuno degli RDBMS candidati rende necessaria l'implementazione di un applicativo che traduca i metadati iniziali (Firebird 1.5) alla sintassi del nuovo database di destinazione (scelto tra Oracle Database XE, SQL Server 2005 Express, e lo stesso Firebird in versione 1.0 o 2.0).

Bisogna premettere che il database e l'utente (con relativi account, password, ruoli e privilegi di accesso) di destinazione è già creato dall'utente amministratore, quindi il tool copia i dati originari in un database esistente, ma completamente privo di tabelle e dati.

Un altro fatto essenziale è che il motore del DBMS sorgente o destinatario oppure entrambi potrebbero non risiedere nello stesso pc con cui si sta lavorando, cioè eseguendo il programma: il tool di trasferimento tiene conto di questa possibilità e quindi lavora indistintamente su un pc client o direttamente sul server dove risiedono i DBMS engine, e questi ultimi possono risiedere su server diversi.

L'applicazione è stata interamente sviluppata con Java 1.4.0-b92 e le componenti principali sono le seguenti:

- interfaccia utente
- connessioni ai database
- routines di trasferimento dati
- corpo dell'applicazione.

Esse sono state raccolte in altrettanti packages all'interno del programma e saranno analizzate singolarmente nei prossimi paragrafi.

I file sorgente, una volta compilati, sono stati raggruppati in un archivio (*ToolTravaso.jar*) in modo da rendere più semplice e veloce l'esecuzione dell'applicazione.

3.1 Interfaccia utente (package IO)

Contiene due classi: *Input* e *Output*. Tutti i parametri necessari per connettersi al database sorgente ed a quello di destinazione vengono raccolti dalla prima, mentre la seconda elabora e visualizza una serie di statistiche relative all'andamento del travaso vero e proprio dei dati.

3.1.1 Classe Input

Lanciando il programma vengono richiesti all'utente i parametri necessari per inizializzare i driver delle connessioni ai due database, sorgente e finale (*Fig. 3.1*). In seguito è riportata parte del codice Java utilizzato.

```
package ToolTravaso.IO;
import java.io.*;

public class Input {
    /*
        [0] database di destinazione -> 1 = Firebird 1.0 / 2.0
                                   -> 2 = Oracle Database 10g Express
                                   -> 3 = Sql Server 2005 Express
        [1] host del database di origine
        [2] percorso del file database di origine
        [3] username del database di origine
        [4] password del database di origine
        [5] host del database di destinazione
        [6] percorso del file database di destinazione (solo per Firebird X.X)
        [7] nome del database di destinazione (solo per Sql Server 2005)
        [8] username del database di destinazione
        [9] password del database di destinazione
        [10] creazione ed eventuale percorso del file di log
    */
    private static String [] input = new String [11];
    private static boolean log = true;
```

```
// costruttore della classe Input
public Input() {
    try {
        /*
         * l'array input viene popolato con i valori immessi dall'utente, e letti per
         * mezzo della funzione leggiRiga
         */
    }
    catch(Exception ex){ /* gestione dell'eventuale eccezione */ }
}

// metodo che legge una riga dall'input standard (tastiera)
private static String leggiRiga() throws IOException {
    BufferedReader In = new BufferedReader(new InputStreamReader(System.in));
    return formattaRiga(In.readLine());
}
/*
 * definizioni di altri metodi della classe Input
 */
}
```

Preliminarmente, l'utente deve scegliere il database di destinazione affinché il programma carichi le routines adatte alla sintassi del destinatario per effettuare la migrazione dei dati. In seguito vengono richiesti all'utente, prima per l'originario poi per il destinatario, l'host su cui risiede il motore del DBMS (indirizzo IP o relativo alias di rete), il percorso del file nell'host appena specificato in cui sono memorizzati i dati (per il database finale è richiesto solo se si tratta di Firebird 1.0 o 2.0), il nome utente e la password. In caso i dati siano destinati a SQL Server, bisogna immettere anche il nome del database finale.

L'ultima richiesta fatta da questa classe è l'eventuale creazione (e in caso affermativo il percorso) del file di riepilogo degli eventi al termine del trasferimento (file di *log*).

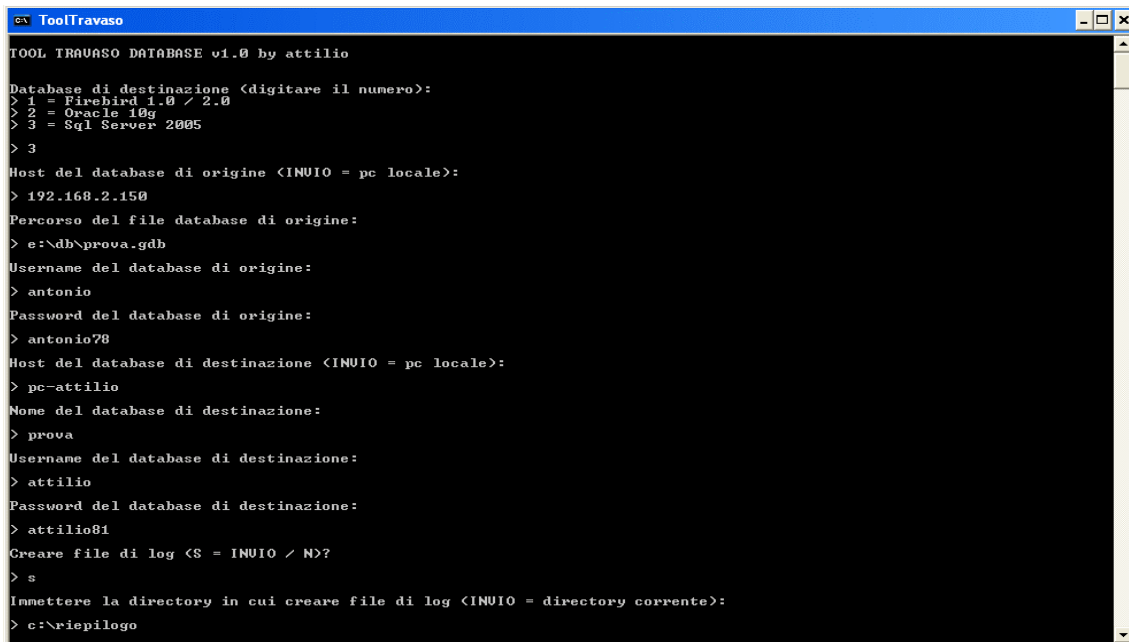


Fig. 3.1

3.1.2 Classe Output

Contiene tutti i metodi che visualizzano le informazioni a video durante l'esecuzione dell'applicazione (elenco tabelle da trasferire, creazione tabelle nel database destinatario, popolamento, aggiunta chiavi esterne e indici) e che creano le statistiche di riepilogo al termine dell'esecuzione (Figg. 3.2 e 3.3). Se richiesto, crea e popola il file di riepilogo con le suddette statistiche e descrizioni dettagliate degli eventuali errori occorsi durante il popolamento del database finale.

```
package ToolTravaso.IO;
import java.io.*;
import java.util.StringTokenizer;
import ToolTravaso.Connessioni.Connettore;

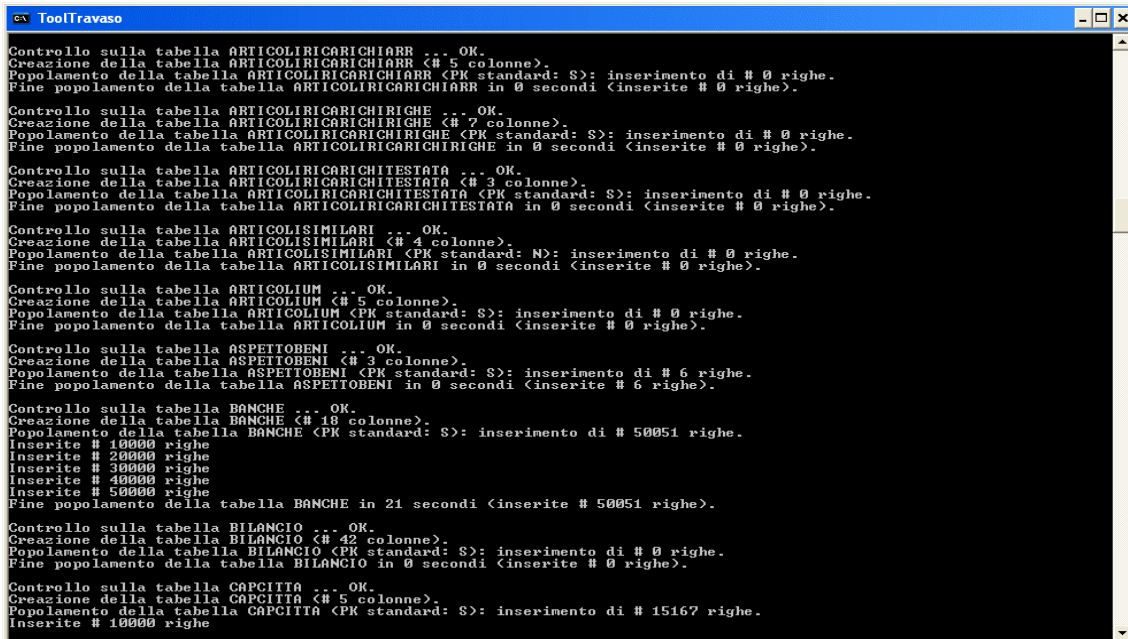
public class Output {

    // costruttore della classe Output
    public Output() {
        // creazione file di log (se richiesto)
        if(Input.creaLog()) {
            /*
             * ...
             */
            FileOutputStream file = new FileOutputStream(path);
```

```

PrintStream Out = new PrintStream(file);
/*
    scrittura nel file log
*/
}
/*
definizioni di altri metodi della classe Output
*/
}

```

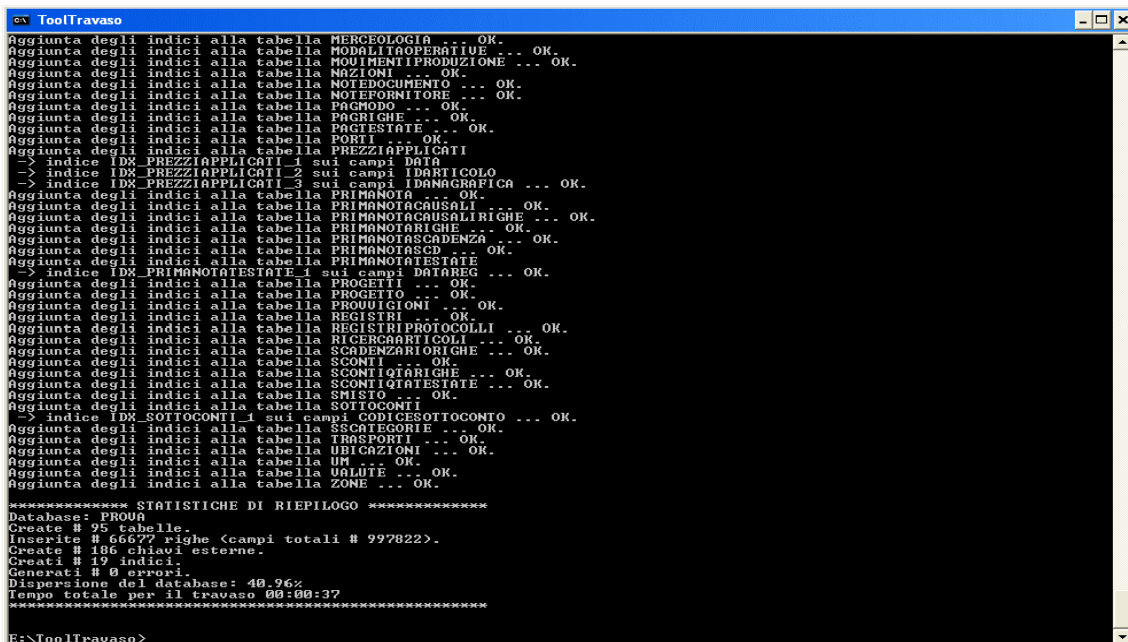


```

ToolTravaso
Controllo sulla tabella ARTICOLIRICARICHIARR ... OK.
Creazione della tabella ARTICOLIRICARICHIARR <# 5 colonne>.
Popolamento della tabella ARTICOLIRICARICHIARR <PK standard: S>: inserimento di # 0 righe.
Fine popolamento della tabella ARTICOLIRICARICHIARR in 0 secondi <inserite # 0 righe>.
Controllo sulla tabella ARTICOLIRICARICHIIRICHE ... OK.
Creazione della tabella ARTICOLIRICARICHIIRICHE <# 7 colonne>.
Popolamento della tabella ARTICOLIRICARICHIIRICHE <PK standard: S>: inserimento di # 0 righe.
Fine popolamento della tabella ARTICOLIRICARICHIIRICHE in 0 secondi <inserite # 0 righe>.
Controllo sulla tabella ARTICOLIRICARICHIITESTATA ... OK.
Creazione della tabella ARTICOLIRICARICHIITESTATA <# 3 colonne>.
Popolamento della tabella ARTICOLIRICARICHIITESTATA <PK standard: S>: inserimento di # 0 righe.
Fine popolamento della tabella ARTICOLIRICARICHIITESTATA in 0 secondi <inserite # 0 righe>.
Controllo sulla tabella ARTICOLISIMILARI ... OK.
Creazione della tabella ARTICOLISIMILARI <# 4 colonne>.
Popolamento della tabella ARTICOLISIMILARI <PK standard: N>: inserimento di # 0 righe.
Fine popolamento della tabella ARTICOLISIMILARI in 0 secondi <inserite # 0 righe>.
Controllo sulla tabella ARTICOLIUM ... OK.
Creazione della tabella ARTICOLIUM <# 5 colonne>.
Popolamento della tabella ARTICOLIUM <PK standard: S>: inserimento di # 0 righe.
Fine popolamento della tabella ARTICOLIUM in 0 secondi <inserite # 0 righe>.
Controllo sulla tabella ASPETTOBENI ... OK.
Creazione della tabella ASPETTOBENI <# 3 colonne>.
Popolamento della tabella ASPETTOBENI <PK standard: S>: inserimento di # 6 righe.
Fine popolamento della tabella ASPETTOBENI in 0 secondi <inserite # 6 righe>.
Controllo sulla tabella BANCHE ... OK.
Creazione della tabella BANCHE <# 18 colonne>.
Popolamento della tabella BANCHE <PK standard: S>: inserimento di # 50051 righe.
Inserite # 40000 righe
Inserite # 20000 righe
Inserite # 30000 righe
Inserite # 40000 righe
Inserite # 50000 righe
Fine popolamento della tabella BANCHE in 21 secondi <inserite # 50051 righe>.
Controllo sulla tabella BILANCIO ... OK.
Creazione della tabella BILANCIO <# 42 colonne>.
Popolamento della tabella BILANCIO <PK standard: S>: inserimento di # 0 righe.
Fine popolamento della tabella BILANCIO in 0 secondi <inserite # 0 righe>.
Controllo sulla tabella CAPCITTA ... OK.
Creazione della tabella CAPCITTA <# 5 colonne>.
Popolamento della tabella CAPCITTA <PK standard: S>: inserimento di # 15167 righe.
Inserite # 10000 righe

```

Fig. 3.2



```

ToolTravaso
Aggiunta degli indici alla tabella MERCEOLOGIA ... OK.
Aggiunta degli indici alla tabella MODALITAOPERATIVE ... OK.
Aggiunta degli indici alla tabella MOVIMENTIPRODUZIONE ... OK.
Aggiunta degli indici alla tabella NAZIONI ... OK.
Aggiunta degli indici alla tabella NOTEDOCUMENTO ... OK.
Aggiunta degli indici alla tabella NOTEFORMITORE ... OK.
Aggiunta degli indici alla tabella PAGMODOD ... OK.
Aggiunta degli indici alla tabella PAGRICHE ... OK.
Aggiunta degli indici alla tabella PAGTESTATE ... OK.
Aggiunta degli indici alla tabella PORTI ... OK.
Aggiunta degli indici alla tabella PREZZIAPPLICATI
-> indice IDX_PREZZIAPPLICATI_1 sui campi DATA
-> indice IDX_PREZZIAPPLICATI_2 sui campi IDARTICOLO
-> indice IDX_PREZZIAPPLICATI_3 sui campi IDANAGRAFICA ... OK.
Aggiunta degli indici alla tabella PRIMANOTA ... OK.
Aggiunta degli indici alla tabella PRIMANOTACASUALI ... OK.
Aggiunta degli indici alla tabella PRIMANOTACASUALIRICHE ... OK.
Aggiunta degli indici alla tabella PRIMANOTARICHE ... OK.
Aggiunta degli indici alla tabella PRIMANOTASCADEENZA ... OK.
Aggiunta degli indici alla tabella PRIMANOTASCD ... OK.
Aggiunta degli indici alla tabella PRIMANOTATESTATE
-> indice IDX_PRIMANOTATESTATE_1 sui campi DATAREG ... OK.
Aggiunta degli indici alla tabella PROGETTI ... OK.
Aggiunta degli indici alla tabella PROGETTO ... OK.
Aggiunta degli indici alla tabella PROUVIGIONI ... OK.
Aggiunta degli indici alla tabella REGISTRI ... OK.
Aggiunta degli indici alla tabella REGISTRIPROTOCOLLI ... OK.
Aggiunta degli indici alla tabella RICERCAARTICOLI ... OK.
Aggiunta degli indici alla tabella SCADENZARIORICHE ... OK.
Aggiunta degli indici alla tabella SCONTI ... OK.
Aggiunta degli indici alla tabella SCONTIQUARICHE ... OK.
Aggiunta degli indici alla tabella SCONTIQUATESTATE ... OK.
Aggiunta degli indici alla tabella SMISTO ... OK.
Aggiunta degli indici alla tabella SOTTOCONTI
-> indice IDX_SOTTOCONTI_1 sui campi CODICESOTTOCONTO ... OK.
Aggiunta degli indici alla tabella SECCATEGORIE ... OK.
Aggiunta degli indici alla tabella TRASPORTI ... OK.
Aggiunta degli indici alla tabella UBICAZIONI ... OK.
Aggiunta degli indici alla tabella UM ... OK.
Aggiunta degli indici alla tabella UMLE ... OK.
Aggiunta degli indici alla tabella ZONE ... OK.
***** STATISTICHE DI RIEPILOGO *****
Database: PROUA
Create # 95 tabelle.
Inserite # 66677 righe <campi totali # 997822>.
Create # 186 chiavi esterne.
Creati # 19 indici.
Generati # 0 errori.
Dispersione del database: 40.96%
Tempo totale per il travaso 00:00:37
*****
E:\ToolTravaso>

```

Fig. 3.3

3.2 Connessioni ai database (package Connessioni)

Contiene un sotto-package *Database* e una classe *Connettore*. Nel primo sono contenute quattro classi, ognuna per ogni tipo di database, aventi i rispettivi costruttori che inizializzano gli url di connessione con i parametri immessi in input; tale diversificazione è necessaria, poiché ogni RDBMS ha una stringa di connessione costruita in modo diverso dalle altre. Nella seconda si trovano, tra gli altri, due metodi principali: uno connette l'applicazione al database sorgente (quindi al server di Firebird 1.5 e relativo file di dati da trasferire), l'altro crea la connessione con il database scelto dall'utente per effettuare il trasferimento.

Entrambi i metodi invocano i costruttori dichiarati nel package *Database* creando degli oggetti di tipo *Connessione*. Le connessioni si creano caricando dapprima i driver JDBC che ognuno dei database utilizzati fornisce in dotazione, e successivamente passando al costruttore della classe *DriverManager* di Java l'url del pc (sorgente e destinatario), il nome utente e la password (sia del database di origine, sia di quello di destinazione). Nel caso di Oracle Database XE, il relativo codice Java della specifica classe è:

```
package ToolTravaso.Connessioni.Database;
import oracle.jdbc.OracleDriver;
import ToolTravaso.Connessioni.Connettore;

public class XE {
    // driver XE
    private static final String forName = "oracle.jdbc.OracleDriver";
    // stringhe di connessione
    private static final String driver = "jdbc:oracle:thin:";
    private static final String porta = "1521";
    private static final String istanza = "XE";

    // costruttore ad un argomento della connessione di Oracle XE
    public XE(String host) {
        Connettore.setFORNAME(forName);
        Connettore.setURL(driver + "@" + host + ":" + porta + ":" + istanza);
    }
}
```

Nella classe *Connettore*, ad esempio, il seguente metodo connette il database destinatario:

```
package ToolTravaso.Connessioni;
import ToolTravaso.Connessioni.Database.*;
import ToolTravaso.IO.Input;

public static Connection connettiDestinazioneDati() throws Exception {
    short tipoDB = Short.parseShort(Input.getInput(0));
    String host = Input.getInput(5);
    String fileDB = Input.getInput(6);
    String SqlServerDB = Input.getInput(7);
    // connessione di destinazione
    switch(tipoDB) {
        case 0: case default: /* eccezione */ break;
        case 1: /* richiamo del costruttore di FB X.x */ break;
        case 2: /* richiamo del costruttore di Oracle XE */ break;
        case 3: /* richiamo del costruttore di SQL Server Express */ break;
    }
    Class.forName(getFORNAME());
    return DriverManager.getConnection(getURL(), Input.getInput(8),
    Input.getInput(9));
}
```

3.3 Routines trasferimento dati (package Routines)

Questo pacchetto è suddiviso in quattro sotto-package aventi i nomi delle routines che il programma esegue durante il trasferimento (*Controlla*, *Create*, *Insert*, *Alter*, *Index*) che a loro volta contengono le classi con le definizioni di metodi specifici per ogni database, e una classe *Routines*.

3.3.1 Routine Controlla

Il package *Controlla* contiene una classe *TabellaCONTROLLA* con un costruttore che corregge eventuali errori nel file di dati originario causati da malfunzionamenti nelle operazioni di backup e restore effettuate in precedenza.

Gli errori più gravi che possono verificarsi e che devono assolutamente essere corretti si riscontrano nelle tabelle con attributi aventi la clausola NOT NULL o di unicità. Può accadere, infatti, che ad un campo di una colonna marcata NOT NULL durante le suddette operazioni di compattazione e ripristino possa essere

assegnato un valore nullo, nel caso in cui il tipo dell'attributo sia VARCHAR; oppure si può verificare che un attributo avente clausola di unicità abbia due o più campi cui è assegnato lo stesso identico valore o più volte il valore nullo. Fortunatamente la possibilità di riscontrare questo genere di errori è decisamente irrisoria, ma nel qual caso si presentassero tali situazioni si genererebbero delle anomalie in fase di popolamento del database finale. I campi, a seconda di quale dei due casi descritti si presenta, sono corretti aggiornandoli con opportuni valori di default. Viene riportato di seguito la sintassi del costruttore:

```
package ToolTravaso.Routines.Controlla;
import javax.sql.*;
import java.sql.*;
import java.util.*;
import ToolTravaso.Routines.Routines;
import ToolTravaso.IO.Output;

public class TabellaCONTROLLA {

    // costruttore della classe TabellaCONTROLLA
    public TabellaCONTROLLA(String tabellaCorrente, DatabaseMetaData dbMD, Statement
        stmt, Connection conn) {
        try {
            // elenco di tutte le colonne della tabella
            String querySELECT = "SELECT FIRST 1 SKIP 0 * FROM " + tabellaCorrente;
            ResultSet rsPrimaRiga = stmt.executeQuery(querySELECT);
            ResultSetMetaData rsPrimaRigaMD = rsPrimaRiga.getMetaData();
            int nCol = rsPrimaRigaMD.getColumnCount();
            // scorrimento delle colonne e controllo di quelle con clausola NOT NULL
            for(int i = 1; i <= nCol; i++) {
                // clausola NOT NULL (presenza/assenza)
                boolean notNull = false;
                int nullable = rsPrimaRigaMD.isNullable(i);
                switch(nullable) {
                    case java.sql.DatabaseMetaData.columnNoNulls: notNull = true; break;
                    default: break;
                }

                String tipoCol = rsPrimaRigaMD.getColumnTypeName(i);
                /*
                 se la colonna ha la clausola NOT NULL ed è del tipo VARCHAR o CHAR
                 imposto un valore di default in caso di inconsistenza
                */
                if(notNull && (tipoCol.equals("VARCHAR"))) {
                    /*
                     impostazione dei valori di default dei campi errati
                     e aggiornamento con query di UPDATE
                    */
                }
            }
        }
    }
}
```

```
        */
    }
}
}
catch(Exception ex) { /* gestione dell'eventuale eccezione */ }
}
}
```

3.3.2 Routine Create

Nel package *Create* si trovano quattro versioni della classe *TabellaCREATE*, ognuna adattata alla regole sintattiche del database destinatario che è stato selezionato dall'utente.

La funzionalità della classe è costruire la struttura di ogni tabella che si intende trasferire, quindi formare l'istruzione `CREATE TABLE` corrispondente, partendo dai metadati originari, ed eseguirla: Java mette a disposizione diversi metodi per estrarli da una struttura esistente. Prendendo la prima riga e scorrendola, per ogni attributo che la compone se ne ricavano nome, tipo (con eventuale dimensione) e clausole `NOT NULL`; infine, se esistono, si ricavano i vincoli di unicità e chiave primaria presenti nella tabella in esame. Tali informazioni sono quindi memorizzate in un buffer che, quando sarà completato con l'ultima colonna, andrà a formare il corpo dell'istruzione `CREATE TABLE`.

Infine, ottenute tutte le informazioni necessarie per costruire lo statement `CREATE TABLE`, esso viene eseguito, come tutte le altre istruzioni SQL eseguite dalle varie routines, con i metodi della classe *PreparedStatement*, di modo che l'istruzione venga compilata e caricata in memoria una sola volta, rendendo più efficienti le successive chiamate.

Segue una piccola parte del codice che espleta queste funzioni per quanto riguarda la sintassi di SQL Server Express.

```
package ToolTravaso.Routines.Create;
import javax.sql.*;
```

```

import java.sql.*;
import java.lang.reflect.Array;
import java.util.*;
import java.util.regex.*;
import ToolTravaso.Routines.Routines;
import ToolTravaso.IO.Output;

public class TabellaCREATE_SSE {
    // strutture della classe TabellaCREATE
    public TabellaCREATE_SSE(String tabellaCorrente, DatabaseMetaData dbMD, Statement
        stmt, Connection conn) {
        try {
            /*
                ...
            */
            StringBuffer corpoCREATE = new StringBuffer();
            queryCREATE = "CREATE TABLE " + tabellaCorrente + " ( ";
            // creazione del corpo della query
            for(int i = 1; i <= nCol; i++) {
                // nome colonna
                String nomeCol = rsPrimaRigaMD.getColumnName(i);
                // tipo colonna
                String tipoCol = setTipo(rsPrimaRigaMD.getColumnTypeName(i));
                // dimensione colonna
                String dimCol = new String();
                // caso VARCHAR
                if(tipoCol.equals("VARCHAR")) {
                    int dim = rsPrimaRigaMD.getColumnDisplaySize(i);
                    // porto la dimensione alla massima consentita da XE
                    if(dim > 8000) {
                        dim = 8000;
                    }
                    dimCol = "(" + dim + ")";
                }
                // clausola NOT NULL (presenza/assenza)
                String clausolaNullCol = new String();
                int nullable = rsPrimaRigaMD.isNullable(i);
                switch(nullable) {
                    case java.sql.DatabaseMetaData.columnNoNulls:
                        clausolaNullCol = "NOT NULL";
                        break;
                    default: break;
                }
            }
            /*
                ...
            */
            queryCREATE += corpoCREATE.toString();
            // popolo la lista dei campi UNIQUE
            List listaUnique = new ArrayList();
            ResultSet rsIndex = dbMD.getIndexInfo("", "", tabellaCorrente, true, false);
            while(rsIndex.next()) { /* riempimento lista campi UNIQUE */ }
        }
    }
}

```

```

        la lista dei campi UNIQUE diventa una stringa che si aggiunge a queryCREATE
    */
    List listaPK = new ArrayList();
    ResultSet rsPK = dbMD.getPrimaryKeys("", "", tabellaCorrente);
    while(rsPK.next()) { /* riempimento lista campi PRIMARY KEY */ }
    /*
        come la precedente, la lista si trasforma in stringa
        e si aggiunge a queryCREATE
    */
    // si esegue l'istruzione CREATE TABLE completa
    PreparedStatement ps = conn.prepareStatement(queryCREATE);
    ps.executeUpdate();
    // chiusura del PreparedStatement per il rilascio di memoria

    ps.close();
}
catch(Exception ex) { /* gestione dell'eventuale eccezione */ }
}
/*
    definizioni di altri metodi della classe TabellaCREATE
*/
}

```

3.3.2.1 Tipi degli attributi

Per quanto riguarda il tipo dell'attributo, è significativo dire che avviene una conversione dai tipi di Firebird 1.5 ai tipi previsti per Oracle XE e SQL Server Express, qualora uno di questi due sia stato selezionato come finale per la copia. Non avviene nessuna conversione se la copia è effettuata su un'altra versione di Firebird.

La seguente tabella riassume le varie conversioni:

Conversioni dei tipi di Firebird 1.5 ai tipi di Oracle XE e SQL Server Express					
Firebird 1.5	VARCHAR	DATE	INTEGER	SMALLINT	DOUBLE PRECISION
Oracle XE	VARCHAR2	DATE	NUMBER(38)	BINARY FLOAT	BINARY DOUBLE
SQL Server Express	VARCHAR	DATETIME	INT	SMALLINT	FLOAT

Tab. I

Si noti che il numero di tipi presi in considerazione è un sottoinsieme alquanto ridotto di quelli effettivamente resi disponibili da ognuno dei database: il tool di trasferimento è, infatti, ottimizzato unicamente sui tipi usati nell'applicativo EASY e nei database dei clienti. La routine *Create*, inoltre, controlla che per il tipo *VARCHAR2* (Oracle XE) la dimensione massima (cioè il numero di caratteri) sia 4000, mentre per *VARCHAR* (SQL Server) sia 8000, poiché queste sono le limitazioni imposte da tali architetture. Le caratteristiche e le differenze tra i vari tipi degli attributi saranno meglio approfondite e analizzate nel capitolo successivo.

Sempre focalizzando sul caso SQL Server Express, la funzione che effettua la conversione di tipo è la seguente, ma il funzionamento nel caso di Oracle Database XE è concettualmente identico:

```
private static String setTipo(String tipo) {  
    /*  
        assegna il tipo adatto alla colonna secondo questo schema:  
        Firebird:  VARCHAR      DATE          INTEGER      SMALLINT      DOUBLE PRECISION  
  
        SqlSeExp:  VARCHAR      DATETIME      INT           SMALLINT      FLOAT  
  
    */  
    String tipoMod = new String();  
  
    if(tipo.equals("DATE")) {  
        tipoMod = "DATETIME";  
    }  
    else {  
        if(tipo.equals("INTEGER")) {  
            tipoMod = "INT";  
        }  
        else {  
            if(tipo.equals("DOUBLE PRECISION")) {  
                tipoMod = "FLOAT";  
            }  
            else {  
                tipoMod = tipo;  
            }  
        }  
    }  
    return tipoMod;  
}
```

3.3.3 Routine Insert

A differenza del precedente, nel package *Insert* si trova la sola classe *TabellaINSERT*, poiché la sintassi dell'istruzione di inserimento è uguale per tutti i database candidati.

Il primo passo necessario a questa routine è riconoscere se la chiave primaria della tabella che si sta per popolare con la copia dei dati originari è *standard* oppure no. Per convenzione, una tabella ha chiave primaria standard se ad essa è associato un unico campo chiave primaria, mentre non lo è se i campi assegnati a questo vincolo sono più di uno (chiave primaria multicampo) oppure totalmente assenti. Questa operazione di riconoscimento è resa necessaria dal fatto che le righe di una tabella non sono estratte blocco con un'unica query, perché così facendo, vista la mole di molte tabelle (alcune con oltre un milione di record), il programma lancia l'eccezione *out-of-memory* a causa dei numerosi cursori aperti per l'inserimento. L'idea è quindi di suddividere ed estrarre i dati a lotti prefissati di $n = 500$ righe: supponendo che la tabella contenga N righe, i dati in essa contenuti saranno trasferiti compiendo $i + 1$ iterazioni (letture di n righe) sulla stessa, e $i = N / n$. Ovviamente si considera la parte intera della divisione N / n . La parte iniziale del codice di questa routine è:

```
package ToolTravaso.Routines.Insert;
import javax.sql.*;
import java.sql.*;
import java.lang.reflect.Array;
import java.util.*;
import ToolTravaso.Routines.Routines;
import ToolTravaso.IO.Output;
import ToolTravaso.Connessioni.Connettore;

public class TabellaINSERT {

    private static final long lotto = 500;
    /*
     * ...
     */
    // costruttore della classe TabellaINSERT
    public TabellaINSERT(String tabellaCorrente, DatabaseMetaData dbMD, Statement
```

```

stmt, Connection conn) {

try {
    ResultSet rsPK = dbMD.getPrimaryKeys("", "", tabellaCorrente);
    String nomePK = new String();
    int nCampiPK = 0;
    while(rsPK.next()) {
        nCampiPK++;
        // caso in cui esiste un unico campo PRIMARY KEY (standard)
        nomePK = rsPK.getString("COLUMN_NAME");
    }
    // chiave primaria standard: caso di default
    boolean isStandard = true;
    // chiave primaria non standard: più o inesistenti campi PRIMARY KEY
    if(nCampiPK > 1 || nomePK.length() == 0) {
        isStandard = false;
    }
    /*
        ...
    */
}
catch(Exception ex) { /* gestione dell'eventuale eccezione */ }
}
/*
    definizioni di altri metodi di TabellaINSERT
*/
}

```

3.3.3.1 Tabelle con chiave primaria standard

Se il programma processa una tabella con chiave primaria standard, N è uguale al valore intero più grande del campo identificativo ($id<nometabella>$) di quella tabella: infatti, nei database clienti di EASY, se una tabella ha un campo chiave primaria standard, questo è di tipo intero progressivo. La query che estrae n righe all'($i+1$)-esima iterazione è del tipo

“SELECT * FROM T WHERE ID BETWEEN A AND B ”

dove T = tabella corrente, ID = campo della chiave primaria, $A = i \times n$, e $B = (i + 1) \times n - 1$.

A titolo esemplificativo, si supponga che la tabella *ANAGRAFICHE* abbia un unico attributo chiave primaria *IDANAGRAFICHE* e contenga $N = 3200$ righe,

quindi $i = \text{parte intera } (3200 / 500) = 6$ e dunque le iterazioni saranno 7. Se, ad esempio, la routine che estrae le righe originarie si trova alla quarta iterazione (perciò $i = \text{numero iterazione} - 1 = 3$), il database esegue la query "SELECT * FROM ANAGRAFICHE WHERE IDANAGRAFICHE BETWEEN 1500 AND 1999".

All'iterazione successiva, l'istruzione da eseguire sarà "SELECT * FROM ANAGRAFICHE WHERE IDANAGRAFICHE BETWEEN 2000 AND 2499": si ricorda che con il costrutto BETWEEN A AND B, gli estremi A e B sono compresi nell'intervallo. Segue la parte di codice specifico di questo caso.

```
/*
...
*/
// se si tratta di una tabella con PRIMARY KEY standard si estrae MAX(PRIMARY KEY)
long maxID = 0;
if(isStandard) {
    String queryMAX = "SELECT MAX(" + nomePK + ") FROM " + tabellaCorrente;
    ResultSet rsMax = stmt.executeQuery(queryMAX);
    while(rsMax.next()) {
        // estrazione di MAX(PRIMARY KEY)
        maxID = rsMax.getLong(1);
    }
}
/*
...
*/
// le iterazioni nel caso standard sono in funzione del MAX(PRIMARY KEY)
if(isStandard) {
    iter = maxID/lotto;
}
else { /* caso non standard */ }

for(int i = 0; i <= iter; i++) {
    String querySELECT = new String();
    // caso standard: query con BETWEEN ... AND ...
    if(isStandard) {
        querySELECT = "SELECT * FROM " + tabellaCorrente + " WHERE " + nomePK + "
            BETWEEN " + (i*lotto) + " AND " + ((i+1)*lotto-1);
    }
    else { /* caso non standard */ }
}
/*
...
*/
```

3.3.3.2 Tabelle con chiave primaria non standard

Altresì, se la tabella non è standard, N corrisponde al numero totale di righe della tabella T ed è fondamentale utilizzare la query

“SELECT FIRST A SKIP B * FROM T ”

con $A = n$ e $B = i \times n$.

Questo particolare costruito è peculiare della sintassi SQL di Firebird, e ritorna come risultato le prime A righe dopo aver escluso le B iniziali.

Ad esempio, si supponga che il programma stia trasferendo la tabella *ARTICOLIRICARICO* e questa non possieda nessun campo chiave primaria (lo stesso varrebbe se esistessero due o più attributi con vincolo chiave primaria). Supponiamo che in essa siano presenti $N = 2700$ record, quindi $i =$ parte intera $(2700 / 500) = 5$ e le iterazioni saranno 6. Soffermendosi sulla quinta iterazione, la query da processare sarà “SELECT FIRST 500 SKIP 2000 * FROM *ARTICOLIRICARICO*” ed in particolare $i = 5 - 1 = 4$: questa istruzione selezionerà 500 righe dopo aver ignorato le prime 2000.

Analizzando la successiva iterazione, $i = 5$ e la sintassi dell’istruzione sarà “SELECT FIRST 500 SKIP 2500 * FROM *ARTICOLIRICARICO*”: risultato di questa query saranno le 200 righe rimanenti dopo aver saltato le prime 2500.

Come nel paragrafo precedente, di seguito si riporta la parte di codice dedicata a questo caso:

```
/*
...
*/
// se non si tratta di PRIMARY KEY standard, si contano le righe della tabella
long count = 0;
String queryCount = "SELECT COUNT(*) FROM " + tabellaCorrente;
ResultSet rsCount = stmt.executeQuery(queryCount);
while(rsCount.next()) {
    // numero totale di righe della tabella
    count = rsCount.getLong(1);
}
/*
...
*/
```

```
*/
if(isStandard) { /* caso standard */ }
// le iterazioni in questo caso sono in funzione del numero totale di righe
else { iter = count/lotto; }

for(int i = 0; i <= iter; i++) {
    String querySELECT = new String();
    if(isStandard) { /* caso standard */ }
    //caso non standard: query con FIRST ... SKIP ...
    else {
        querySELECT = "SELECT FIRST " + Lotto + " SKIP " + (i*Lotto) + " * FROM " +
                        tabellaCorrente;
    }
}
}
/*
...
*/
```

3.3.3.3 Inserimenti con tecnica mista

É necessario utilizzare una tecnica che mista tra quella per l'estrazione dei dati da copiare da tabelle con chiave primaria standard e quelle con chiave primaria non standard. Non si adotta solamente la seconda tecnica per due motivi: il primo è che si riscontra un cospicuo rallentamento nella routine d'inserimento (circa il 30% in più nelle tabelle con oltre 200.000 record rispetto al costrutto BETWEEN A AND B), il secondo è per il fatto che il numero di tabelle con chiave primaria non standard è esiguo nei database di EASY.

3.3.3.4 Gestione valori nulli

Una volta estratto un lotto di righe dalla tabella *T*, la routine procede scorrendo una ad una tutte le righe, e completando per ognuna la query "INSERT INTO *T* VALUES (?, ?, ... , ?)" con i valori dei singoli campi al posto dei parametri ?.

Esiste un metodo, *impostaValore*, che controlla e imposta ogni valore da copiare: dove previsto dai vincoli, se questo è nullo, viene gestito direttamente dal driver JDBC che, a seconda del tipo dell'attributo a cui esso è riferito, assegna il

corrispondente valore nullo con codifica che varia da RDBMS a RDBMS; se il valore non è invece nullo, il metodo si limita ad assegnare al parametro ? la copia del valore originario con la codifica prevista dal database di destinazione.

Si riporta di seguito il costrutto principale di *impostaValore*:

```
public static void impostaValore(PreparedStatement ps, ResultSet rs,
    ResultSetMetaData md, int col) throws Exception {

    // si trova il tipo originario della colonna
    int tipo = md.getColumnType(col);

    switch(tipo) {
        // VARCHAR
        case java.sql.Types.VARCHAR:
            String s = rs.getString(col);
            // se il valore è nullo viene gestito dal driver
            if(rs.wasNull()) {
                ps.setNull(col, java.sql.Types.VARCHAR);
            }
            // se il valore non è nullo viene inserito nel campo di destinazione
            else {
                ps.setString(col, s);
            }
            break;
        /*
            lo stesse operazioni sono effettuate per tutti gli altri tipi: DATE, INTEGER,
            SMALLINT, DOUBLE PRECISION, FLOAT e NUMERIC
        */
    }
}
```

Completata la sintassi dell'istruzione di INSERT, questa viene eseguita dal database di destinazione. Alla fine di ogni lotto copiato viene eseguito il commit delle operazioni di inserimento tranne se il database finale è SQL Server Express, poiché in questo caso il commit è effettuato automaticamente.

3.3.4 Routine Alter

Il package *Alter* contiene le classi *TabellaALTER*, ognuna per ogni possibile database di destinazione, che eseguono le istruzioni ALTER TABLE necessarie

ad aggiungere i vincoli di chiave esterna alle tabelle, una volta che tutte queste sono state create e interamente popolate.

Grazie ai metodi forniti da Java, si riescono ad ottenere i nomi della chiave esterna e del relativo campo, della tabella e del campo chiave primaria ad essa associate, ed inoltre definire le regole di UPDATE e DELETE partendo da quelle originarie. Anche in questo caso, una volta effettuate, tutte le modifiche sono rese permanenti, eccetto per SQL Server Express, con lo statement commit.

Di seguito si riporta parte del codice della classe relativa a Firebird 1.0 e 2.0.

```
package ToolTravaso.Routines.Alter;
import javax.sql.*;
import java.sql.*;
import java.lang.reflect.Array;
import java.util.*;
import ToolTravaso.Routines.Routines;
import ToolTravaso.IO.Output;

public class TabellaALTER_FBXX {

    public TabellaALTER_FBXX(String tabellaCorrente, DatabaseMetaData dbMD,
        Connection conn) {

        try {
            ResultSet rsImpKeys = dbMD.getImportedKeys("", "", tabellaCorrente);
            // si ricavano i valori necessari per creare le FOREIGN KEY
            while(rsImpKeys.next()) {
                String nomeFK = rsImpKeys.getString("FK_NAME");
                String campoFK = rsImpKeys.getString("FKCOLUMN_NAME");
                String tabellaPK = rsImpKeys.getString("PKTABLE_NAME");
                String campoPK = rsImpKeys.getString("PKCOLUMN_NAME");
                String onDeleteRule = new String(), onUpdateRule = new String();
                short onDelete = rsImpKeys.getShort("DELETE_RULE");
                switch(onDelete) {
                    case java.sql.DatabaseMetaData.importedKeyCascade:
                        onDeleteRule = "ON DELETE CASCADE";
                        break;
                    case java.sql.DatabaseMetaData.importedKeyNoAction:
                        onDeleteRule = "ON DELETE NO ACTION";
                        break;
                    default: break;
                }
                short onUpdate = rsImpKeys.getShort("UPDATE_RULE");
                switch(onUpdate) {
                    case java.sql.DatabaseMetaData.importedKeyCascade:
                        onUpdateRule = "ON UPDATE CASCADE";
                        break;
                    case java.sql.DatabaseMetaData.importedKeyNoAction:
```

```
        onUpdateRule = "ON UPDATE NO ACTION";
    break;
    default: break;
}
/*
    il ciclo si conclude quando, per ogni tabella, sono state create tutte le
    FOREIGN KEY
*/
}
}
catch(Exception ex){ /* gestione dell'eventuale eccezione */ }
}
}
```

3.3.5 Routine Index

Quest'ultimo package, *Index*, contiene le tre classi *TabellaINDEX* che, richiamate a seconda del DBMS finale, aggiungono eventuali indici definiti dall'utente (in questo caso dai progettisti del database di EASY), al fine di velocizzare la ricerca di determinati record.

Una volta ottenuti i nomi dei campi indicizzati e degli indici stessi, la presenza o meno del vincolo di unicità e dell'eventuale ordinamento decrescente, la routine prosegue scartando gli indici di sistema, che Firebird crea sulle chiavi primarie ed esterne di ogni tabella, e i vincoli di unicità già aggiunti con la routine *Create*. A questo punto, tutte le informazioni necessarie per completare l'istruzione *CREATE INDEX* sono state estratte dai metadati del database sorgente, per cui la query viene eseguita e gli indici aggiunti. Si noti che, nel caso la copia sia effettuata su Oracle Database XE, non sono aggiunti gli indici sugli attributi chiave primaria, poiché lo stesso database non lo permette. Il codice essenziale di questa classe in particolare è il seguente:

```
package ToolTravaso.Routines.Index;
import javax.sql.*;
import java.sql.*;
import java.lang.reflect.Array;
import java.util.*;
import java.util.regex.*;
import ToolTravaso.Routines.Routines;
import ToolTravaso.IO.Output;
```

```
public class TabellaINDEX_XE {

    private static String queryINDEX = new String();

    public TabellaINDEX_XE(String tabellaCorrente, DatabaseMetaData dbMD, Connection
    conn) {

        try {
            /*
             * ...
             */
            ResultSet rsIndex = dbMD.getIndexInfo("", "", tabellaCorrente, false, false);
            List listaIDX = new ArrayList();
            /*
             * si scorre l'insieme di tutti gli indici del database, ricavandone le
             * informazioni necessarie
             */
            while(rsIndex.next()) {
                String nomeIDX = rsIndex.getString("INDEX_NAME");
                String campoIDX = rsIndex.getString("COLUMN_NAME");
                boolean nonUnique = rsIndex.getBoolean("NON_UNIQUE");
                /*
                 * solamente gli indici user-defined sono copiati nella lista, nel caso
                 * particolare di Oracle XE vengono però anche esclusi quelli creati su
                 * campi PRIMARY KEY
                 */
            }
            /*
             * in seguito si svolgono tutte le operazioni necessarie per creare ed
             * eseguire le istruzioni di CREATE INDEX
             */
        }
        catch(Exception e) { /* gestione dell'eventuale eccezione */ }
    }
    /*
     * definizioni di altri metodi della classe TabellaINDEX
     */
}
```

3.3.6 Classe Routines

Contiene i metodi utilizzati nel corpo dell'applicazione: essi richiamano e inizializzano le precedenti routines per il trasferimento dei dati, a seconda del database di destinazione selezionato dall'utente. Segue un'esemplificazione del codice:

```
package ToolTravaso.Routines;
import javax.sql.*;
import java.sql.*;
import java.lang.reflect.Array;
import java.util.*;
import ToolTravaso.IO.Output;
import ToolTravaso.Connessioni.Connettore;
import ToolTravaso.Routines.Controlla.*;
import ToolTravaso.Routines.Create.*;
import ToolTravaso.Routines.Insert.*;
import ToolTravaso.Routines.Alter.*;
import ToolTravaso.Routines.Index.*;

public class Routines {

    // costruttore esplicito della classe Routines
    public Routines() {}

    public static void controlla(String tabella, DatabaseMetaData dbMD, Statement
        stmt, Connection conn) {

        new TabellaCONTROLLA(tabella, dbMD, stmt, conn);
    }

    public static void create(String tabella, DatabaseMetaData dbMD, Statement stmt,
        Connection conn) {

        switch(Connettore.getENUMDB()) {
            case 1: new TabellaCREATE_FBXX(tabella, dbMD, stmt, conn); break;
            case 2: new TabellaCREATE_XE(tabella, dbMD, stmt, conn); break;
            case 3: new TabellaCREATE_SSE(tabella, dbMD, stmt, conn); break;
            default: break;
        }
    }
    /*
        in seguito, ognuna delle routine TabellaINSERT, TabellaALTER e TabellaINDEX è
        inizializzata come la precedente
    */
}
```

3.4 Corpo dell'applicazione (package Main)

Il corpo dell'applicazione contiene un'unica classe *Main* che a sua volta contiene il metodo principale del programma, il *main*. In esso vengono eseguite, nell'ordine, le seguenti operazioni:

- Vengono create la connessione di origine e di destinazione.

- Si stila l'elenco delle tabelle da copiare: vengono scartate le tabelle temporanee create con l'utilizzo di EASY e quelle di sistema create da Firebird.
- In un primo ciclo, tabella per tabella, vengono eseguite le routines *Controlla*, *Create* e *Insert*: il database finale è così interamente creato e popolato.
- In un secondo ciclo, ad ogni tabella sono aggiunti i vincoli di chiave esterna con la routine *Alter*.
- In un terzo ciclo, del tutto simile al precedente, sono creati gli indici definiti dall'utente eseguendo la routine *Index*.
- Le connessioni ai database vengono chiuse.
- Se richiesto, è creato il file con il riepilogo delle statistiche del trasferimento.

In altre parole, la classe *Main* attiva e disattiva le connessioni ai database sorgente e destinatario, e inoltre coordina le routines di trasferimento dati sulle tabelle e relativi dati da copiare. Viene riportato, in seguito, il codice essenziale di questa classe:

```
package ToolTravaso.Main;
import javax.sql.*;
import java.sql.*;
import java.lang.reflect.Array;
import java.util.*;
import java.util.regex.*;
import java.io.*;
import ToolTravaso.IO.*;
import ToolTravaso.Connessioni.Connettore;
import ToolTravaso.Routines.Routines;
import ToolTravaso.Routines.Controlla.*;
import ToolTravaso.Routines.Create.*;
import ToolTravaso.Routines.Insert.*;
import ToolTravaso.Routines.Alter.*;
import ToolTravaso.Routines.Index.*;

public class Main {

    // costruttore esplicito della classe Main
```

```
public Main() {}

// metodo principale main
public static void main(String args[]) {

    try {
        Connettore conn = new Connettore();
        // connessione e statement di origine
        Connection connSORG = conn.connettiOrigineDati();
        Statement stmtSORG = connSORG.createStatement();
        // connessione di destinazione
        Connection connDEST = conn.connettiDestinazioneDati();
        // lista tabelle del database originario, Firebird 1.5
        DatabaseMetaData dbMDSORG = connSORG.getMetaData();
        ResultSet rsNomiTabelleSORG = dbMDSORG.getTables("", "", "", null);
        // lista dei nomi delle tabelle da copiare (escluse le tabelle di sistema)
        List listaTabelleFinali = new ArrayList();
        while(rsNomiTabelleSORG.next()) {
            String tabella = rsNomiTabelleSORG.getString("TABLE_NAME");
            // controllo che la tabella sia da travasare e l'aggiungo alla lista
            if(daTravasare(tabella)) {
                listaTabelleFinali.add(tabella);
            }
        }
        /*
        ...
        */
        for(int i = 0; i < nTabelle; i++) {
            String tabella = arrayTabelleFinali[i];
            // Controllo della tabella[i]
            Routines.controlla(tabella, dbMDSORG, stmtSORG, connSORG);
            // CREATE della tabella[i]
            Routines.create(tabella, dbMDSORG, stmtSORG, connDEST);
            // INSERT della tabella[i]
            Routines.insert(tabella, dbMDSORG, stmtSORG, connDEST);
        }

        for(int i = 0; i < nTabelle; i++) {
            String tabella = arrayTabelleFinali[i];
            // ALTER della tabella[i]
            Routines.alter(tabella, dbMDSORG, connDEST);
        }

        for(int i = 0; i < nTabelle; i++) {
            String tabella = arrayTabelleFinali[i];
            // INDEX della tabella[i]
            Routines.index(tabella, dbMDSORG, connDEST);
        }
        connSORG.close();
        connDEST.close();
        /*
        ...
        */
    }
}
```

```
    }
    catch (Exception ex) { /* gestione dell'eventuale eccezione */ }
}
/*
    questo metodo per mezzo delle espressioni regolari (regex) individua le tabelle
    temporanee e di sistema
*/
private static boolean daTravasare(String tabella) {

    boolean isTabellaTemporanea = false, isTabellaDiSistema = false;
    boolean travasa = false;
    // tabelle temporanee di EASY: DD_XXXXXX || DXXXXXX dove D è un numero da 0 a 9
    Pattern pT1 = Pattern.compile("^([0-9]+)(\\p{Alpha}+)$");
    Pattern pT2 = Pattern.compile("^([0-9]+)(\\p{Alpha}+)$");
    // tabelle di sistema di Firebird 1.5: RDB$XXXXXX || IBE$XXXXXX
    Pattern pS1 = Pattern.compile("^RDB\\$(.+$)");
    Pattern pS2 = Pattern.compile("^IBE\\$(.+$)");
    /*
        si effettuano i controlli sul nome della tabella
    */
    if(!isTabellaTemporanea && !isTabellaDiSistema) {
        travasa = true;
    }
    return travasa;
}
}
```


4. Implementazione in EASY delle architetture degli RDBMS candidati

Dopo aver creato un'applicazione che renda possibile il trasferimento del database originario da Firebird ai diversi RDBMS candidati a sostituirlo, è necessario implementare uno "strato" software che permetta l'utilizzo del gestionale con tutti i database in esame, incluso l'attuale.

Le modifiche da apportare al codice del gestionale sono molte, ma in questo capitolo saranno analizzate solamente le principali.

4.1 Classe SysDatiDitteBean

La classe *SysDatiDitteBean* estrae da una tabella, *DITTE*, contenuta nel database *syseasydb.gdb* situato sul server aziendale, le informazioni essenziali inerenti al database cui l'utente vuole connettersi, cioè:

- stringa di connessione al database finale
- nome utente
- password
- il *JDBC driver path*, cioè il nome del driver JDBC.

Per distinguere i vari RDBMS, è necessario aggiungere un attributo *tipoDB* di tipo SMALLINT alla tabella *DITTE*, che assumerà i seguenti valori secondo il database selezionato:

- 0, per Firebird 1.5
- 1, per Firebird 1.0 / 2.0
- 2, per Oracle Database XE

- 3, per SQL Server Express

Quindi, questo nuovo campo deve essere compreso nel risultato della query descritta in precedenza; la sintassi del comando che estrae tutte queste informazioni è:

```
/*
    ...
*/
// variabile che contiene le informazioni di connessione
private static String[] param = null;
try {
    risultato = dichiarazione.executeQuery("SELECT CONNECTIONSTRING, UTENTE,
PASSWORD,
    JDBCDRIVERPATH, TIPODB FROM DITTE WHERE IDDITTA = '" + idDitta + "'");
    param = new String[5];
    while (risultato.next()){
        param[0] = risultato.getString(1);
        param[1] = risultato.getString(2);
        param[2] = risultato.getString(3);
        param[3] = risultato.getString(4);
        param[4] = risultato.getString(5);
    }
}
catch(Exception ex) { /* gestione dell'eventuale eccezione */ }
/*
    metodo getData che restituisce param
*/
/*
    ...
*/
```

Come per tutte le classi che terminano con la parola *Bean*, *SysDatiDitteBean* infine viene compilato con l'interfaccia Java *SysDatiDitteRemote* e la classe *SysDatiDitteHome*, che contiene le informazioni per interfacciarsi con l'application server, JBoss, descritto nel primo capitolo. Viene così creato il relativo EJB, che altro non è che il file *<nome della classe>.jar*, situato nella cartella *deploy* di JBoss.

4.2 Classe ConnectionManager

ConnectionManager è la classe che gestisce e sincronizza le connessioni al database, anche in modo concorrente, da parte dei vari utenti con le chiamate alla JDBC API, creando delle sessioni per garantire la sicurezza delle operazioni compiute da un certo operatore in quel momento.

Per rendere questa classe compatibile all'utilizzo degli altri RDBMS oltre a Firebird 1.5, si apportano le seguenti modifiche:

- Aggiunta delle variabili relative ai tipo di RDBMS utilizzato, richiamato dalla classe *SysDatiDitteBean*.

```
private static short tipoDB;
private static final short Firebird15 = 0;
private static final short FirebirdXX = 1;
private static final short OracleXE = 2;
private static final short SqlServerExpress = 3;
// metodo che restituisce l'RDBMS utilizzato
public static short getTipoDB() {
    return tipoDB;
}
/*
    ...
*/
```

- Nel metodo *createConnection*, si visualizza e imposta il tipo di RDBMS che è stato selezionato.

```
/*
    ...
*/
// impostazione della variabile tipoDB
tipoDB = Short.parseShort(ditta.getTipoDB());
switch(tipoDB) {
    case Firebird15:
        System.out.println(" ---- database corrente: Firebird 1.5");
        break;
    case FirebirdXX:
        System.out.println(" ---- database corrente: Firebird 1.0 / 2.0");
        break;
    case OracleXE:
        System.out.println(" ---- database corrente: Oracle XE");
        break;
    case SqlServerExpress:
        System.out.println(" ---- database corrente: Sql Server Express");
```

```
        break;  
    }  
    /*  
        ...  
    */
```

4.3 Differenze tra i tipi degli attributi dei vari RDBMS utilizzati

Uno dei problemi principali nell'implementazione delle diverse architetture dei motori di database candidati al funzionamento in EASY è la differenza sintattica e strutturale dei tipi originari degli attributi utilizzati dai progettisti del database, già menzionati nel primo capitolo. I tipi si possono facilmente suddividere in tre aree: i tipi della famiglia Firebird, i tipi di Oracle Database XE e quelli SQL Server Express. Una differenza fondamentale di quest'ultimo dagli altri, è che le parole chiave dei tipi degli attributi sono salvate, nei metadati, in minuscolo, anche se al momento della dichiarazione il parser le accetta in entrambi i modi; diversamente, negli altri due, i tipi sono salvati in caratteri maiuscoli. Per convenzione e facilità di lettura, tutte le parole chiave dei tipi degli attributi saranno indicate, in seguito, sempre in caratteri maiuscoli.

4.3.1 Tipo stringa di caratteri a lunghezza variabile

È fondamentalmente uguale per tutti e tre i gruppi, anche se cambiano le dimensioni massime consentite in fase di salvataggio.

Sia per Firebird sia per SQL Server Express, la parola chiave di questo tipo è VARCHAR (Variable-length Character). Per entrambi può essere specificata una dimensione massima di caratteri con la sintassi VARCHAR(n), e n non può superare gli 8.000 caratteri.

Per Oracle Database XE, la parola chiave è VARCHAR2, ma la dimensione massima è 4.000 caratteri. In EASY, comunque, le dimensioni massime che sono raggiunte per un campo di questo tipo sono sull'ordine del migliaio di caratteri, quindi tutti gli RDBMS possiedono, per questo tipo di dati, i requisiti necessari.

Non sono stati usati campi di tipo BLOB (Binary Large Objects), tipo comune a tutti e tre i gruppi sintattici, per due motivi: il primo perché questo tipo di dati è utilizzato quando non si conoscono le dimensioni finali del testo da immettere, mentre per EASY queste sono state stimate; il secondo perché le dimensioni fisiche di queste strutture crescono notevolmente e con esse i tempi di accesso alle informazioni contenute.

4.3.2 Tipo data

È il tipo in assoluto più difficile da gestire, poiché ognuno degli RDBMS studiati, per esso, prevede diversi formati predefiniti; l'unico aspetto comune è che il valore della data è trattato come una stringa di caratteri nella sintassi SQL di qualsiasi istruzione, ma salvato come intero a 32 bit all'interno del database.

Il tipo DATE di Firebird è dichiarato nel *Dialect 3*, la sintassi SQL più recente, comune alle versioni 1.0, 1.5 e 2.0. Il formato della data salvata nel database è del tipo '*mm-gg-aaaa*', cioè il formato anglosassone: ad esempio, la data 24 novembre 2006, nel formato di Firebird è scritta come '*11-24-2006*'. Il gestionale si interfaccia con questo tipo seguendo la sua specifica sintassi.

Oracle Database XE utilizza la stessa parola chiave DATE per questo tipo di dati, ma il formato di default salvato nel database è del tipo '*gg-~~MMM~~-aaaa*': la precedente data è scritta come '*24-NOV-2006*'.

SQL Server Express, a sua volta, possiede un altro formato, '*aaaammgg*', e per portare l'esempio già noto, la data è scritta come '*20061124*'. Cambia anche la parola chiave, DATETIME.

4.3.3 Tipo intero

In Firebird e SQL Server Express, si suddividono in due tipi: SMALLINT, per i dati di tipo intero piccolo, e INTEGER, per il tipo intero. Più precisamente, in SQL Server è accettata la parola chiave INTEGER, che il parser provvede a salvare, nei metadati, come INT. In entrambi gli RDBMS, SMALLINT occupa 2 byte (con valori da -32.768 a +32.767) mentre INTEGER (INT) occupa 4 byte (con valori ammessi da -2.147.483.648 a +2.147.483.647).

In Oracle Database XE non esistono gli interi segnati, perciò si adotta la seguente convenzione per poter distinguere queste due tipologie di interi:

- Per il tipo intero piccolo, si utilizza il tipo floating-point (virgola mobile) a precisione singola (32 bit), BINARY_FLOAT, che occupa 5 byte (con un range di valori da 1,17549E-38F a 3,40282E+38F).
- Per il tipo intero, si utilizza il tipo numerico con parola chiave NUMBER(p), che identifica un valore numerico con precisione p , il cui valore di default è 38. La dimensione del tipo di dati NUMBER(p) varia da 1 a 22 byte.

4.3.4 Tipo decimale

Questo tipo è trattato, in Firebird, con la sintassi del *Dialect 1*, con la parola chiave DOUBLE PRECISION. Questo tipo è un decimale con mantissa che varia da 10 a 18 cifre, può occupare fino a 64 bit.

Anche per SQL Server Express lo spazio occupato arriva fino a 64 bit, ma cambiano sia la parola chiave, FLOAT, e la precisione, che scende a 15 cifre decimali.

Oracle Database XE utilizza una terza parola chiave per identificare questo tipo di dati, `BINARY_DOUBLE`: occupa sempre 64 bit ma la precisione della mantissa può arrivare fino a 127 cifre decimali.

4.4 Classe `DataConversion`

I metodi contenuti in questa classe, in un primo tempo, servivano ad effettuare la conversione dal formato della data con costrutto italiano, usato abitualmente (il classico *'gg-mm-aaaa'*, con l'esempio precedente: *'24-11-2006'*), a quello anglosassone di Firebird. Ora, oltre a Firebird, permette la conversione delle date anche nei formati di Oracle Database XE e SQL Server Express. Il metodo che permette questa operazione è il seguente:

```
public static String conversioneMeseGiorno(String data) {

    short tipoDB = Short.parseShort(Main.ottieniDitta().getTipoDB());
    // in questo momento, data è espresso nel formato gg-mm-aaaa
    if (!data.equals("")) {
        switch(tipoDB) {
            case Firebird15: case FirebirdXX:
                // conversione a mm-gg-aaaa
                data = data.substring(3, 6) + data.substring(0, 3) + data.substring(6, 10);
                break;
            case OracleXE:
                // conversione a gg-MMM-aaaa
                data = data.substring(0, 3) + getMeseXE(data.substring(3, 5)) +
                    data.substring(5, 10);
                break;
            case SqlServerExpress:
                // conversione a aaaammgg
                data = data.substring(6, 10) + data.substring(3, 5) + data.substring(0, 2);
                break;
            default: break;
        }
    }
    else { data = ""; }
    return data;
}
```

Un altro metodo interessante della classe `DataConversion` è `getMeseXE`, che ritorna la stringa col corrispondente nome di tre lettere del mese:

```
public static String getMeseXE(String mese) {

    String mXE = new String();
    short m = Short.parseShort(mese);
    switch(m) {
        case 1: mXE = "GEN"; break; case 2: mXE = "FEB"; break;
        case 3: mXE = "MAR"; break; case 4: mXE = "APR"; break;
        case 5: mXE = "MAG"; break; case 6: mXE = "GIU"; break;
        case 7: mXE = "LUG"; break; case 8: mXE = "AGO"; break;
        case 9: mXE = "SET"; break; case 10: mXE = "OTT"; break;
        case 11: mXE = "NOV"; break; case 12: mXE = "DIC"; break;
        default: break;
    }
    return mXE;
}
```

4.5 Classe Bridge

È una delle classi principali del programma, poiché gestisce qualsiasi operazione di riconoscimento dei tipi e delle strutture di dati da estrarre o salvare nel database finale. Le modifiche che si rendono necessarie, oltre a permettere la corretta esecuzione di queste operazioni sui nuovi RDBMS utilizzati, garantiscono alla classe *Bridge* anche alcuni metodi aggiuntivi che integrano i particolari costrutti SQL di ciascun database engine.

Preliminarmente sono state importate le classi *ConnectionManager* e *DataConversion*, e ognuno dei tre costruttori di *Bridge* inizializza la variabile *tipoDB*. Di seguito è presentato il codice del primo costruttore senza parametri: come si nota *Bridge* implementa l'interfaccia *Serializable* contenuta in *java.io*.

```
/*
    ...
*/
import it.easysoft.componenti.dataconnection.ConnectionManager;
import it.easysoft.utility.data_conversion.DataConversion;

public class Bridge implements Serializable {
    /*
        ...
    */
    private static short tipoDB;
    /*
        ...
    */
}
```

```
*/
public Bridge() {
    this.tipoDB = ConnectionManager.getTipoDB();
    lista = new Vector();
}
/*
    definizioni dei metodi della classe Bridge
*/
}
```

Il seguente il metodo popola, con i tipi degli attributi, il vettore che contiene le informazioni dei campi delle tabelle richiamate dalle query del programma.

```
private void getNamesAndTypes(ResultSetMetaData rsMetaData) {

    try {
        // ottengo il database corrente
        switch(tipoDB) {
            case Firebird15: case FirebirdXX:
                /*
                    caso poco interessante
                */
                break;
            case OracleXE:
                for(int runner = 1; runner <= columns; runner++ ) {
                    this.names[runner] = rsMetaData.getColumnLabel(runner );
                    // OFFE è l'alias di alcuni campi che hanno sempre valore VARCHAR
                    if(names[runner].equals("OFFE")) {
                        types[runner] = java.sql.Types.VARCHAR;
                    }
                    else {
                        // caso VARCHAR
                        if(rsMetaData.getColumnTypeName(runner ).equals("VARCHAR2")) {
                            types[runner] = java.sql.Types.VARCHAR;
                        }
                        else {
                            // caso DATE
                            if (rsMetaData.getColumnTypeName(runner ).equals("DATE")) {
                                types[runner] = java.sql.Types.DATE;
                            }
                            else {
                                // caso SMALLINT
                                if(rsMetaData.getColumnTypeName(runner ).equals("BINARY_FLOAT")){
                                    types[runner] = java.sql.Types.SMALLINT;
                                }
                                else {
                                    // caso INTEGER
                                    if(rsMetaData.getColumnTypeName(runner ).equals("NUMBER")) {
                                        types[runner] = java.sql.Types.INTEGER;
                                    }
                                    else {
                                        // caso DOUBLE

```

```
        if(rsMetaData.getColumnTypeName(runner).equals("BINARY_DOUBLE"))
        {
            types[runner] = java.sql.Types.DOUBLE;
        }
    }
}
}
}
}
}
break;
// gli identificativi dei tipi estratti sono in caratteri minuscoli
case SqlServerExpress:
    for(int runner = 1; runner <= columns; runner++ ) {
        this.names[runner] = rsMetaData.getColumnLabel(runner );
        if(names[runner].equals("OFFE")) {
            types[runner] = java.sql.Types.VARCHAR;
        }
        else {
            // caso VARCHAR
            if(rsMetaData.getColumnTypeName(runner ).equals("varchar")) {
                types[runner] = java.sql.Types.VARCHAR;
            }
            else {
                // caso DATE
                if(rsMetaData.getColumnTypeName(runner ).equals("datetime")) {
                    types[runner] = java.sql.Types.DATE;
                }
                else {
                    // caso SMALLINT
                    if(rsMetaData.getColumnTypeName(runner ).equals("smallint")){
                        types[runner] = java.sql.Types.SMALLINT;
                    }
                    else {
                        // caso INTEGER
                        if(rsMetaData.getColumnTypeName(runner ).equals("int")) {
                            types[runner] = java.sql.Types.INTEGER;
                        }
                        else {
                            // caso DOUBLE
                            if(rsMetaData.getColumnTypeName(runner ).equals("float")) {
                                types[runner] = java.sql.Types.DOUBLE;
                            }
                        }
                    }
                }
            }
        }
    }
}
}
break;
default: break;
}
```

```
}  
catch(Exception ex) { /* gestione dell'eventuale eccezione */ }  
}
```

4.5.1 CAST a run-time e tipo numerico

Alcune query possono richiedere la conversione esplicita da un tipo ad un altro. In questa operazione si introduce anche il tipo di Firebird NUMERIC(p , s), un numero floating-point con precisione decimale p e scala (lunghezza totale del numero) s . Lo stesso nome è utilizzato in SQL Server Express, mentre per Oracle Database XE si deve utilizzare la parola chiave NUMBER(p , s). La seguente funzione specifica il tipo finale cui si riferisce il CAST a run-time nell'esecuzione di una query.

```
public static String setTipo(String tipo) {  
    /*  
        assegna il tipo adatto alla colonna (ottimizzando il CAST) a seconda  
        del database in uso seguendo questo schema:  
        Firebird: DATE      VARCHAR  INTEGER      SMALLINT      DOUBLE PRECISION  
        NUMERIC  
        OracleXE: DATE      VARCHAR2  NUMBER(38)  BINARY_FLOAT  BINARY_DOUBLE  
        NUMBER  
        SqlSeExp: DATETIME  VARCHAR  INT          SMALLINT      FLOAT  
        NUMERIC  
    */  
    String tipoMod = new String();  
    switch(tipoDB) {  
        // il tipo passato di default è il tipo primitivo per Firebird  
        case Firebird15: case FirebirdXX:  
            tipoMod = tipo;  
            break;  
        case OracleXE:  
            if(tipo.equals("VARCHAR")) {  
                tipoMod = "VARCHAR2";  
            }  
            else {  
                if(tipo.equals("INTEGER")) {  
                    tipoMod = "NUMBER(38)";  
                }  
                else {  
                    if(tipo.equals("SMALLINT")) {  
                        tipoMod = "BINARY_FLOAT";  
                    }  
                    else {  
                        if(tipo.equals("DOUBLE PRECISION") || tipo.equals("FLOAT")) {
```

```

        tipoMod = "BINARY_DOUBLE";
    }
    else {
        if(tipo.equals("NUMERIC")) {
            tipoMod = "NUMBER";
        }
        else {
            tipoMod = tipo;
        }
    }
}
}
break;
case SqlServerExpress:
    /*
        funzionamento simile a quello già visto per la stesso metodo
        nel terzo capitolo
    */

    break;
}
return tipoMod;
}

```

4.5.2 Concatenazione di stringhe

Alcune query richiedono come risultato la concatenazione di due o più campi di tipo stringa, o di un campo con una stringa fissa all'interno della query stessa. Questa operazione è eseguita, sia in Firebird sia in Oracle Database XE, tramite l'operatore `||`. Ad esempio, l'SQL di questi due RDBMS accetta l'istruzione `SELECT campo_stringa_1 || campo_stringa_2` oppure `SELECT campo_stringa || '<valore stringa>'`. Nel dialetto di SQL Server Express, l'operatore di concatenazione è il simbolo `+`, e la sintassi è identica alla precedente.

```

public static String concatena() {
    // usa il simbolo di concatenazione a seconda della sintassi SQL dell'RDBMS
    String conc = new String();
    switch(tipoDB) {
        case Firebird15: case FirebirdXX: case OracleXE:
            conc = " || ";
            break;
        case SqlServerExpress:

```

```
        conc = " + ";
        break;
    }
    return conc;
}
```

4.6 Classe QueryBuild

Come la precedente Bridge, anche questa è una delle classi fondamentali per il funzionamento dell'applicativo. Essa gestisce il corretto funzionamento delle istruzioni di INSERT, UPDATE e DELETE che l'operatore, utilizzando il programma, compie fisicamente sul database finale. Sono state apportate le modifiche necessarie ad integrare il funzionamento delle suddette operazioni per il tipo data. Segue la parte essenziale del codice di tali modifiche:

```
public static String getQuery (Bridge bridge, int rowNum, String nameTable,
    String nameId)
{
    /*
        ...
    */
    switch (tipoOperazione) {
    case Bridge.INSERT:
        query = "INSERT INTO " + nameTable + " (";
        for(int j = 1; j <= bridge.columnsCount(); j++) {
            /*
                aggiunta alla query dei nomi campi della tabella
            */
        }
        query += ") ";
        query += "VALUES (";
        for(int j = 1; j <= bridge.columnsCount(); j++) {
            // la variabile viene inizializzata con il
            destinationField = bridge.getName(j);
            if(j > 1) { query += ", "; }
            switch (bridge.getType(j)) {
            case java.sql.Types.VARCHAR:
                /*
                    inserimento del valore di tipo VARCHAR
                */
                break;
            case java.sql.Types.INTEGER:
                /*
                    inserimento del valore di tipo INTEGER
                */
            }
        }
    }
}
```

```

break;
case java.sql.Types.SMALLINT:
    /*
        inserimento del valore di tipo SMALLINT
    */
break;
case java.sql.Types.DOUBLE:
    /*
        inserimento del valore di tipo DOUBLE
    */
break;
case java.sql.Types.DATE:
    short tipoDB = Bridge.getTipoDB();
    switch(tipoDB) {
        case Firebird15: case FirebirdXX:
            stringValue = "" + stringValue.substring(3, 6) +
                stringValue.substring(0, 3) +
                stringValue.substring(6, 10) + "";
            break;
        case OracleXE:
            /*
                la funzione to_date converte la data standard
                di Oracle XE nel formato specificato
            */
            stringValue = "to_date('" + stringValue.substring(3, 6) +
                stringValue.substring(0, 3) +
                stringValue.substring(6, 10) + "','mm-dd-yyyy')";
            break;
        case SqlServerExpress:
            stringValue = "" + stringValue.substring(6, 10) +
                stringValue.substring(3, 5) +
                stringValue.substring(0, 2) + "";
            break;
        default: break;
    }
    // completamento delle query di INSERT
    query += stringValue;
    break;
}
break;
case Bridge.UPDATE:
    /*
        inizializzazione delle variabili
    */
    query = "UPDATE " + nameTable + " SET ";
    for(int j = 1; j <= bridge.columnsCount(); j++) {
        destinationField = bridge.getName(j);
        if(j > 1) { query += ", "; }
        switch (bridge.getType(j)) {
            case java.sql.Types.VARCHAR:
                /*
                    aggiornamento del valore di tipo VARCHAR
                */

```

```
        */
        break;
        case java.sql.Types.INTEGER:
            /*
                aggiornamento del valore di tipo INTEGER
            */
            break;
        case java.sql.Types.SMALLINT:
            /*
                aggiornamento del valore di tipo SMALLINT
            */
            break;
        case java.sql.Types.DOUBLE:
            /*
                aggiornamento del valore di tipo DOUBLE
            */
            break;
        case java.sql.Types.DATE:
            /*
                aggiornamento del valore di tipo DATE, identico all'inserimento
            */
            break;
    }
}
// completamento della query di UPDATE
query += " WHERE " + nameId + " = " + id.intValue() ;
break;
case Bridge.DELETE:
    /*
        ...
    */
    int idValue = idTable.intValue();
    query = "DELETE FROM " + nameTable + " WHERE " + nameId + " = " + idTable;
    break;
}
}
```

4.7 Classi BeanInput e SchLogin

Queste due classi definiscono dei metodi che ricavano le informazioni di *login*, ossia le informazioni relative all'operatore che si connette.

Tra le altre, *BeanInput* definisce un metodo che, assegnata la data di sistema con il formato italiano, la salva nel database nel formato corretto, a seconda del database engine utilizzato.

```

public String getDateForDB () {
    // il campo estratto è una data
    if (inputCampo == TipoCampo.DATA) {
        String data = this.jTextField1.getDateValue();
        short tipoDB = Short.parseShort(Main.ottieniDitta().getTipoDB());
        switch(tipoDB) {
            case Firebird15: case FirebirdXX:
                data = data.substring(3, 6) + data.substring(0, 3) + data.substring(6,
10);
                break;
            case OracleXE:
                data = data.substring(0, 3) + DataConversion.getMeseXE(data.substring(3,
5)) +
                    data.substring(5, 10);
                break;
            case SqlServerExpress:
                data = data.substring(6, 10) + data.substring(3, 5) + data.substring(0,
2);
                break;
            default: break;
        }
        return data;
    }
    else { return ""; }
}

```

In *SchLogin*, invece, aggiorna un oggetto di tipo *Ditta*, nel quale sono salvate tutte le informazioni relative all'azienda utente (partiva IVA, indirizzo, e-mail e tante altre), oltre che ai permessi assegnati all'operatore che accede e usa il programma. Inoltre, con le nuove modifiche, all'oggetto *Ditta* deve essere assegnato il nome dell'RDBMS in uso, tramite le seguenti istruzioni:

```

/*
    ...
*/
String tipoDB = Integer.toString( ((Integer) vDitta.get(15)).shortValue());
ditta.setTipoDB(tipoDB);
/*
    ...
*/

```

4.8 Query con STARTING WITH

Firebird permette di specificare, nelle clausole WHERE delle query, il costrutto *<campo> STARTING WITH '<stringa o carattere>'*: esso individua i record relativi

al campo specificato che iniziano con una data stringa o con un singolo carattere.

I dialetti degli altri RDBMS non contemplano questa soluzione, e, visto che questo costrutto è stato ampiamente utilizzato durante la programmazione del gestionale per quanto concerne le ricerche, è necessario sostituire questa istruzione con la classica `<campo> LIKE '<stringa o carattere>%'`. Il codice generico, per questo genere di clausole, risulta quindi essere il seguente:

```
/*
...
*/
String condition = "";
// è la variabile contenente il pattern di ricerca passato dall'operatore
txt = QueryBuild.getFiltro(txt);
if (!txt.equals("")) {
// caso 'inizia con' (tasto F5)
if (type == 1) condition = "AND /* campo */ LIKE '" + txt + "%'";
// caso 'contiene' (tasto F9)
if (type == 2) condition = "AND /* campo */ LIKE '%" + txt + "%'";
// caso 'è uguale a' (tasto ENTER)
if (type == 3) condition = "AND /* campo */ = " + txt;
}
/*
...
*/
```

4.9 Altre modifiche

Sono necessarie diverse altre modifiche, di lieve entità rispetto le precedenti, ma, nonostante questo, indispensabili per il funzionamento tra i vari RDBMS candidati, a causa delle differenze sintattiche dei vari dialetti SQL. Tra le molteplici, le principali sono due:

- Quando viene dichiarato un alias di un campo, Firebird permette di utilizzare, successivamente nella query, sia il nome originale del campo sia l'alias, indifferentemente; questo, invece, non è assolutamente permesso né da Oracle Database XE, né da SQL Server Express, che impongono l'utilizzo esclusivo dell'alias se questo è stato dichiarato.

- Per motivi prettamente relativi alla progettazione del gestionale, nelle maschere che visualizzano un *id*, la query di SELECT che estrae i dati da inviare a video contiene due volte il campo identificativo della tabella (per portare un esempio, nel caso della tabella *ARTICOLI* la query sarà del tipo `SELECT idcodice, idcodice, ... FROM ARTICOLI ...`): Firebird e Oracle XE accettano questa sintassi, mentre la stessa è rifiutata da SQL Server, che pretende sia dichiarato un alias per almeno uno dei due campi uguali.

5. Studio comparativo dei DBMS candidati al funzionamento in EASY

In questo studio, le prestazioni dei database sono rilevate tramite un processo di valutazione in cui i quattro database candidati e quello attualmente in uso sono comparati sulla base di misure oggettive e soggettive durante lo svolgimento dei task essenziali di amministrazione dei database e nei casi d'uso concreti (ordinari e critici, cioè che più impegnano il motore del database) dell'applicativo EASY.

5.1 Convenzioni per i nomi dei database

Per convenzione, i nomi dei database utilizzati in seguito saranno:

- *FB 1.0* per Firebird® 1.0.3 Version WI-V6.2.972 (Superserver)
- *FB 1.5* per Firebird® 1.5 Version 1.5.2.4870 (Superserver)
- *FB 2.0* per Firebird® 2.0 Release Candidate Version 2.0.0.12654 (Superserver)
- *SS* per Microsoft® SQL Server 2005 Express Edition Service Pack 1
- *OR* per Oracle® Database 10g Express Edition Release 2 (10.2.0.1.0)

5.2 Configurazioni dei pc utilizzati

Nello studio si tiene conto dell'architettura di rete presente nella realtà lavorativa del cliente finale, perciò sono stati utilizzati un pc server e un pc client. La rete utilizzata è una rete aziendale da 100Mbit/s.

5.2.1 Pc server

- *Sistema Operativo:*

Microsoft® Windows XP® Professional Versione 2002 Service Pack 2

- *Processore:*

Dual Intel® Pentium® 4 CPU 3.20 GHz, cache 2MB ECC Sincrono ATC

- *Controller IDE ATA/ATAPI:*

Intel® 82801FB Ultra ATA Storage Controllers – 2651

Intel® 82801FB/FBM Ultra ATA Storage Controllers - 266F

- *Hard disk:*

Maxtor® 6B160M0 160GB Serial ATA/300 7200RPM, buffer 8MB

- *Memoria:*

504MB DDR2-533 (PC2-4300)

5.2.2 Pc client

- *Sistema Operativo:*

Microsoft® Windows XP® Professional Versione 2002 Service Pack 2

- *Processore:*

Dual Intel® Pentium® 4 CPU 3.20 GHz, cache 2MB ECC Sincrono ATC

- *Controller IDE ATA/ATAPI:*

Intel® 82801GB Serial ATA Storage Controllers – 27C0

Intel® 82801GB Ultra ATA Storage Controllers – 27DF

- *Hard disk:*

Maxtor® 6V160E0 160GB Serial ATA2 7200RPM, buffer 8MB

- *Memoria:*

0.99GB DDR2-533 (PC2-4300)

5.3 Task

Un'operazione (*task*) è definita come un'unità completa di lavoro, composta da uno o più passi (*step*), ognuno dei quali produce un effetto significativo sull'alterazione dello stato del database: creare un utente è, per esempio, uno step. Ogni task è misurato in tempo e complessità.

5.4 Tempo e complessità

Il tempo è definito come il totale dei secondi necessari per completare un task voluto dall'utente.

Per i task asincroni, cioè quando l'utente (amministratore) può completare un'operazione fintantoché altre stanno progredendo in background, il tempo è misurato esclusivamente come i secondi necessari all'utente per compiere i passi di configurazione, inizializzazione e lancio di un determinato task.

Per tutti gli altri task, cioè quelli sincroni, in cui l'utente deve attendere il termine di un task precedente prima di proseguire con un altro (ad esempio un ripristino di dati nel database in uso), il tempo è calcolato sommando il tempo di configurazione ed esecuzione da parte dell'utente e il tempo che impiega il database a completare l'operazione.

La complessità, invece, è definita come il numero di step necessari a completare un determinato task.

5.5 Distribuzione ponderata del lavoro nell'amministrazione del database

Nel complesso dell'amministrazione del database, i task hanno differenti livelli di importanza e complessità, e sono eseguiti con diversi livelli di frequenza. Ad esempio, le operazioni necessarie per creare una nuova tabella oppure per aggiungere un nuovo attributo sono eseguite di gran lunga più frequentemente che non creare un nuovo database.

Per tenere conto accuratamente di questi livelli, ogni insieme di task (*area*) di amministrazione del database è suddiviso in porzioni che misurano, per ognuno di essi, il loro tipico grado di utilizzo. Le aree in cui è possibile suddividere l'amministrazione del database e le relative percentuali di uso sono riassunte nella tabella *Tab. II* e nel grafico di *Fig. 5.1*.

Distribuzione ponderata del lavoro nell'amministrazione del database	
Installazione e configurazione	5%
Amministrazione ordinaria del database	35%
Backup e ripristino dei dati	25%
Controllo delle prestazioni	15%
Altro	20%
Totale	100%

Tab. II

La categoria "Altro" rappresenta operazioni non incluse in questo studio, quali l'acquisizione e manutenzione delle licenze del software e l'aggiornamento del database. Nell'analisi della distribuzione del lavoro, seppure necessarie, sono considerate attività separate dall'"Amministrazione ordinaria del database", ed hanno uguale peso per ogni prodotto candidato, in modo da non favorire un

database rispetto agli altri. Quest'area non è quindi significativa ai fini della valutazione prestazionale e, pertanto, non verrà presa in considerazione.

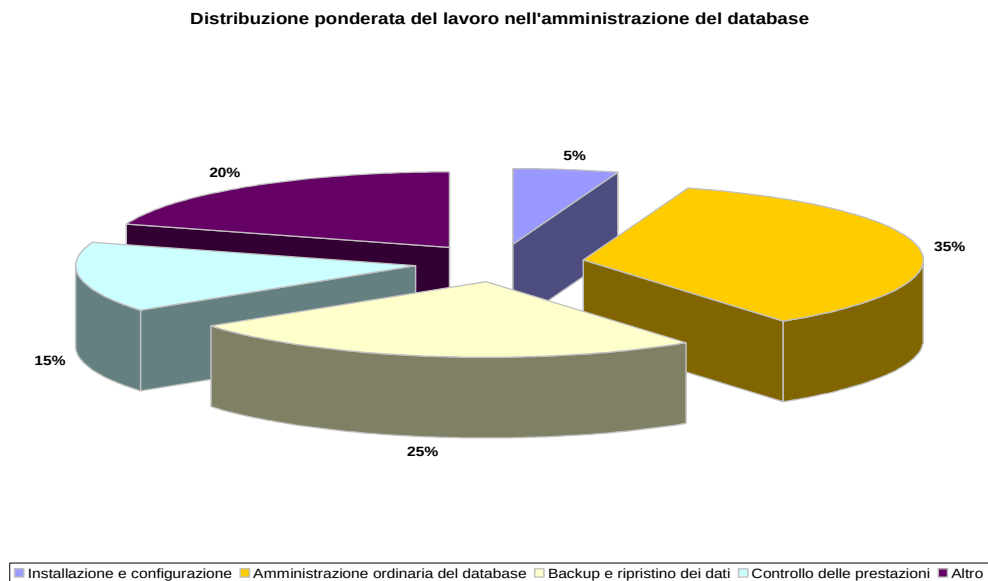


Fig. 5.1

5.5.1 Dettaglio delle aree di amministrazione

Di seguito, è riportato l'elenco dei task che compongono ogni singola area di amministrazione del database individuata nello studio.

- *Installazione e configurazione*
 - Installazione del software e del database: settaggio *out-of-box*
 - Creazione di un database e di un'istanza del server
- *Amministrazione ordinaria del database*
 - Creazione dell'utente con ruoli e privilegi
 - Creazione del tablespace
 - Creazione di una tabella
 - Creazione di un indice
 - Compattazione delle tabelle frammentate

- Lettura dati da un file di script
- *Backup e ripristino dei dati*
 - Configurazione ed esecuzione del backup
 - Ripristino di una tabella cancellata
 - Ripristino del database
 - Ripristino dati da una transazione errata
- *Controllo delle prestazioni (solo per SS e OR)*
 - Diagnosi dei problemi prestazionali
 - Risoluzione dei problemi prestazionali
 - Settaggio della memoria

5.6 Risultati della comparazione degli RDBMS nelle aree di amministrazione

Alfine di mettere a confronto i cinque RDBMS in esame, si è utilizzato il database più popolato in possesso dell'azienda. Le caratteristiche di questo database sono le seguenti:

- Dimensione fisica del file di dati *.gdb* originario: 608 MB.
- Numero tabelle: 124.
- Numero chiavi esterne: 191.
- Numero indici definiti dall'utente: 20.
- Numero righe totali: 3.270.515.
- Numero campi totali: 77.172.718.
- Numero righe nella tabella *ARTICOLI*: 408.946.

- Numero righe nella tabella *CONDIZIONI*: 1.070.839.
- Dispersione del database (campi nulli / campi totali): 47.59%.

Da notare che, dopo il trasferimento dei dati, la dimensione fisica del file di dati per ciascun database ha subito variazioni trascurabili (comprese tra ± 10 MB, mentre tutte le altre caratteristiche sono rimaste ovviamente invariate.

Nella tabella *Tab. III* sono raccolti i tempi e le complessità rilevati per le operazioni di amministrazione. Ciascun task è stato eseguito una sola volta per ogni diverso RDBMS, in quanto una singola operazione di amministrazione non subisce nessun incremento prestazionale se ripetuta più volte, al contrario delle istruzioni SQL, come sarà osservato nel paragrafo 5.8.

Nella tabella che segue, inoltre, sono evidenziati gli scostamenti percentuali da *FB 1.5*, in modo da evidenziare immediatamente il guadagno o la perdita, in termini di tempistica e complessità, per ognuno dei database candidati alla sostituzione.

La formula per calcolare gli scostamenti utilizzati è:

$$(DB - FB\ 1.5) / FB\ 1.5$$

dove *DB* è il valore calcolato, rispettivamente, per *FB 1.0*, *FB 2.0*, *SS* e *OR*.

La formula per il calcolo dei tempi e delle complessità totali è:

$$\sum_i (Area_i \times Peso_i) / \sum_i (Peso_i)$$

con $i = \{\text{installazione e configurazione, amministrazione ordinaria del database, backup e ripristino dei dati}\}$ e *Peso* = percentuale di distribuzione del lavoro.

Queste formule valgono per il calcolo di tutti gli scostamenti da *FB 1.5* e dei totali riportati in questo studio.

Comparazione dei database candidati											
Task	Descrizione	Tempo (secondi) ⁽¹⁾					Complessità (step)				
		FB 1.5	FB 1.0	FB 2.0	SS	OR	FB 1.5	FB 1.0	FB 2.0	SS	OR
Installazione e configurazione											
1	Installazione software e database	213	241	208	2157	1325	1	1	1	3	1
2	Creazione database e istanza server	174	182	164	372	633	1	1	1	2	2
	Subtotale tempo area ⁽²⁾	6,45	7,05	6,20	42,15	32,63	2	2	2	5	3
	Scostamento da FB 1.5		+9,3%	-3,9%	+553,5%	+405,9%		+0%	+0%	+150%	+50%
Amministrazione ordinaria del database											
3	Creazione utente con ruoli e privilegi	30	32	30	62	51	2	2	2	2	2
4	Creazione tablespace	20	23	21	32	55	1	1	1	2	2
5	Creazione tabella	15	15	16	35	33	1	1	1	1	1
6	Creazione indice	12	14	12	19	15	1	1	1	1	1
7	Compattazione tabelle frammentate	14	30	12	16	25	1	1	1	2	1
8	Lettura dati da file script	31	36	32	52	61	2	2	2	1	2
	Subtotale tempo area ⁽²⁾	2,03	2,50	2,05	3,60	4,00	8	8	8	9	9
	Scostamento da FB 1.5		+23%	+0,8%	+77%	+96,7%		+0%	+0%	+12,5%	+12,5%
Backup e ripristino dei dati											
9	Backup database	115	134	111	78	225	3	3	3	2	2
10	Ripristino database	481	512	486	71	192	3	3	3	1	2
11	Ripristino tabella cancellata	115	134	111	78	20	5	5	5	4	1
12	Ripristino dati da transazione errata	481	512	486	78	27	5	5	5	4	1
	Subtotale tempo area ⁽²⁾	19,87	21,53	19,90	5,08	7,73	16	16	16	11	6
	Scostamento da FB 1.5		+8,4%	+0,2%	-74,4%	-61,1%		+0%	+0%	-31,3%	-62,5%
Controllo delle prestazioni											
13	Diagnosi problemi prestazionali	non disp.	non disp.	non disp.	185	131	non disp.	non disp.	non disp.	3	1
14	Risoluzione problemi prestazionali	non disp.	non disp.	non disp.	152	97	non disp.	non disp.	non disp.	4	3
15	Settaggio memoria	non disp.	non disp.	non disp.	0	104	non disp.	non disp.	non disp.	0	2
	Subtotale tempo area ⁽³⁾				5,62	5,53				7	6
	Scostamento da SS					-1,5%					-14,3%
Totali											
	Tempi e complessità ⁽⁴⁾	9,23	10,17	9,23	7,14	7,64	10,62	10,62	10,62	9,46	7,38
	Scostamento da FB 1.5		+10,2%	+0%	-22,7%	-17,3%		+0%	+0%	-10,9%	-30,4%
⁽¹⁾ I tempi sono arrotondati per eccesso al secondo superiore. Tutti i task s'intendono eseguiti in locale sul pc server.											
⁽²⁾ I subtotali dei tempi sono espressi in minuti (somma secondi / 60) e arrotondati al centesimo di minuto.											
⁽³⁾ I subtotali dei tempi e delle complessità misurati per le prestazioni mettono a confronto solamente SS e OR.											
⁽⁴⁾ I totali dei tempi e delle complessità non tengono conto dell'area di controllo delle prestazioni.											

Tab. III

I seguenti grafici (Figg. 5.2 – 5.6) visualizzano in maniera più chiara i principali risultati ottenuti in precedenza.

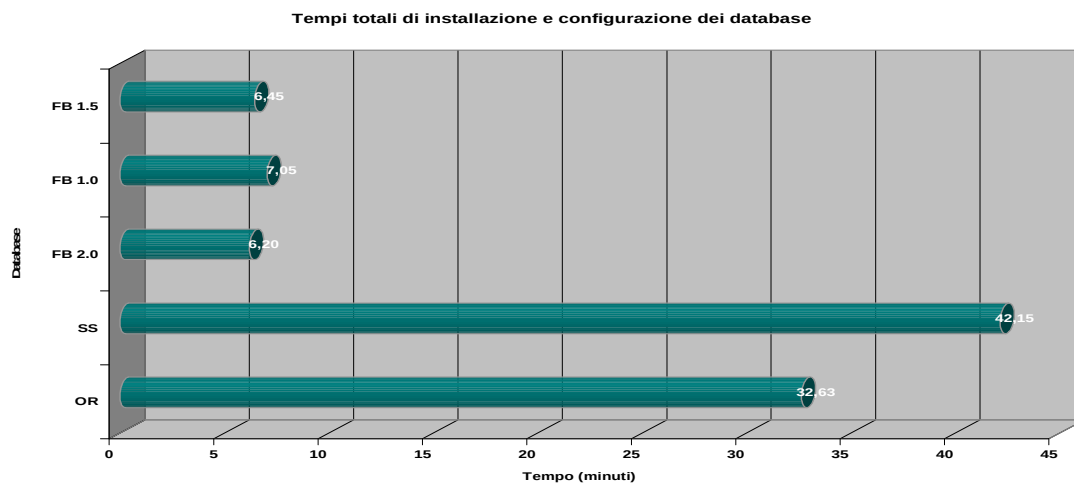


Fig. 5.2

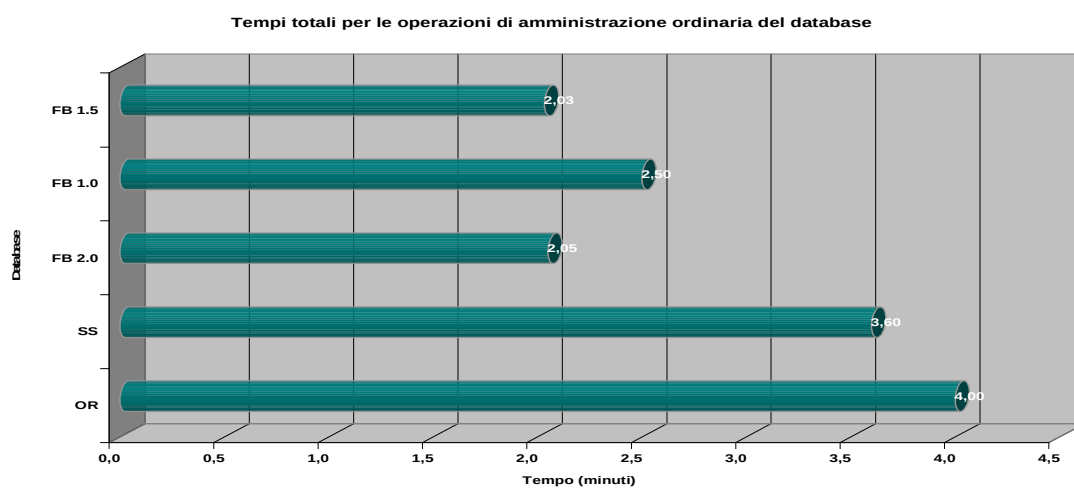


Fig. 5.3

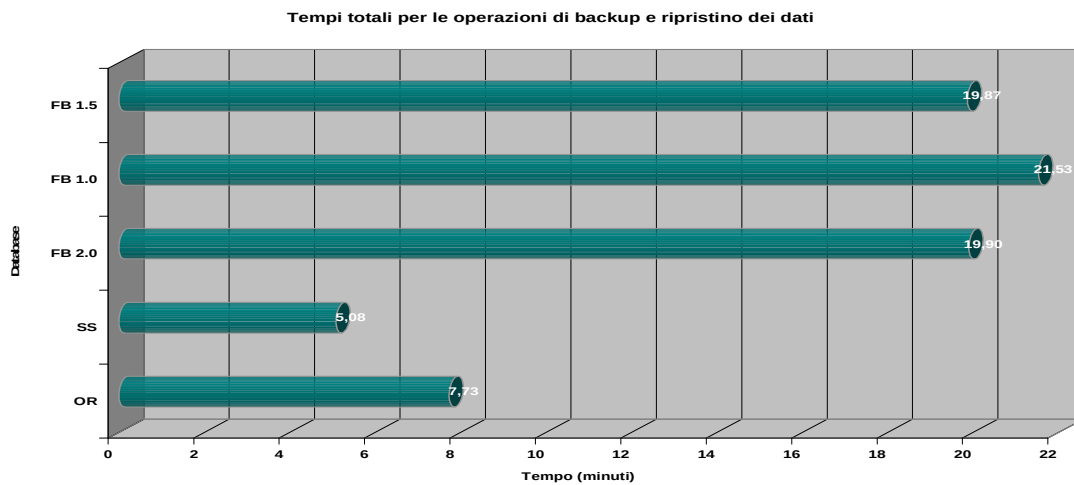


Fig. 5.4

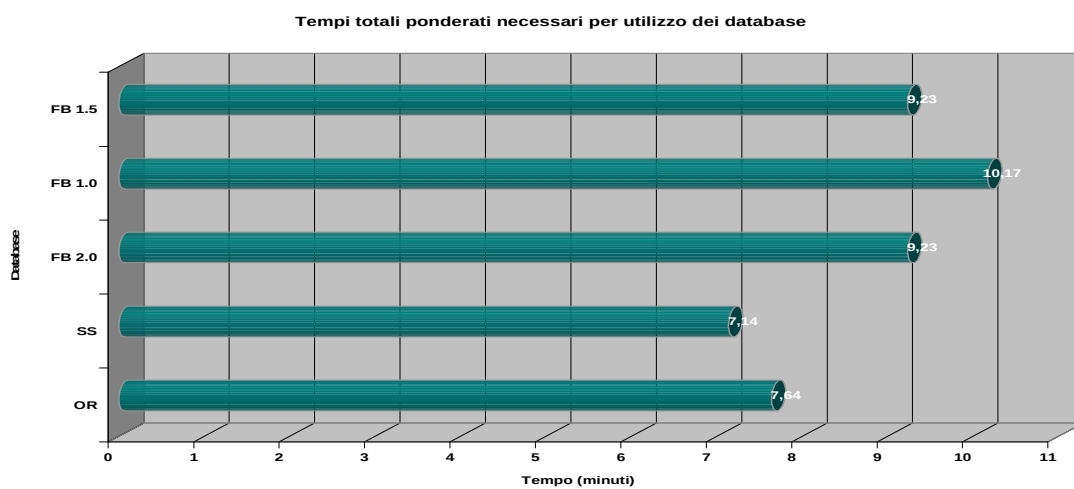


Fig. 5.5

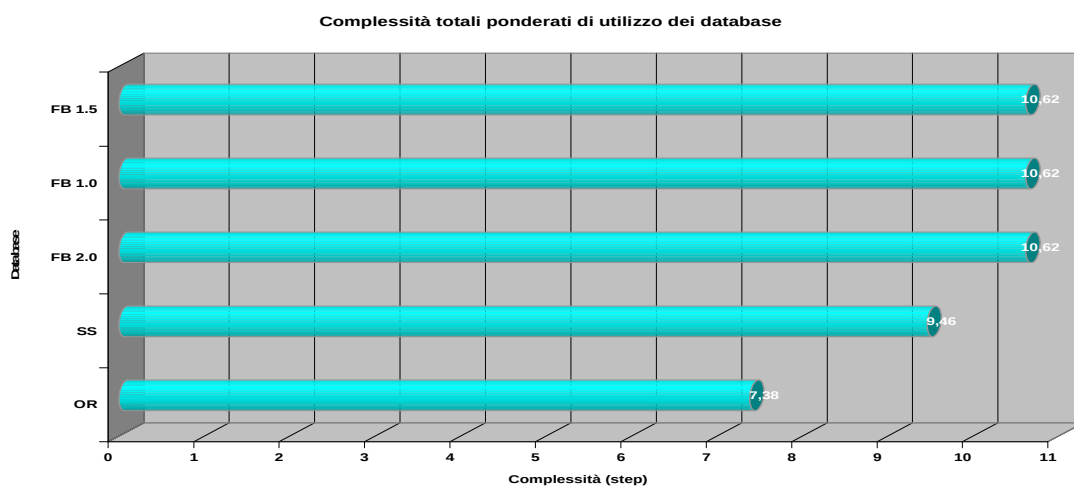


Fig. 5.6

5.6.1 Osservazioni sui risultati precedenti

Innanzitutto, le prestazioni della famiglia *FB* costituiscono già un sottoinsieme di paragone: si registrano dei leggeri miglioramenti nel passaggio di versione da *1.0* a *1.5*, mentre tra la versione *1.5* e la *2.0* sostanzialmente non vi sono differenze degne di nota.

Per quanto riguarda gli altri due database, *SS* e *OR*, l'unico effettivo decremento prestazionale rispetto alla famiglia *FB* si riscontra nell'installazione dei prodotti, mentre per le suddette due aree le prestazioni sono più che paragonabili: nell'amministrazione ordinaria si nota un incremento dei tempi per *SS* e *OR* rispettivamente del 77 e del 96,7% circa, ma la differenza è minima, poiché si tratta comunque di scarti massimi inferiori ai 2 minuti rispetto ai tempi di *FB 1.5*. Un notevole miglioramento, invece, si nota per quanto riguarda l'area di manutenzione dei dati, ossia dei tempi di backup e ripristino: le prestazioni di *OR* rispetto a *FB 1.5* sono migliori del 61,1%, mentre per *SS* i tempi si accorciano addirittura del 74,4%. Le differenze, in questo caso, sono più consistenti, nell'ordine di 10 minuti od oltre.

Non costituisce termine di paragone l'area di controllo delle prestazioni, essendo essa disponibile solo con le utilità incluse in *SS* e *OR*.

Analizzando le complessità, nell'area di installazione e configurazione, la famiglia *FB* e *OR* sono leggermente di più semplice esecuzione rispetto a *SS*. Nelle due principali aree di confronto, gli scostamenti più sostanziali si riscontrano solamente nel backup e ripristino dei dati: *SS* e *OR* abbassano notevolmente gli step necessari ad eseguire i task rispetto alla famiglia *FB*.

5.7 Distribuzione ponderata del lavoro nei casi d'uso concreti di EASY

Nell'uso quotidiano del software EASY, ogni utente finale ha esigenze diverse e in modo altrettanto diverso utilizza le varie funzionalità del programma. In linea generale, grazie all'esperienza maturata dagli sviluppatori del programma e dal servizio di assistenza della Easy Soft S.r.l., è possibile riconoscere e raggruppare i task svolti più frequentemente e addirittura più volte durante l'arco della stessa giornata rispetto a quelli svolti in maniera più sporadica, che per la loro complessità impegnano notevolmente il motore del database.

Come per le precedenti aree di amministrazione, ad ogni gruppo di operazioni è assegnato un certo grado di utilizzo, come riassunto in *Tab. IV* e in *Fig. 5.7*.

Distribuzione ponderata del lavoro nei casi d'uso concreto di EASY	
Trasferimento del database	1%
Operazioni di frequente utilizzo	66%
Operazioni di utilizzo estremo del database	19%
Altro	14%
Totale	100%

Tab. IV

Per ogni gruppo di operazioni si analizzano le principali e nel contempo quelle che meglio di altre permettono un effettivo confronto dei cinque database in esame. Nella categoria “*Altro*” sono incluse tutte le altre funzionalità che l'utente finale comunque utilizza, ma che non sono considerate indicative ai fini di questo studio comparativo. Come nell'analogo caso delle aree di amministrazione, questo gruppo di task non sarà esaminato.

Distribuzione ponderata del lavoro nei casi d'uso concreti di EASY

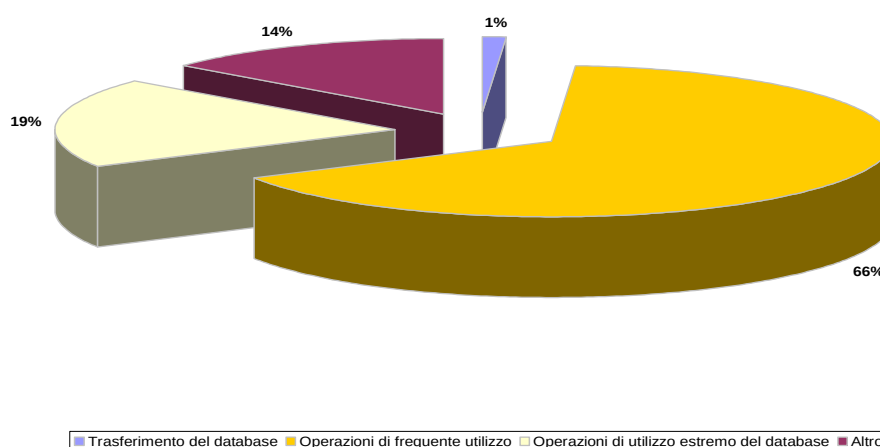


Fig. 5.7

5.7.1 Dettaglio dei gruppi di operazioni

Ogni gruppo di operazioni è suddiviso in base alla frequenza di utilizzo da parte dell'utente finale ed al carico di lavoro cui detti task sottopongono il motore del database.

- *Trasferimento del database*
 - Trasferimento fisico dei dati da *FB 1.5* al database di destinazione
- *Operazioni di frequente utilizzo*
 - Accesso al database
 - Inserimento degli articoli nella fattura
 - Salvataggio dei documenti creati
 - Ricerca degli articoli in modalità "inizia con" (tasto *F5*)
 - Ricerca degli articoli in modalità "contiene" (tasto *F9*)
 - Fatturazione differita
 - Creazione di una vendita al dettaglio

- Raggruppamento di una vendita al dettaglio
- Stampa del registro IVA
- Stampa del bilancio
- *Operazioni di utilizzo estremo del database*
 - Stampa di un partitario contabile
 - Stampa di un partitario di magazzino
 - Ricarico dei listini
 - Riaggiornamento delle giacenze di magazzino

5.8 Risultati della comparazione dei database nei casi d'uso concreti di EASY

La tabella *Tab. V* mostra tutti i valori registrati nell'analisi comparata dei vari sistemi candidati. È utile specificare che per ogni task vengono effettuate tre prove, potendo da un lato osservare che, per ogni operazione ripetuta, ciascuno degli RDBMS opera una sorta di *caching* del comando SQL velocizzandone l'esecuzione, e dall'altro confrontare le medie dei tempi osservati, in maniera tale da "lisciare" le prestazioni dei DBMS alla luce della precedente considerazione.

Le prove sono eseguite dapprima in locale, cioè sul pc server dove sono stati installati i motori dei database, e in seguito eseguite in remoto sul pc client, in modo da confrontare le prestazioni dei candidati in un contesto di rete. A questo proposito, è fondamentale sottolineare il fatto che, al momento dell'esecuzione delle prove per un certo RDBMS, tutti i servizi relativi agli altri DBMS vengono arrestati, in modo da annullare qualsiasi eventuale interazione

tra i sistemi a confronto. Il database utilizzato per i rilievi di questa sezione è il medesimo menzionato nel paragrafo 5.5.

Infine, le formule riguardanti il calcolo degli scostamenti da *FB 1.5* e i tempi totali ponderati sono identiche, nella forma, a quelle utilizzate nel confronto delle aree di amministrazione dei cinque candidati.

Prestazioni dei database candidati nei casi di utilizzo concreti del software EASY											
# op.	Descrizione	Tempo (secondi) ⁽¹⁾ in locale					Tempo (secondi) ⁽¹⁾ in remoto				
		FB 1.5	FB 1.0	FB 2.0	SS	OR	FB 1.5	FB 1.0	FB 2.0	SS	OR
Trasferimento database											
1	Primo trasfer. database ⁽²⁾	4446	4612	4560	1624	2557	4391	4587	4458	1626	2400
2	Secondo trasfer. database ⁽²⁾	4445	4614	4555	1624	2550	4390	4595	4462	1625	2409
3	Terzo trasfer. database ⁽²⁾	4451	4611	4558	1627	2549	4391	4594	4455	1626	2407
	Tempo medio per l'operazione	4447,33	4612,33	4557,67	1625,00	2552,00	4390,67	4592,00	4458,33	1625,67	2405,33
	Scostamento medio da FB 1.5		+3,7%	+2,5%	-63,5%	-42,6%		+4,6%	+1,5%	-63%	-45,2%
	Subtotale tempo medio gruppo ⁽³⁾	74,12	76,87	75,96	27,08	42,53	73,18	76,53	74,31	27,09	40,09
	Scostamento da FB 1.5		+3,7%	+2,5%	-63,5%	-42,6%		+4,6%	+1,5%	-63%	-45,2%
Operazioni di frequente utilizzo											
4	Primo accesso al database clienti	10,62	15,68	8,12	5,14	9,71	10,51	14,7	10,07	7,32	10,01
5	Secondo accesso al database clienti	8,14	13,44	6,01	3,26	5,5	5,54	9,53	6,26	3,6	4,86
6	Terzo accesso al database clienti	5,9	9,25	5,67	3,17	4,62	5,37	9,18	5,28	3,46	4,79
	Tempo medio per l'operazione	8,22	12,79	6,60	3,86	6,61	7,14	11,14	7,20	4,79	6,55
	Scostamento medio da FB 1.5		+55,6%	-19,7%	-53,1%	-19,6%		+56,0%	+0,9%	-32,9%	-8,2%
7	Primo inserimento articoli in fattura	5,12	9,3	4,05	11,56	29,54	4,79	11,02	5,15	9,26	22,41
8	Secondo inserimento articoli in fattura	4,72	6,24	3,03	4,87	23,22	3,35	9,87	3,53	4,09	15,62
9	Terzo inserimento articoli in fattura	3,42	5,11	2,99	3,02	18,84	3,04	8,54	3,39	3,78	15,56
	Tempo medio per l'operazione	4,42	6,88	3,36	6,48	23,87	3,73	9,81	4,02	5,71	17,86
	Scostamento medio da FB 1.5		+55,7%	-24,1%	+46,7%	+440%		+163,2%	+8%	+53,2%	+379,3%
10	Primo salvataggio documenti	9,54	9,56	4,86	9,51	27,35	4,94	8,34	4,82	12,68	14,57
11	Secondo salvataggio documenti	1,56	2,87	1,22	4,66	20,52	1,26	2,34	1,68	7,98	6,65
12	Terzo salvataggio documenti	1,06	2,43	0,82	3,38	12,37	0,95	1,65	1,48	3,64	5,7
	Tempo medio per l'operazione	4,05	4,95	2,30	5,85	20,08	2,38	4,11	2,66	8,10	8,97
	Scostamento medio da FB 1.5		+22,2%	-43,3%	+44,3%	395,4%		+72,4%	+11,6%	+239,9%	+276,5%
13	Prima ricerca articoli (F5)	6,48	9,25	4,32	10,17	44,18	2,68	12,29	3,09	9,64	82,25
14	Seconda ricerca articoli (F5)	3,06	8,41	2,87	9,84	40,28	2,36	11,69	2,82	9,61	79,72
15	Terza ricerca articoli (F5)	2,34	7,96	2,55	9,23	38,71	2,28	11,82	2,69	9,48	71,29
	Tempo medio per l'operazione	3,96	8,54	3,25	9,75	41,06	2,44	11,93	2,87	9,58	77,75
	Scostamento medio da FB 1.5		+115,7%	-18%	+146,1%	+936,8%		+389,1%	+17,5%	+292,5%	+3087%
16	Prima ricerca articoli (F9)	29,48	55,09	22,84	8,26	70,34	23,62	56,15	34,42	7,68	97,1
17	Seconda ricerca articoli (F9)	21,95	52,68	19,43	7,41	64,64	21,09	50,93	31,45	7,22	90,28

Tab. V

Prestazioni dei database candidati nei casi di utilizzo concreti del software EASY											
# op.	Descrizione	Tempo (secondi) ⁽¹⁾ in locale					Tempo (secondi) ⁽¹⁾ in remoto				
		FB 1.5	FB 1.0	FB 2.0	SS	OR	FB 1.5	FB 1.0	FB 2.0	SS	OR
Operazioni di frequente utilizzo											
18	Terza ricerca articoli (F9)	19,56	46,36	18,12	6,98	62,58	20,95	50,87	31,33	7,19	57,5
	Tempo medio per l'operazione	23,66	51,38	20,13	7,55	65,85	21,89	52,65	32,40	7,36	81,63
	Scostamento medio da FB 1.5		+117,1%	-14,9%	-68,1%	+178,3%		+140,6%	+48%	-66,4%	+273%
19	Prima fattura differita	1,75	2,55	1,03	5,32	5,21	1,23	1,42	1,45	5,34	4,39
20	Seconda fattura differita	1,31	1,62	0,89	3,27	4,41	0,86	1,39	1,17	3,01	3,14
21	Terza fattura differita	1,15	1,31	0,84	1,78	2,56	0,81	1,37	0,95	1,92	2,43
	Tempo medio per l'operazione	1,40	1,83	0,92	3,46	4,06	0,97	1,39	1,19	3,42	3,32
	Scostamento medio da FB 1.5		+30,2%	-34,4%	+146,3%	+189,3%		+44,1%	+23,1%	+254,1%	+243,4%
22	Prima vendita al dettaglio	2,15	2,03	1,3	2,13	6,61	1,32	1,54	1,61	2,07	6,59
23	Seconda vendita al dettaglio	1,94	1,88	0,83	1,52	5,72	0,79	1,5	1,12	1,45	5,84
24	Terza vendita al dettaglio	1,15	1,53	0,52	1,04	5,18	0,46	1,48	0,98	0,92	5,59
	Tempo medio per l'operazione	1,75	1,81	0,88	1,56	5,84	0,86	1,51	1,24	1,48	6,01
	Scostamento medio da FB 1.5		+3,8%	-49,4%	-10,5%	+234,2%		+75,9%	+44,4%	+72,8%	+601,2%
25	Primo raggruppamento vendite al dettaglio	0,81	0,79	0,65	1,11	2,35	0,7	0,82	0,85	0,87	2,49
26	Secondo raggruppamento vendite al dettaglio	0,7	0,66	0,54	0,66	1,57	0,54	0,65	0,68	0,5	1,74
27	Terzo raggruppamento vendite al dettaglio	0,64	0,42	0,42	0,32	0,92	0,52	0,62	0,54	0,34	1,06
	Tempo medio per l'operazione	0,72	0,62	0,54	0,70	1,61	0,59	0,70	0,69	0,57	1,76
	Scostamento medio da FB 1.5		-13%	-25,1%	-2,8%	+125,1%		+18,8%	+17,6%	-2,8%	+200,6%
28	Prima stampa registri IVA	3,96	4,45	3,14	3,22	4,42	2,84	3,27	1,92	3,84	3,89
29	Seconda stampa registri IVA	1,12	1,81	1,25	1,25	0,82	1,28	2,07	1,01	1,23	0,75
30	Terza stampa registri IVA	1,06	1,71	1,09	1,03	0,76	1,25	1,96	0,98	1,04	0,67
	Tempo medio per l'operazione	2,05	2,66	1,83	1,83	2,00	1,79	2,43	1,30	2,04	1,77
	Scostamento medio da FB 1.5		+29,8%	-10,7%	-10,4%	-2,3%		+35,9%	-27,2%	+13,8%	-1,1%
31	Prima stampa bilancio	2,34	1,84	1,62	1,79	2,14	1,71	2,29	1,15	1,96	1,34
32	Seconda stampa bilancio	0,95	1,37	0,96	0,79	0,93	0,87	1,57	0,95	0,92	0,87
33	Terza stampa bilancio	0,81	1,26	0,82	0,76	0,73	0,78	1,43	0,7	0,75	0,65
	Tempo medio per l'operazione	1,37	1,49	1,13	1,11	1,27	1,12	1,76	0,93	1,21	0,95
	Scostamento medio da FB 1.5		+9%	-17,1%	-18,5%	-7,3%		+57,4%	-16,7%	+8%	-14,9%
	Subtotale tempo medio gruppo ⁽³⁾	0,86	1,55	0,68	0,70	2,87	0,71	1,62	0,91	0,74	3,44
	Scostamento da FB 1.5		+80,2%	-20,7%	-18,3%	+233,8%		+127,1%	+27,1%	+3,2%	+381,6%

Tab. V

Prestazioni dei database candidati nei casi di utilizzo concreti del software EASY											
# op.	Descrizione	Tempo (secondi) ⁽¹⁾ in locale					Tempo (secondi) ⁽¹⁾ in remoto				
		FB 1.5	FB 1.0	FB 2.0	SS	OR	FB 1.5	FB 1.0	FB 2.0	SS	OR
Operazioni di utilizzo estremo del database											
34	Prima stampa partitario contabile	11,32	23,03	10,21	46,18	56,31	9,4	44,27	9,07	44,98	57,91
35	Seconda stampa partitario contabile	9,93	20,45	8,54	41,23	32,53	7,12	41,83	7,05	15,42	54,43
36	Terza stampa partitario contabile	6,34	20,28	6,86	39,59	32,14	6,81	40,72	6,94	15,24	52,22
	Tempo medio per l'operazione	9,20	21,25	8,54	42,33	40,33	7,78	42,27	7,69	25,21	54,85
	Scostamento medio da FB 1.5		+131,1%	-7,2%	+360,3%	+338,5%		+443,6%	-1,2%	+224,2%	+605,4%
37	Prima stampa partitario magazzino	32,79	67,54	26,5	9,57	9,05	30,35	56,81	23,62	6,14	8,78
38	Seconda stampa partitario magazzino	30,67	65,24	23,39	4,6	6,57	29,12	55,64	22,93	3,62	6,39
39	Terza stampa partitario magazzino	30,323	64,81	23,36	3,34	4,22	28,96	54,37	22,75	1,26	4,25
	Tempo medio per l'operazione	31,26	65,86	24,42	5,84	6,61	29,48	55,61	23,10	3,67	6,47
	Scostamento medio da FB 1.5		+110,7%	-21,9%	-81,3%	-78,8%		+88,6%	-21,6%	-87,5%	-78%
40	Primo ricarico	58,78	125,04	83,62	89,81	161,32	52,31	74,35	81,23	48,67	154,87
41	Secondo ricarico	52,96	119,17	80,92	57,65	158,96	51,24	73,76	79,86	48,2	153,62
42	Terzo ricarico	52,39	116,86	80,78	53,01	157,41	50,96	72,43	81,78	47,5	152,11
	Tempo medio per l'operazione	54,71	120,36	81,77	66,82	159,23	51,50	73,51	80,96	48,12	153,53
	Scostamento medio da FB 1.5		+120%	+49,5%	+22,1%	+191%		+42,7%	+57,2%	-6,6%	+198,1%
43	Primo riaggiornamento giacenze	1094,47	1424,58	894,23	1742,84	3715,61	955,36	1256,88	901,51	1868,06	3495,73
44	Secondo riaggiornamento giacenze	1085,22	1419,66	888,47	1740,45	3684,24	953,58	1255,35	897,77	1866,47	3490,37
45	Terzo riaggiornamento giacenze	1080,72	1417,47	886,51	1732,17	3651,85	951,19	1253,63	889,62	1861,52	3487,34
	Tempo medio per l'operazione	1086,80	1420,57	889,74	1738,49	3683,90	953,38	1255,29	896,30	1865,35	3491,15
	Scostamento medio da FB 1.5		+30,7%	-18,1%	+60%	+239%		+31,7%	-6%	+95,7%	+266,2%
	Subtotale tempo medio gruppo ⁽³⁾	19,70	27,13	16,74	30,89	64,83	17,37	23,78	16,80	32,37	61,77
	Scostamento da FB 1.5		+37,7%	-15%	+56,8%	+229,1%		+36,9%	-3,3%	+86,4%	+255,6%
Totali											
	Totali	5,87	8,08	5,11	7,68	17,02	5,24	7,39	5,27	8,03	16,75
	Scostamento da FB 1.5		+37,5%	-13,1%	+30,7%	+189,8%		+41,1%	+0,7%	+53,4%	+219,9%
⁽¹⁾ I tempi sono arrotondati per eccesso al centesimo di secondo superiore.											
⁽²⁾ I tempi sono arrotondati per eccesso al secondo superiore.											
⁽³⁾ I subtotali dei tempi sono espressi in minuti (somma secondi / 60) e arrotondati al centesimo di minuto.											

Tab. V

Anche in questo caso, i grafici seguenti (Figg. 5.8 – 5.24) evidenziano meglio i risultati ottenuti.

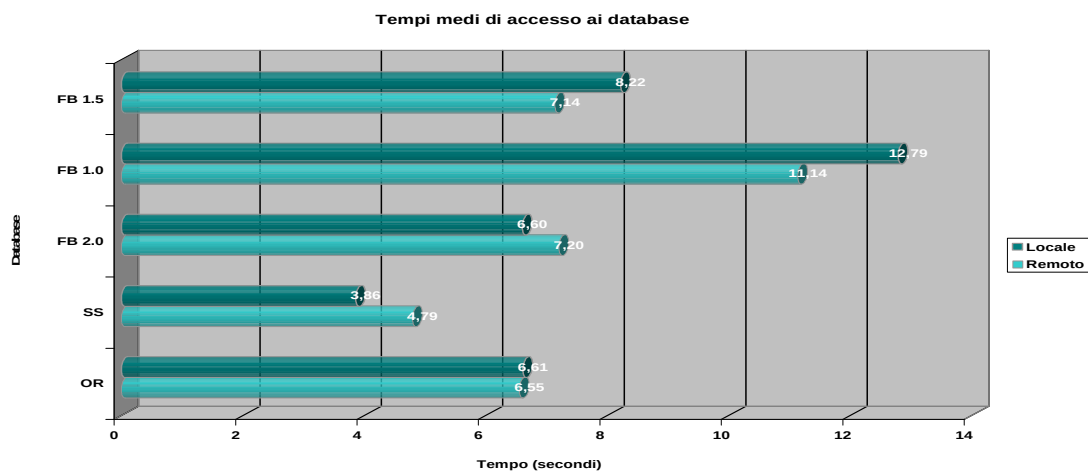


Fig. 5.8

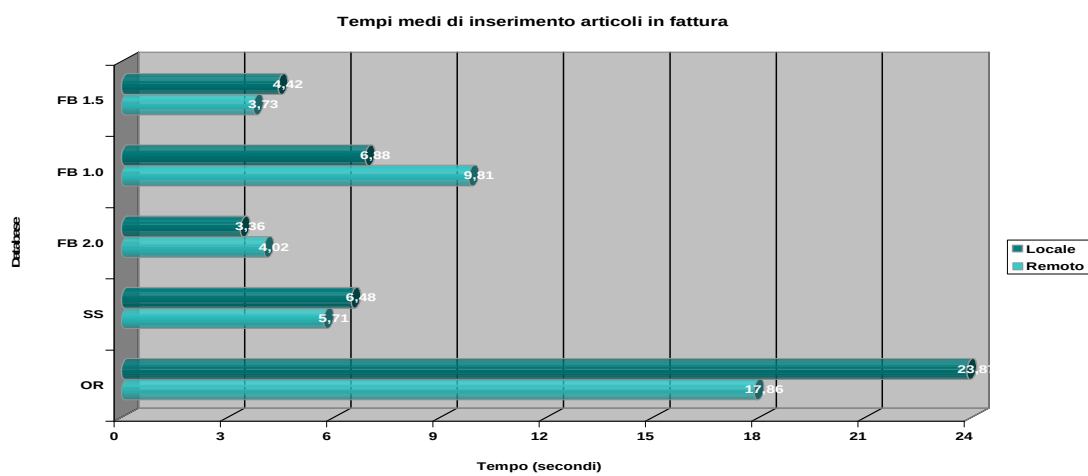


Fig. 5.9

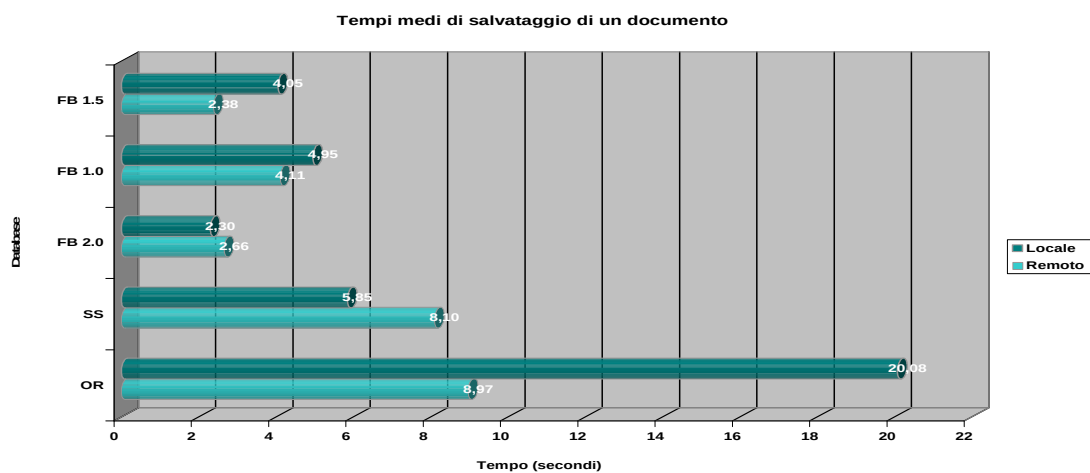


Fig. 5.10

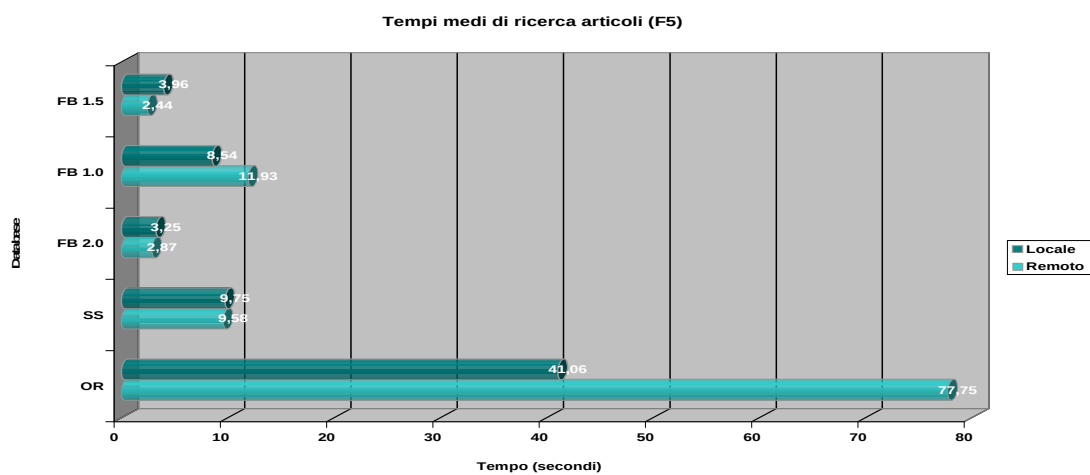


Fig. 5.11

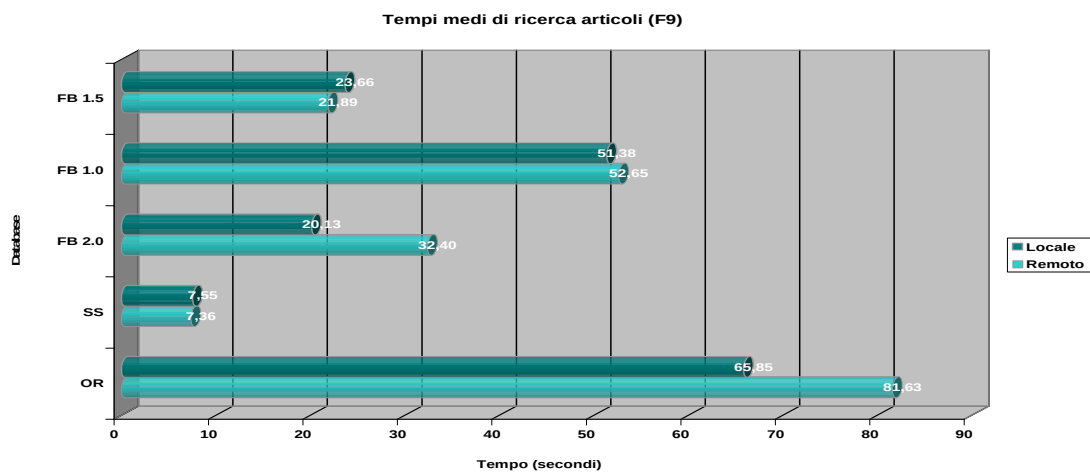


Fig. 5.12

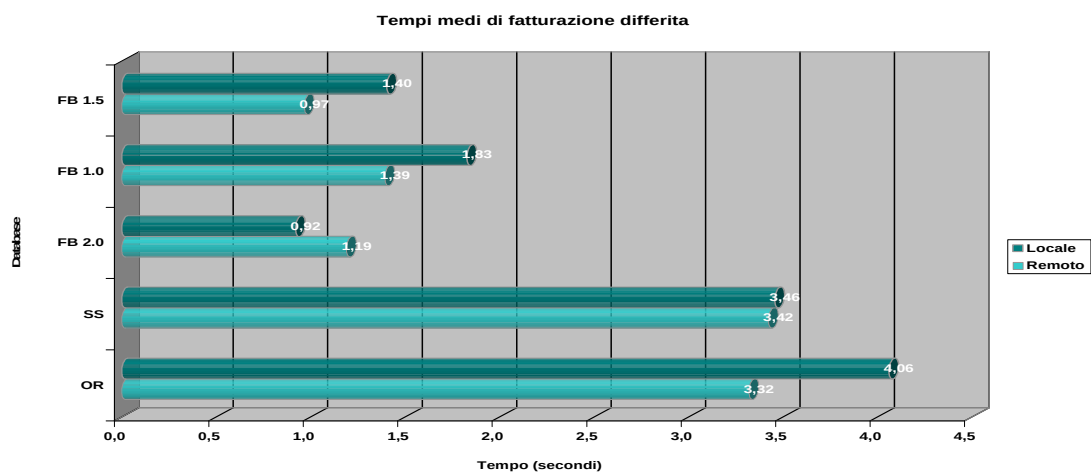


Fig. 5.13

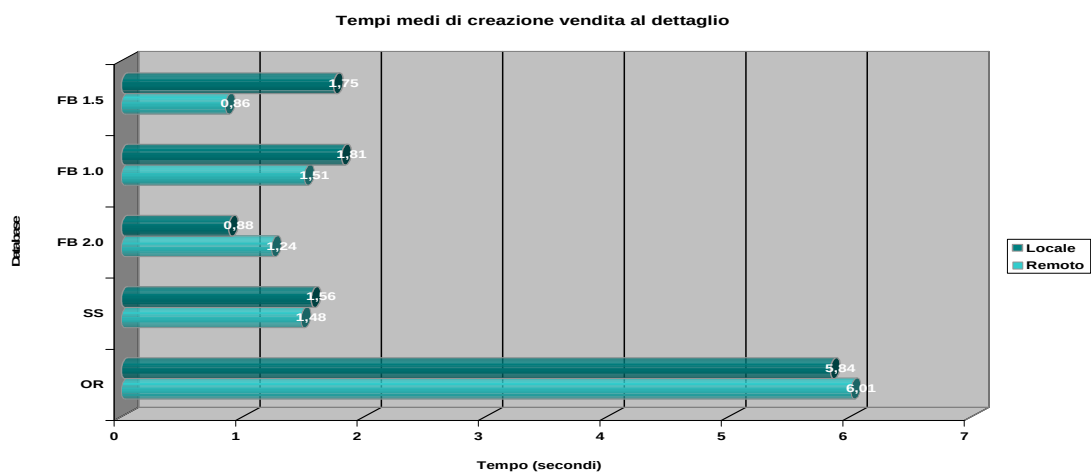


Fig. 5.14

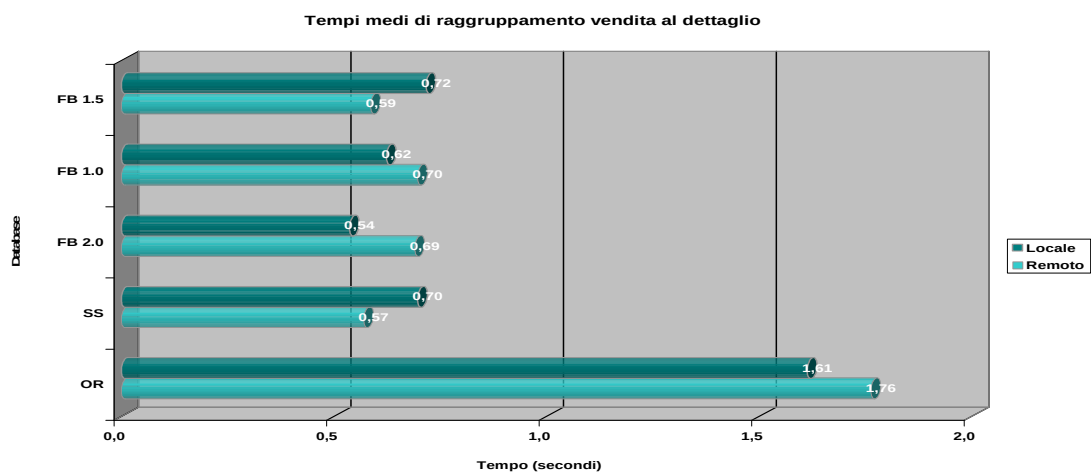


Fig. 5.15

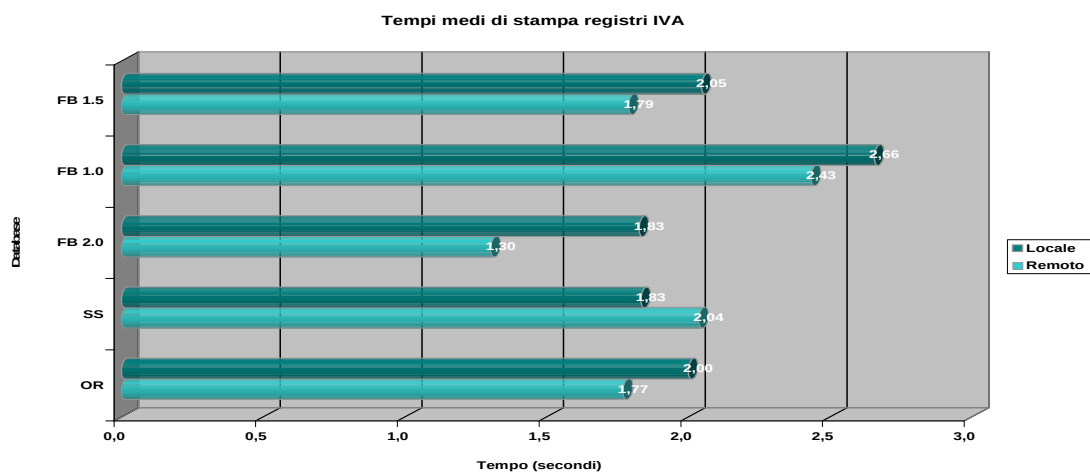


Fig. 5.16

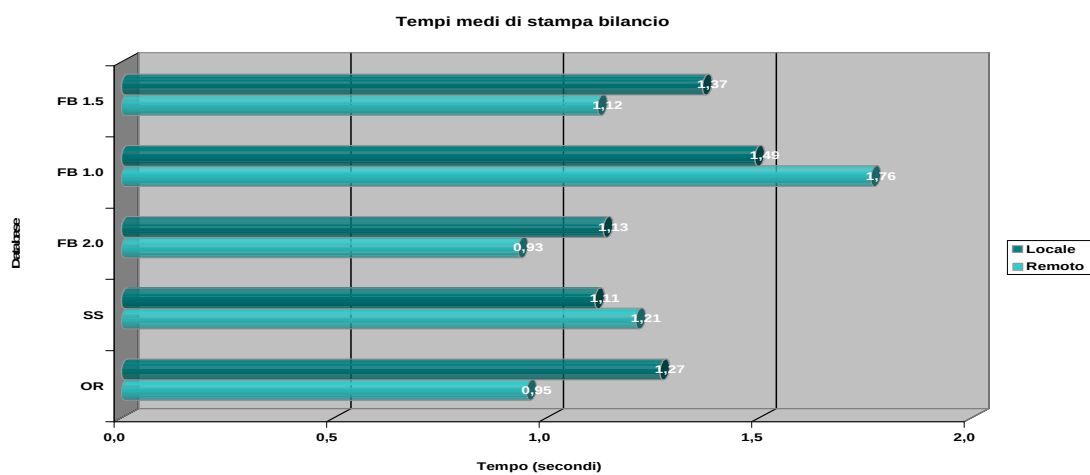


Fig. 5.17

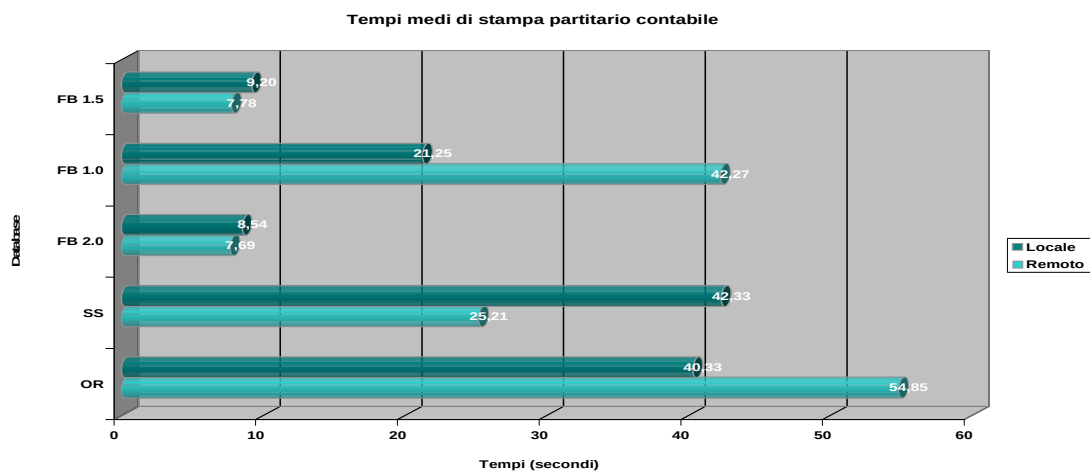


Fig. 5.18

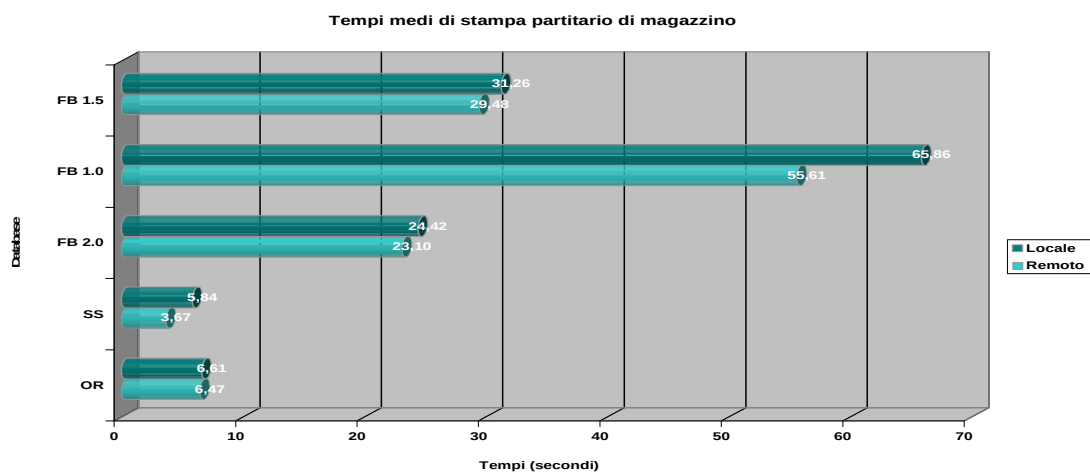


Fig. 5.19

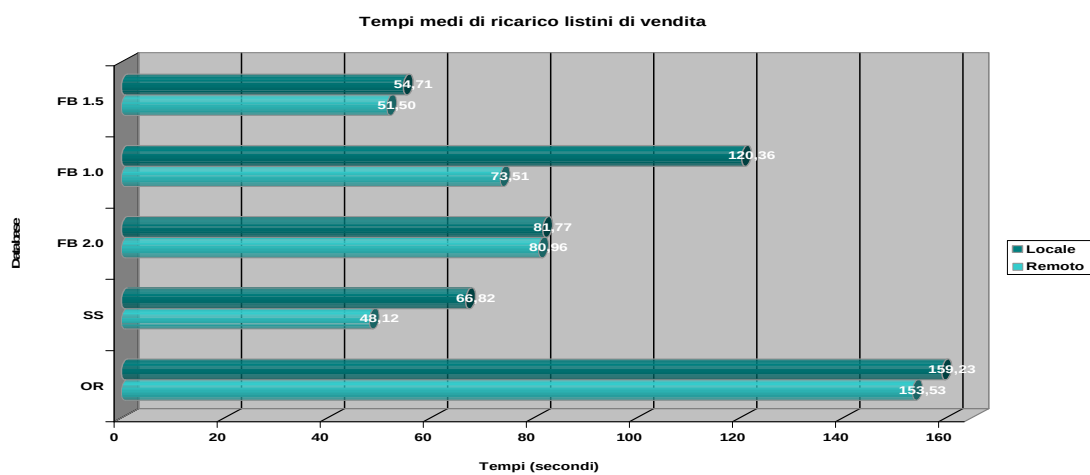


Fig. 5.20

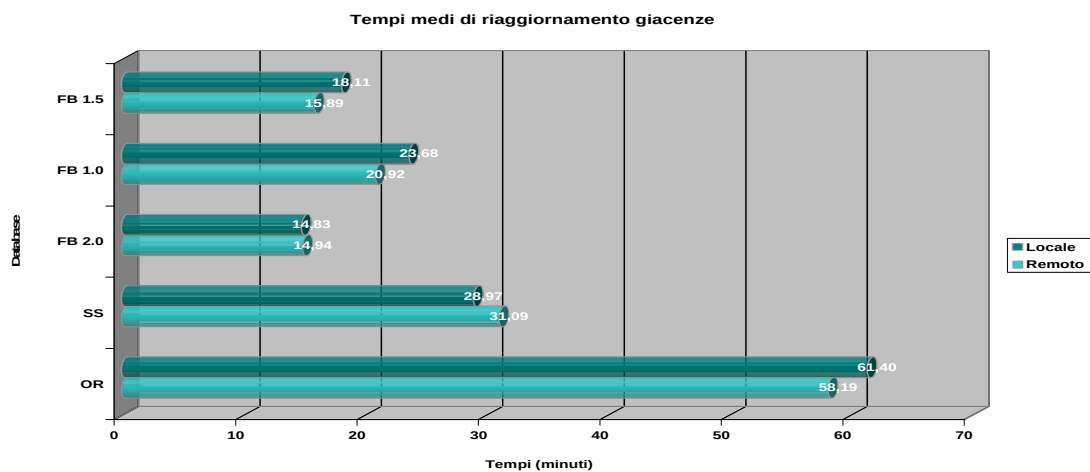


Fig. 5.21

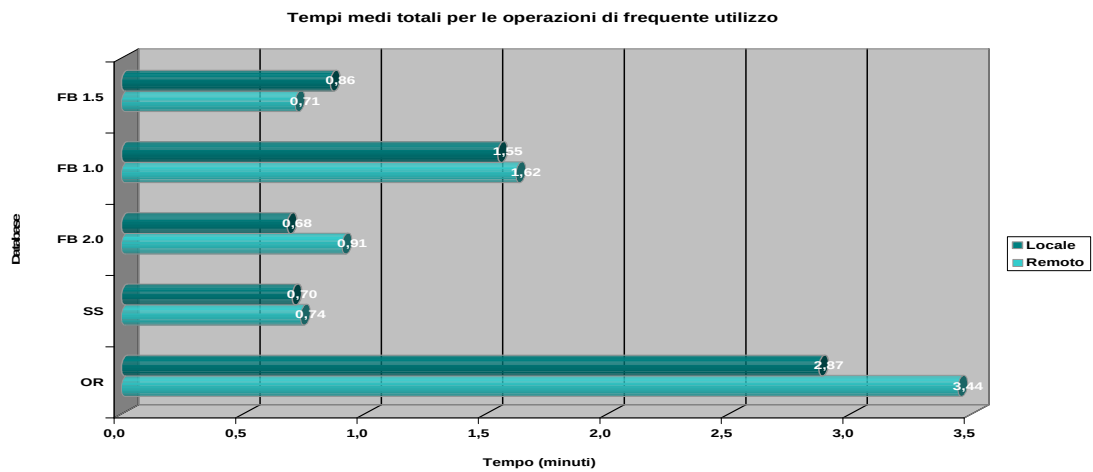


Fig. 5.22

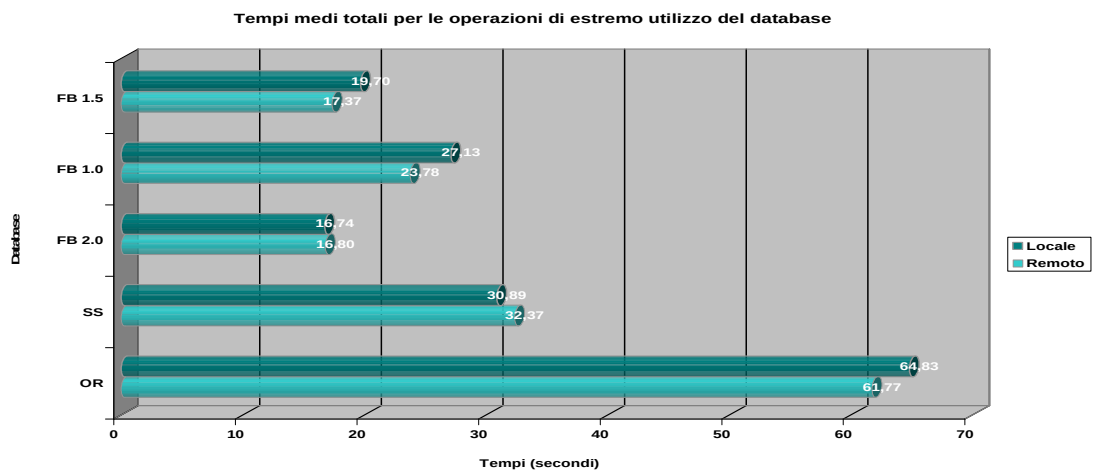


Fig. 5.23

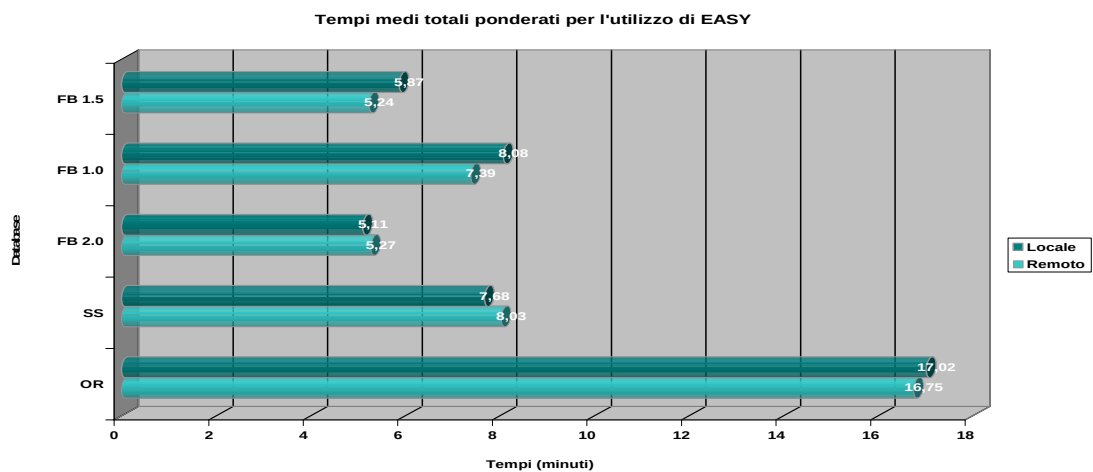


Fig. 5.24

5.8.1 Osservazioni sui risultati precedenti

In generale, non si notano particolari miglioramenti utilizzando un RDBMS diverso da *FB 1.5*. Ciò è evidenziato dagli ultimi tre grafici del paragrafo precedente (*Figg. 5.22 – 5.24*): si può notare che *FB 2.0* e *SS* hanno performance molto simili al primo per quanto riguarda le operazioni di frequenti utilizzo, mentre per i task di utilizzo più sporadico, solo *FB 2.0* fa osservare un leggero miglioramento rispetto a *FB 1.5*. Quest'ultima situazione è rilevabile anche nei tempi medi totali per l'utilizzo di *EASY*, mentre *SS* mostra di avere prestazioni paragonabili a *FB 1.0*. Per quanto riguarda *OR*, in termini di tempi medi totali, fa sempre registrare prestazioni di gran lunga peggiori di *FB 1.5* e degli altri candidati: in generale, i tempi di utilizzo sono rallentati di circa tre volte rispetto all'RDBMS attualmente in uso. Infine, per quanto riguarda *SS* e *OR*, bisogna tener conto dei costi delle licenze delle versioni più evolute rispetto a quella base *Express*.

5.9 Limitazioni e costi delle licenze degli RDBMS utilizzati

Nelle tabelle successive (*Tabb. VI – VII*) sono specificate le caratteristiche principali di ogni database soggetto allo studio, e nei casi delle versioni non gratuite di *SQL Server Express* e *Oracle Database XE*, anche i prezzi medi delle licenze nonché i costi delle *CAL* (Client Access License) aggiuntive.

Licenze dei database candidati					
Descrizione	Database				
	FB 1.5	FB 1.0	FB 2.0	SS	OR
Licenza open-source	sì	sì	sì	no ⁽¹⁾	no ⁽¹⁾
Sistemi operativi supportati	Linux, Windows	Linux, Windows	Linux, Windows	Windows	Linux, Windows
Lingua	Inglese ⁽²⁾	Inglese ⁽²⁾	Inglese ⁽³⁾	Italiano (multilingua)	Italiano (multilingua)
Limite processori ammessi	illimitato	illimitato	illimitato	1 CPU (anche multi-core)	1 CPU (o un dual-core)
Limite memoria RAM indirizzabile	illimitata	illimitata	illimitata	1 GB	1 GB
Limite massimo istanze server aperte	illimitate	illimitate	illimitate	50	1
Limite delle dimensioni dei dati utente su disco	illimitato (oltre 16 TB)	illimitato (oltre 16 TB)	illimitato (oltre 16 TB)	4 GB	4 GB
Limite utenti concorrenti	illimitato	illimitato	illimitato	illimitato	illimitato
⁽¹⁾ Si rimanda alla successiva tabella delle licenze di SS e OR. ⁽²⁾ I tool di amministrazione (IBOConsole) sono scaricabili in versione italiana. ⁽³⁾ Installazione in italiano.					

Tab. VI

Prezzi medi delle licenze di SS e OR a giugno 2006				
Descrizione versione o componente aggiuntiva	SS		OR	
	Componente	Prezzo	Componente	Prezzo
Express	Express Edition	\$0 - gratuita	Express Edition	\$0 - gratuita
Workgroup	Workgroup Edition con 5 CAL	\$3.899	inesistente	
	CAL aggiuntivo	\$146		
Standard	Standard Edition con 5 CAL	\$1.849	Standard Edition	\$15.000
	CAL aggiuntivo	\$162	CAL aggiuntivo	\$250
Enterprise	Enterprise Edition con 25 CAL	\$24.999	Enterprise Edition	\$40.000
	CAL aggiuntivo	\$162	CAL aggiuntivo	\$250
	Strumenti di gestione	Incluso	Management Pack	\$6.000
	Protezione avanzata	Incluso	Protezione avanzata	\$10.000
	Business Intelligence	Incluso	Business Intelligence	\$40.000
	Istanza di database aggiuntiva	Incluso	Istanza di database aggiuntiva	\$96.000
	Core aggiuntivo (processore multi-core)	Incluso	Core aggiuntivo (processore dual-core)	\$192.000

Tab. VII

6. Progetto operativo per l'eventuale transizione del database

La possibilità di passare ad uno dei quattro RDBMS esaminati, escluso Firebird 1.5, può essere tenuta in considerazione alla luce dei risultati emersi dallo studio. In particolare, dei miglioramenti effettivi, seppur lievi, sembrano avvenire con il passaggio di versione a Firebird 2.0. In seguito saranno esaminati, in maniera schematica, i percorsi necessari per compiere gli adeguamenti di sistema necessari alla migrazione dei dati, sia per il passaggio da Firebird 1.5 ad un'altra release dello stesso, sia per il passaggio da Firebird 1.5 a SQL Server Express oppure Oracle Database XE. Infatti, sono da valutare tutte le soluzioni, poiché al cliente finale che utilizza EASY, può interessare il cambio di RDBMS per motivi quali l'incremento prestazionale, o semplicemente il passaggio ad un sistema che lo stesso cliente considera più affidabile o che, non da ultimo, si rende necessario per aumentare la compatibilità e la portabilità di altri componenti che già sono utilizzati nella sua azienda.

6.1 Valutazione dell'infrastruttura esistente

Dopo aver sottoposto il precedente studio al cliente e con questi concordato la transizione del sistema ad uno degli altri candidati, il primo elemento da valutare è l'infrastruttura già esistente presso l'azienda del cliente. In particolare, si dovranno considerare:

- la determinazione delle capacità e dell'utilizzo del server
- la documentazione dei processi di assistenza e gestione
- l'inventariazione dell'infrastruttura server

- la determinazione del costo di gestione e di manutenzione della struttura.

6.2 Individuazione degli obiettivi del consolidamento del server

Il passo successivo alla disamina della struttura già esistente, è l'eventualità di rafforzare ed espandere la tecnologia del server in base alle nuove esigenze che spingono il cliente a cambiare DMBS. Dovranno, quindi, essere analizzati:

- le priorità aziendali e tecnologiche dell'azienda
- il bilanciamento delle priorità, quali la mutazione dei costi e della flessibilità dell'infrastruttura
- i requisiti di capacità ideali del nuovo ambiente
- gli obiettivi di costo per la gestione e la manutenzione dell'infrastruttura consolidata
- la pianificazione del budget per il progetto di consolidamento.

6.3 Progettazione del nuovo ambiente

Una volta definiti i precedenti punti, è necessario predisporre un prospetto di ciò che dovrà essere la realtà fisica del nuovo ambiente. Ad esempio, nel caso di un semplice passaggio di versione di Firebird alla 1.0 oppure alla 2.0, le risorse hardware e software esistenti sono più che sufficienti, poiché i requisiti di questi RDBMS sono identiche a quelli di Firebird 1.5. Un eventuale migrazione a SQL Server Express deve, invece, tener conto dell'acquisto obbligatorio, se non è già in possesso, di licenze per l'utilizzo di Microsoft Windows XP per il

server e per ogni client, mentre anche in questo caso il lato hardware resterebbe pressoché invariato. Proprio quest'ultimo, invece, non è affatto trascurabile per quanto riguarda la migrazione ad Oracle Database XE, poiché, come esaminato nel secondo capitolo, l'architettura di questo RDBMS richiede, per aumentare le prestazioni e la sicurezza dei dati, che Data, Redo Log e Control file risiedano su dispositivi fisici diversi.

Allo scopo di considerare le varie problematiche che può far sorgere la scelta di un RDBMS rispetto ad un altro, si rende necessario:

- analizzare le opzioni relative all'hardware e al software
- selezionare un'infrastruttura in base alle esigenze di capacità e di crescita aziendali
- predisporre un progetto in vista del *failover*, ossia la capacità di passaggio automatico ad una macchina server ridondante o in standby in caso di grave inconsistenza del database
- valutare l'impatto aziendale di ciascuna alternativa di consolidamento
- individuare i ruoli e le responsabilità organizzative durante e dopo il consolidamento.

6.4 Implementazione di un nuovo ambiente pilota

Dopo aver selezionato il progetto che meglio di tutti si adatta alle nuove esigenze aziendali, esso deve essere realizzato in termini pratici, e in un primo momento deve essere testato l'effettivo funzionamento della nuova struttura. Ciò avviene:

- analizzando l'architettura dell'infrastruttura aziendale e la specifica architettura di rete

- individuando eventuali limiti tecnici e logistici, e i rischi annessi
- realizzando e testando il nuovo ambiente consolidato.

6.5 Finalizzazione del piano di migrazione di dati e utenti

È il passo più delicato dopo il consolidamento della struttura del server. Come analizzato nel terzo e quarto capitolo, il passaggio di RDBMS che avviene all'interno della famiglia Firebird è relativamente semplice e non necessita di particolari stravolgimenti sia a livello di database esistente che di adattamento del gestionale. Questo, assolutamente, non si può dire nel caso di una migrazione verso SQL Server Express, oppure Oracle Database XE, per la quale sono indispensabili i seguenti accorgimenti:

- stesura di una procedura per il trasferimento di dati e utenti, sia con il tool di trasferimento descritto nel terzo capitolo, sia con gli strumenti che i due RDBMS mettono a disposizione
- realizzazione di una pianificazione dettagliata della distribuzione, comprendente i piani di emergenza
- sviluppo e documentazione delle procedure di manutenzione e di gestione successive al consolidamento
- verifica della disponibilità dei backup e dei piani di emergenza prima di procedere alla migrazione
- realizzazione della migrazione secondo la pianificazione dettagliata
- test dell'ambiente consolidato, comprensivo a questo punto di dati e utenti
- passaggio al nuovo ambiente di produzione.

6.6 Analisi e revisione del progetto

La parte finale del progetto di transazione, riguarda la correzione di tutte le imperfezioni strutturali e procedurali che si sono riscontrate nelle precedenti fasi di consolidamento ambientale e migrazione fisica del database. È importante approntare l'analisi dei risultati del progetto di consolidamento, inclusi i costi e le procedure di manutenzione, e dell'effettivo raggiungimento degli obiettivi preposti con il cambio di RDBMS. Conseguentemente, è utile programmare una revisione periodica e ottimizzazione dell'ambiente nella sua totalità, sia per quanto riguarda il lato hardware, sia per quanto riguarda quello software.

7. Conclusioni

Oltre alle osservazioni relative alla comparazione dei cinque DBMS, già menzionate nel quinto capitolo, è necessario tener conto anche di altri aspetti che non siano meramente legati alle singole prestazioni.

Alla luce dei risultati ottenuti con questo studio, è importante per la software house valutare attentamente le richieste dei clienti che intendono migrare i dati contenuti nei propri database gestiti da Firebird 1.5, e le proposte che a questi possono essere rivolte dalla Easy Soft stessa. Nel qual caso avvengano implementazioni, nel gestionale, di uno o più RDBMS, sarà necessario dedicare molta attenzione alla revisione dello schema ER del database di EASY e delle query che il gestionale esegue durante il suo utilizzo. Infatti, questi elementi sono fortemente ottimizzati per l'architettura di Firebird 1.5, ma non altrettanto si può dire per le altre architetture, in particolare per quelle di SQL Server Express e di Oracle Database XE.

Durante le fasi dello studio, infatti, è apparso evidente che gli indici definiti dai progettisti dei database non vengono quasi mai utilizzati nel piano di query di questi due RDBMS, che seguono, invece, gli indici creati automaticamente sulle chiavi primarie ed esterne. Ciò rallenta, in modo consistente specialmente in Oracle Database XE, la quasi totalità delle query del gestionale, che utilizzano la sintassi LEFT OUTER JOIN nella clausola FROM.

La decisione di intraprendere un'eventuale migrazione del database verso qualcuno dei nuovi motori di database esaminati, perciò, sia dal punto di vista della stessa Easy Soft, sia dal punto di vista del cliente finale, deve tener conto di diversi aspetti. In generale, oltre alla precedentemente citata necessità di ottimizzazione della struttura del gestionale, sono da considerare rilevanti ai fini decisionali:

- l'aumento della complessità di programmazione e aggiornamento, in fase di sviluppo, del gestionale EASY, dovuto alla convivenza presso l'azienda costruttrice di più DBMS
- i costi di progettazione e implementazione della transizione, descritti nel capitolo precedente
- le limitazioni della specifica versione di ciascun RDBMS esaminato e i costi delle licenze dei prodotti non open-source per acquisire nuove versioni prive di limitazioni, riportati nel quinto capitolo.

Bibliografia

- Susi Dulli, Vittorio Favero, “Modelli e Strutture per il Data Warehousing”, Diade, 2000, cap. 1, 3, 4.
- Giuseppe Callegarin, “Nuovo Corso di Informatica – Basi di Dati e Sistemi Informativi”, Cedam, 2000, cap. 6 – 7.
- Helen Borrie, “The Firebird Book: A Reference for Database Developers”, Apress, 2004, cap. xxiii, 36.
- AA.VV., “Firebird Version 1 Release Notes”, <http://www.ibphoenix.com>, marzo 2002.
- Helen Borrie, “Firebird 1.5.3 Release Notes”, <http://www.ibphoenix.com>, gennaio 2005.
- Helen Borrie, “Firebird 2.0 Release Notes”, <http://www.ibphoenix.com/>, agosto 2006.
- Gorge Rajesh, “Panoramica su SQL Server 2005 Express Edition”, <http://www.visual-basic.it>, novembre 2005.
- Nicola Boni, “Struttura logica e fisica di Oracle9i”, <http://www.programmazione.it>, giugno 2002.
- Nicola Boni, “La gestione della memoria in Oracle9i”, <http://www.programmazione.it>, luglio 2002.
- AA.VV, “Oracle Database Documentation Library 10g Release 2”, <http://www.oracle.com>, giugno 2005, cap. 4 – 7.
- Edison Group Inc., “Comparative Management Cost Study of Oracle Database 10g and Microsoft SQL Server 2005”, <http://www.oracle.com>, marzo 2006.

Ringraziamenti

In questi ultimi mesi ho dedicato anima e corpo al lavoro che ha portato alla redazione di questo scritto, che a sua volta pone termine ad un lungo periodo della mia vita, iniziato alla tenera età di (quasi) sei anni. Credetemi, c'è da commuoversi, ora, pensando a quante cose sono accadute da allora, talvolta stupende, talvolta terribili, a quante persone hanno partecipato a quell'infinito banchetto di anime che sono stati i miei primi venticinque anni di vita. Alcuni buoni commensali siedono intorno ancora affamati, molti fortunatamente sono stati solo di passaggio, alcuni purtroppo se ne sono andati via per sempre, ma il loro ricordo rimarrà indelebile e immutato. Voglio ringraziare, e sono tutti "grazie" che provengono anche dalla più recondita profondità del mio cuore, tutte le persone che mi hanno seguito, consolato e stimolato a dare il meglio di me in questi anni di studio, e ultimi mesi di lavoro intenso, che hanno plasmato la mia persona.

Grazie ai miei genitori, Eleonora e Vinicio, ed a tutti i miei nonni, anche chi non c'è più, per essermi stati vicini dalla notte dei tempi, ognuno a modo suo e nonostante i comprensibili incidenti di percorso, sempre col proprio amore, ne sono certo. L'unica ricompensa ai loro sforzi è questa laurea, sicuramente troppo poco, ma conquistata anche per loro.

Grazie a chi questa vita me l'ha fatta ammirare sotto un'altra luce, non mi ha fatto perdere il filo della ragione durante più di qualche forte momento di debolezza della mia esistenza, e che ha saputo conquistare il mio eterno amore per lei: Roberta, l'unica stupenda donna della mia vita, il cui solo pensiero mi emoziona fortemente. Ti amo.

Grazie a Enrico, Daniele detto Sebamed, a mio fratello Aurelio nonostante i contrasti, Debe, Simone, perché con loro mi sono confrontato, anche se non

sempre le cose sono andate bene, e con loro ho passato gli anni proverbialmente più duri, quelli dell'adolescenza: solo loro posso considerare miei veri amici.

Grazie ai compari di università Ale e Fede, che si aggiungono ai precedenti, mitici anche come compagni di bevute, soprattutto il secondo, e sollazzi, forse troppi; ringrazio quindi anche baretto Zanellato che ci ha ospitato oltre misura; a Checco per essere sé stesso; ad Ale Settembrini per aver condiviso i tempi duri di Informatica.

Grazie a Roberto per aver condotto me e Ale a terminare i progetti di programmazione, e per avermi insegnato quel poco che so sui database; ad Elisa ed Illari (mia tutor!) per avermi aiutato con la Statistica, gli appunti e tutto quello che perdevo in giro.

Grazie al Calcio Padova 1910 per avermi trasmesso una passione, il soccer, di avermi fatto assaporare la serie A almeno una volta nella vita, avermi regalato due indimenticabili spareggi.

Infine, un grazie di cuore ai personaggi della Easy Soft: a Luca, grande professionista, colui che ha gettato le basi del lavoro e mi ha introdotto con pazienza in un mondo sconosciuto; a Max, detto il "Misericordioso" per aver guidato ciò che Luca ha lasciato in eredità, cioè me; ad Antonio, per essersi dimostrato il pagliaccio intellettuale che mancava, anche in pausa pranzo; a Cristina e Laura, per aver sopportato me, Antonio e i clienti; a Sonia per la sua simpatia; a Gloria per avermi registrato tutte le ore di presenza; ad Alessandro ed Andrea del laboratorio per averci sempre aiutato con internet!