Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria di Modena

Corso di Laurea in Ingegneria Informatica

CONFRONTO DEI DBMS RELAZIONALI SQLSERVER E MYSQL

Relatore:
Prof. Sonia Bergamaschi

Candidato:
Luca Sculco

Correlatore:
Ing. Maurizio Vincini

Anno Accademico 2005/2006

INDICE DEGLI ARGOMENTI TRATTATI:

INTRODUZIONE

Un Data Base (traducibile in italiano come "base di dati") non è un altro che un insieme di dati logicamente correlati fra loro.
I Data Base Management System (DBMS) sono quindi i prodotti software in grado di gestire i database e le loro caratteristiche sono:
-consentire l'accesso ai dati attraverso uno schema concettuale, invece che attraverso uno schema fisico
-capacità di gestire grandi quantità di dati
-condivisione dei dati fra più utenti e applicazioni
-utilizzo di sistemi di protezione e autorizzazione per l'accesso ai dati stessi

Possiamo identificare diversi tipi di database, in base alla loro struttura logica:
-database gerarchici
-database reticolari
-database relazionali
-database ad oggetti

I database di tipo relazionale sono, attualmente, di gran lunga i più diffusi e per questo abbiamo deciso di confrontare il migliore DBMS open source, MySQL versione 5.0.22, con il miglior prodotto Microsoft per la gestione dei database SQLServer, versione 2000.
Il confronto si baserà sull’analisi dei tempi di risposta a diversi tipi di query che verranno effettuate in uguale contesto su entrambi i dbms.

1 – SQL E IL MODELLO RELAZIONALE:

1.1 – La storia dell'SQL:

Con l'arrivo del modello relazionale fu introdotto anche l' SQL (Structured Query Language) il linguaggio standard per la definizione, manipolazione e interrogazione delle basi di dati relazionali; consente di operare sui dati tramite frasi che contengono parole chiave prese dal linguaggio corrente (ovviamente della lingua inglese!). La storia di SQL (che si pronuncia facendo lo spelling inglese delle lettere che lo compongono, e quindi "ess-chiu-el" e non "siquel" come si sente spesso) inizia nel 1974 con la definizione da parte di Donald Chamberlin e di altre persone che lavoravano presso i laboratori di ricerca dell'IBM di un linguaggio per la specificazione delle caratteristiche dei database che adottavano il modello relazionale. Questo linguaggio si chiamava SEQUEL (Structured English Query Language) e venne implementato in un prototipo chiamato SEQUEL-XRM fra il 1974 e il 1975. Le sperimentazioni con tale prototipo portarono fra il 1976 ed il 1977 ad una revisione del linguaggio (SEQUEL/2), che in seguito cambiò nome per motivi legali, diventando SQL. Il prototipo (System R) basato su questo linguaggio venne adottato ed utilizzato internamente da IBM e da alcuni sui clienti scelti. Grazie al successo di questo sistema, che non era ancora commercializzato, anche altre compagnie iniziarono a sviluppare i loro prodotti relazionali basati su SQL. A partire dal 1981 IBM cominciò a rilasciare i suoi prodotti relazionali e nel 1983 cominciò a vendere DB2. Nel corso degli anni ottanta numerose compagnie (ad esempio Oracle e Sybase, solo per citarne alcuni) commercializzarono prodotti basati su SQL, che divenne lo standard industriale di fatto per quanto riguarda i database relazionali.
Nel 1986 l'ANSI (American National Standards Institute) adottò SQL (sostanzialmente adottò il dialetto SQL di IBM) come standard per i linguaggi relazionali con il quale obbligò tutte le versioni a supportare le stesse parole chiave (come SELECT, UPDATE, DELETE, INSERT, WHERE) e a far sì che queste producessero i medesimi risultati e nel 1987 esso diventò anche standard ISO. Questa versione dello standard va sotto il nome di SQL-86. Negli anni successivi esso ha subito varie revisioni che hanno portato prima alla versione SQL-89, successivamente alla SQL-92, alla SQL-99 e alla attuale SQL-2003.
Tale processo di standardizzazione mirava alla creazione di un linguaggio che funzionasse su tutti i DBMS relazionali, ma purtroppo questo obiettivo non fu raggiunto. Dal punto di vista pratico le cose andarono differentemente: infatti, i vari produttori implementarono il linguaggio con numerose variazioni e adottarono gli standard ad un livello non superiore al minimo, definito dall'Ansi come Entry Level., estendendolo in maniera proprietaria a seconda della propria visione del mondo dei database.
La versione di SQL che adotteremo per le nostre interrogazioni è quella del 1992 (SQL 92).
Come scritto poco sopra, SQL viene utilizzato per compiere interrogazioni su dati di tipo relazionale: vediamo ora in dettaglio questo modello di dati, introdotto nel 1970 da Codd. [1]

1.2 – Il modello relazionale:

I database relazionali sono il tipo di database attualmente più diffuso. I motivi di questo successo sono fondamentalmente due:

1. forniscono sistemi semplici ed efficienti per rappresentare e manipolare i dati
2. si basano su un modello, quello relazionale, con solide basi teoriche

Il modello relazionale è stato proposto originariamente da Edgar F. Codd in un famoso articolo del 1970 “A relational model of data for large shared data banks” (Un modello relazionale di dati per gestire grandi banche dati condivise). Grazie alla sua coerenza ed usabilità, il modello è diventato negli anni '80 quello più utilizzato per la produzione di DBMS.
La struttura fondamentale del modello relazionale è appunto la "relazione", cioè una tabella bidimensionale costituita da righe (tuple) e colonne (attributi). Le relazioni rappresentano le entità che si ritiene essere interessanti nel database. Ogni istanza dell'entità troverà posto in una tupla della relazione, mentre gli attributi della relazione rappresenteranno le proprietà dell'entità. Ad esempio, se nel database si dovranno rappresentare delle persone, si potrà definire una relazione chiamata "Persone", i cui attributi descrivono le caratteristiche delle persone (Figura 2). Ciascuna tupla della relazione "Persone" rappresenterà una particolare persona.

Persone

nome	cognome	data_nascita	sesso	stato_civile
Mario	Rossi	29/03/1965	M	Coniugato
Giuseppe	Russo	15/11/1974	M	Celibe
Alessandra	Mondella	13/06/1970	F	Nubile

In realtà, volendo essere rigorosi, una relazione è solo la definizione della struttura della tabella, cioè il suo nome e l'elenco degli attributi che la compongono. Quando essa viene popolata con delle tuple, si parla di "istanza di relazione". Perciò la precedente Figura 2 rappresenta un'istanza della relazione persona. Una rappresentazione della definizione di tale relazione potrebbe essere la seguente:

Persone (nome, cognome, data_nascita, sesso, stato_civile)

Nel seguito si indicheranno entrambe (relazione ed istanza di relazione) con il termine "relazione", a meno che non sia chiaro dal contesto a quale accezione ci si riferisce.
Le tuple in una relazione sono un insieme nel senso matematico del termine, cioè una collezione non ordinata di elementi differenti. Per distinguere una tupla da un'altra si ricorre al concetto di "chiave primaria", cioè ad un insieme di attributi che permettono di identificare univocamente una tupla in una relazione. Naturalmente in una relazione possono esserci più combinazioni di attributi che permettono di identificare univocamente una tupla ("chiavi candidate"), ma fra queste ne verrà scelta una sola da utilizzare come chiave primaria. Gli attributi della chiave primaria non possono assumere il valore null (che significa un valore

non determinato), in quanto non permetterebbero più di identificare una particolare tupla in una relazione. Questa proprietà delle relazioni e delle loro chiavi primarie va sotto il nome di integrità delle entità (entity integrity).
Ogni attributo di una relazione è caratterizzato da un nome e da un dominio. Il dominio indica quali valori possono essere assunti da una colonna della relazione. Spesso un dominio viene definito attraverso la dichiarazione di un tipo per l'attributo (ad esempio dicendo che e' una stringa di dieci caratteri), ma e' anche possibile definire domini più complessi e precisi. Ad esempio per l'attributo "sesso" della nostra relazione "Persone" possiamo definire un dominio per cui gli unici valori validi sono 'M' e 'F'; oppure per l'attributo "data_nascita" potremmo definire un dominio per cui vengono considerate valide solo le date di nascita dopo il primo gennaio del 1960, se nel nostro database non e' previsto che ci siano persone con data di nascita antecedente a quella. Il DBMS si occuperà di controllare che negli attributi delle relazioni vengano inseriti solo i valori permessi dai loro domini. Caratteristica fondamentale dei domini di un database relazionale è che siano "atomici", cioè che i valori contenuti nelle colonne non possano essere separati in valori di domini più semplici. Più formalmente si dice che non è possibile avere attributi multivalore (multivalued). Ad esempio, se una caratteristica delle persone nel nostro database fosse anche quella di avere uno o più figli, non sarebbe possibile scrivere la relazione Persone nel seguente modo:

Persone (nome, cognome, data_nascita, sesso, stato_civile, figli)

Infatti l'attributo figli è un attributo non-atomico, sia perchè una persona può avere più di un figlio, sia perchè ogni figlio avrà varie caratteristiche che lo descrivono. Per rappresentare queste entità in un database relazionale bisogna definire due relazioni:

Persone(*numero_persona, nome, cognome, data_nascita, sesso, stato_civile)
Figli(*numero_persona, *nome_cognome, eta, sesso)

Nelle precedenti relazioni gli asterischi (*) indicano gli attributi che compongono le loro chiavi primarie. Si noti l'introduzione nella relazione Persone dell'attributo numero_persona, attraverso il quale si assegna a ciascuna persona un identificativo numerico univoco che viene utilizzato come chiave primaria. Queste relazioni contengono solo attributi atomici. Se una persona ha più di un figlio, essi saranno rappresentati in tuple differenti della relazione Figli. Le varie caratteristiche dei figli sono rappresentate dagli attributi della relazione Figli. Il legame fra le due relazioni è costituito dagli attributi numero_persona che compaiono in entrambe le relazioni e che permettono di assegnare ciascuna tupla della relazione figli ad una particolare tupla della relazione Persone. Più formalmente si dice che l'attributo numero_persona della relazione Figli è una chiave esterna (foreign key) verso la relazione Persone. Una chiave esterna è una combinazione di attributi di una relazione che sono chiave primaria per un'altra relazione. Una caratteristica fondamentale dei valori presenti in una chiave esterna è che, a meno che non siano null, devono corrispondere a valori esistenti nella chiave primaria della relazione a cui si riferiscono. Nel nostro esempio ciò significa che non può esistere nella relazione Figli una tupla con un valore dell'attributo numero_persona, senza che anche nella relazione Persone esista una tupla con lo stesso valore per la sua chiave primaria. Questa proprietà va sotto il nome di integrità referenziale (referential integrity).
Abbiamo così introdotto brevemente il modello relazionale analizzando anche le proprietà di base e fondamentali che servono per la creazione di relazioni. [2]

1.2.1 - I vantaggi del modello relazionale:

Uno dei grandi vantaggi del modello relazionale è che esso definisce anche un algebra, chiamata appunto "algebra relazionale". Tutte le manipolazioni possibili sulle relazioni sono ottenibili grazie ala combinazione di cinque soli operatori: RESTRICT, PROJECT, TIMES, UNION e MINUS. Per comodità sono stati anche definiti tre operatori addizionali che comunque possono essere ottenuti applicando i soli cinque operatori fondamentali: JOIN, INTERSECT e DIVIDE. Gli operatori relazionali ricevono come argomento una relazione o un insieme di relazioni e restituiscono una singola relazione come risultato.
Vediamo brevemente questi otto operatori:

RESTRICT: restituisce una relazione contenente un sottoinsieme delle tuple della relazione a cui viene applicato. Gli attributi rimangono gli stessi.

PROJECT: restituisce una relazione con un sottoinsieme degli attributi della relazione a cui viene applicato. Le tuple della relazione risultato vengono composte dalle tuple della relazione originale in modo che continuino ad essere un insieme in senso matematico.

TIME: viene applicato a due relazioni ed effettua il prodotto cartesiano delle tuple. Ogni tupla della prima relazione viene concatenata con ogni tupla della seconda.

JOIN: vengono concatenate le tuple di due relazioni in base al valore di un insieme dei loro attibuti.

UNION: applicando questo operatore a due relazioni compatibili, se ne ottiene una contenente le tuple di entrambe le relazioni. Due relazioni sono compatibili se hanno lo stesso numero di attributi e gli attributi corrispondenti nelle due relazioni hanno lo stesso dominio.

MINUS: applicato a due relazioni compatibili, ne restituisce una terza contenente le tuple che si trovano solo nella prima relazione.

INTERSECT: applicato a due relazioni compatibili, restituisce una relazione contenente le tuple che esistono in entrambe le relazioni.

DIVIDE: applicato a due relazioni che abbiano degli attributi comuni, ne restituisce una terza contenente tutte le tuple della prima relazione che possono essere fatte corrispondere a tutti i valori della seconda relazione.

I database relazionali compiono tutte le operazioni sulle tabelle utilizzando l'algebra relazionale, anche se normalmente non permettono all'utente di utilizzarla. L'utente interagisce con il database attraverso un'interfaccia differente, il linguaggio SQL, che permette di descrivere insiemi di dati. Le istruzioni SQL vengono scomposte dal DBMS in una serie di operazioni relazionali.

1.2.2 - RDBMS:

Naturalmente, visto l'ampio successo dei database relazionali, sono molti gli RDBMS (Relational Data Base Management System) presenti sul mercato: tra i principali, citiamo IBM DB2, Oracle, Microsoft SQL Server, Sybase, Filemaker Pro, Microsoft Access, Informix, PostgreSQL, SQLite, oltre naturalmente a MySql. Alcuni di questi sono software proprietari, mentre altri fanno parte della categoria open source: questi ultimi, fra quelli citati, sono MySql, PostgreSQL ed SQLite.
Ovviamente ogni software ha le sue caratteristiche, e la grande diffusione dei DB relazionali portò l'inventore del modello a definire, nel 1985, una serie di regole ("le 12 regole" di Codd) alle quali un DBMS dovrebbe attenersi per potersi considerare Relazionale. Queste regole sono state fissate nell'articolo "Is Your DBMS Really Relational?" pubblicato sul magazine ComputerWorld nel 1985 (vedi http://www.itworld.com/nl/db_mgr/05072001/pf_index.html). Codd scrisse questi criteri nell'ambito di un'iniziativa che serviva ad evitare che la sua definizione di database relazionale fosse resa meno restrittiva quando, nei primi anni ottanta, i venditori di database si affannarono per rimpiazzare i vecchi prodotti con un prodotto pseudo-relazionale. Interpretando rigidamente queste regole, sono ben pochi (o forse nessuno?!) i sistemi che potrebbero propriamente fregiarsi di questo titolo.
Andiamo adesso ad introdurre i 2 DBMS che saranno messi a confronto.

2 - SQLSERVER:

2.1 – Descrizione:

Per riuscire a capire il ruolo primario che SQL Server ricopre nel panorama mondiale dei software per la gestione dei database relazionali è utile conoscerne la storia e le tappe evolutive.

-Le origini di SQL Server
Nel 1987 Microsoft e Sybase stipularono un accordo per lo sviluppo e la commercializzazione di un software per la gestione di database distribuibile su diverse piattaforme hardware (mainframe, minicomputer e personal computer) e software (UNIX,VMS e OS/2, Windows, MS-DOS).
Il nucleo di partenza del progetto fondava le sue basi sul codice sorgente di "Sybase Dataserver", disponibile allora solo per sistemi UNIX, il quale presentava funzionalità avanzate come trigger e stored procedure ed una concezione fortemente orientata all'elaborazione client/server.
Al contrario, in ambito commerciale comparivano difficoltà legate in gran parte alla mancanza di un mercato di riferimento a cui puntare. Per colmare il gap Microsoft decise di appoggiarsi al leader del mercato DBMS di allora, Ashton-Tate produttrice del popolare dBase. La collaborazione tra Microsoft/Ashton-Tate/Sybase fu proficua e nel 1988 venne annunciato il rilascio di Ashton-Tate/Microsoft SQL Server il primo software per personal computer capace di rendere il database relazionale più accessibile agli utenti.
Nel 1989 Ashton-Tate terminò la collaborazione al progetto perché intuì che Microsoft intendeva abbandonare il porting di SQL Server su OS/2 per concentrarsi esclusivamente sul proprio futuro sistema operativo denominato Windows/NT. La collaborazione tra Microsoft e Sybase invece proseguì ancora per alcuni anni assicurando così la disponibilità del prodotto sia su sistemi UNIX che Windows e OS/2.
Nel 1993 fu rilasciata la versione 4.21 di SQL Server l'ultima targata Microsoft/Sybase, infatti l'anno successivo approfittando della scadenza dell'accordo di licenza siglato sette anni prima, Microsoft e Sybase terminarono la loro collaborazione escludendo un rinnovo dell'accordo.
Microsoft decise di correre da sola e come prima mossa acquista una copia del codice sorgente di SQL Server da Sybase (fino ad allora il codice era scambiato su nastri magnetici tra le sedi delle sue società) e utilizzando questo come punto di partenza cominciò a lavorare ad una propria versione del software tagliata su misura per Windows NT.
Allo scopo venne creato un team di sviluppo dedicato esclusivamente a SQL Server, capitanato inizialmente da Ron Soukup ed allargato poi negli anni successivi a personaggi del calibro di Hal Berenson ed altri guru del "data management" provenienti dal mondo accademico e da aziende concorrenti come Oracle, DEC e IBM.
Dal 1994 il codice di SQL Server venne in parte riscritto e ottimizzato per interagire al meglio con i sistemi di Windows NT, vennero aggiunti nuovi moduli e nuove API di accesso ai dati quali OLE DB e ADO.
Il lavoro continuo del team di sviluppo prese forma nel 1998 con il rilascio della versione 6.0/6.5. di SQL Server. Qui le tracce del codice sorgente acquistato anni prima da Sybase sono veramente esigue, nasce quindi un SQL Server totalmente "made in" Microsoft.

-Nome in codice Sphinx, SQL Server 7.0
Negli anni successivi il team di sviluppo continuò lo sviluppo della versione 6.5 e nel Gennaio 1999 uscì la versione 7.0 di SQL Server. Quest'ultima definì nuovi standard in termini di affidabilità, performance e scalabilità ed offrì ai nuovi e vecchi clienti un prodotto maturo, di classe superiore in grado di affrontare la sfida del gigante Oracle.
Grazie alla sua versatilità, semplicità d'uso (amministrazione semplificata tramite Enterprise Manager, pubblicazione guidata di report con Web Assistant, ecc…) e ai prezzi concorrenziali SQL Server 7.0 diventò uno dei software RDBMS più venduti. Il boom Internet di quegli anni non fece che consolidare la leadership di SQL Server, perché siti di commercio elettronico di grande successo come www.barnesandnoble.com lo scelsero come motore di database per le proprie applicazioni web.

-Nome in codice Shiloh, SQL Server 2000
Il lavoro del team di sviluppo non cessò e a solo un anno di distanza, nell'agosto del 2000, venne rilasciata una nuova versione: SQL Server 2000. Questa release fu molto importante, perché apportò modifiche sostanziali al motore di SQL Server (in parte riscritto) e gettò le basi di un'architettura solida e duratura che a parere del team di sviluppo avrebbe dovuto persistere per almeno un decennio (Service Pack permettendo).
Ecco alcune innovazioni chiave introdotte:

Supporto di memoria esteso (AWE) fino a 64GB
Supporto multiprocessore simmetrico (SMP)
Backup in linea
Funzioni Definite dall'utente
Viste Indicizzabili
Server Federati con Viste Partizionate distribuite (anche tra nodi di cluster) Caratteristica che pone SQL Server 2000 al top dei test TCP-C
Nuovi tipologie di Triggers (INSTEAD OF e AFTER)
Nuovi tipi dati
Rafforzamento dell'integrità referenziale
Istanze multiple denominate di SQL Server
Accesso ai dati via Web
Supporto XML
Navigazione e distribuzione dei files di LOG

-Nome in codice Yukon, SQL Server 2005
SQL Server 2005 è l'ultima versione rilasciata da Microsoft nel novembre 2005. È in grado di gestire basi dati di grandi dimensioni e d'integrarsi con numerose applicazioni e fonti dati diverse.
Comunque avendo raggiunto il principale obbiettivo di introdurre SLQ Server 2000, le innovazioni introdotte da quest'ultima versione, che sono tante e varie, non saranno oggetto di questa tesina. [3,4]

2.2 – SQLServer e lo standard SQL-92:

Come ne abbiamo già parlato ampiamente nel paragrafo 1.1, SQL è un linguaggio di alto livello molto semplice da utilizzare perchè permette di interrogare il database con "domande" in linguaggio naturale (SELECT ...FROM), ma possiede un limite: non è stato concepito per la programmazione.
La programmazione richiede funzionalità aggiuntive che vanno dalle istruzioni per il controllo del flusso alla modularità. Per ovviare a questo limite Microsoft ha creato per SQL Server il Transact-SQL.
T-SQL a differenza di SQL è un linguaggio procedurale. T-SQL non è standard come l'SQL anche se è conforme alle specifiche dell'ANSI-92 SQL. T-SQL è un linguaggio proprietario che possiamo utilizzare solo con Microsoft SQL Server. Non è supportato da altri database (come Oracle o MySQL).
Possiamo definire il Transact SQL come una collezione di estensioni capaci di aumentare le potenzialità dell'ANSI-SQL 92.
Oltre a questo Transact SQL aggiunge costrutti tipici della programmazione come istruzioni per il controllo del flusso (if e while per esempio), variabili, gestione degli errori, ecc... che ci mettono in grado di fare esercizio di programmazione dei databases.
Possiamo così creare con il T-SQL veri e propri programmi (le stored procedure e i triggers) sfruttando da un lato la potenza nell'interrogazione di basi dati delle istruzioni SQL e dall'altro funzioni e costrutti del T-SQL per la creazione della logica del programma.
T-SQL oltre ad ereditare i vantaggi di semplicità e immediatezza di SQL nell'interrogazione dei database include una vasta gamma di comandi e istruzioni di controllo che permettono all'utente di lavorare su ogni oggetto e su ogni informazione contenuta in SQL Server (tabelle, indici, login, jobs, alert, backup, ecc...). Per finire il T-SQL potremmo rappresentarlo con questa formula:

T-SQL = SQL + Estensioni Microsoft

2.3 – Calcolo del tempo di esecuzione:

Per il calcolo del tempo di risposta alle query, Microsoft mette a disposizione Profiler, un tool che permette di avere informazioni di diverso tipo sulle operazioni eseguite con Query Analyzer; una di queste è la voce “Duration” che appunto ci dice quanto ha impiegato il dbms ad eseguire i comandi, in millisecondi, che nel nostro caso saranno delle query. Per usare questo tool è molto semplice: basta aprire il programma e iniziare una nuova traccia che ci farà connettere al server selezionato; poi aperto Query Analyzer e connesso al server, non ci basta che eseguire le operazioni e andare a vedere i valori mostrati da Profiler.

3 - MYSQL:

3.1 – Descrizione:

Il MySql, gestore di database, viene sviluppato fin dal 1979 dalla TxC (Società di Consulenza svedese), adesso MySQL AB, desiderosa di migliorare la propria realtà aziendale, ma è solo dal 1996 che viene distribuita una versione che supporta SQL, utilizzando in parte codice di un altro prodotto: mSQL, che è un “piccolo” dbms che supporta le connessioni TCP/IP ed un piccolo sottoinsieme dell'SQL; molto semplice e veloce, dalla versione 3 può funzionare come mono-processo (per risparmiare memoria) o multiprocesso (per maggiori prestazioni). Originariamente sviluppato nel 1994, mSQL ha riempito il vuoto che esisteva tra i database embedded da desktop come Microsoft Access e i database commerciali di alto livello, come Oracle e DB2. Tra il 1994 e il 1997, è cresciuto in popolarità ed è divenuto la prima scelta dei programmatori Open Source; sebbene fosse il database Open Source più usato, mSQL di per sé non era una tecnologia aperta e dal 1996, quando lo sviluppo di mSQL ha iniziato a ristagnare, MySQL ha preso il suo posto. Dal 1999, MySQL è andato ben oltre mSQL in termini di popolarità, cosicché oggi la maggior parte dei programmatori nemmeno conoscono mSQL. Una volta sviluppato ed immesso sul mercato mondiale, il MySql ottenne un rapido ed immediato successo grazie a tre semplici motivi:

-Il mySql veniva distribuito gratuitamente per le organizzazioni non a scopo di lucro
-Il mySql presentava per gli utilizzatori finali un database davvero potente ed estremamente flessibile
-Il mySql si può considerare un database molto personalizzabile. Infatti il codice sorgente dello stesso veniva fornito unitamente al prodotto finito per poter esser modificato da personale esperto in base alle esigenze.

Le principali caratteristiche del mySql sono le seguenti:
-Centouno connessioni contemporanee. Questo dato non significa che sul nostro sito web, adottante mySql, possiamo avere massimo centoun visitatori. Questo indice numerico determina il numero di accessi massimi consentiti nel database nello stesso frangente di tempo.
-Estrema velocità di accesso ai dati contenuti al suo interno
-Elevata sicurezza nel rispetto dei dati contenuti al suo interno da parte di accessi indesiderati e non autorizzati
-Database multipiattaforma: con questo termine si indica la possibilità di usare (dopo una determinata installazione e configurazione) il mySql in diversi ambienti (ad esempio windows98, windows 2000 oppure linux)
-Disponibilità di personalizzare il codice sorgente.

Se teniamo presente le 12 regole descritte nel paragrafo 1.2.2, anche a MySql, soprattutto nelle versioni non recentissime, mancano alcune funzionalità come le viste e l'integrità referenziale, a causa delle quali si può sostenere che non sia del tutto corretto considerare MySql come un RDBMS; qualcuno infatti preferisce definirlo "un DBMS con supporto SQL".

Tuttavia la versione 5.0 di MySql, finalmente resa disponibile nell'ottobre 2005, introduce nuove funzionalità che vanno a colmare (anche se non ancora completamente) le lacune cui abbiamo accennato, e soprattutto mettono MySql sullo stesso piano dei DBMS concorrenti relativamente a caratteristiche avanzate come i triggers, le viste, le stored procedures, i cursori. Il linguaggio SQL di MySQL comprende numerose estensioni che sono tipiche di altri DBMS, quali PostgreSQL, Oracle e Sybase. In questo modo le query non standard scritte per altri DBMS in alcuni casi funzioneranno senza problemi.
In sostanza, quello che già era il più diffuso database open source diventa un vero e proprio DBMS di livello enterprise. [5,6]

3.2 – MySQL e lo standard SQL-92:

MySQL non e' mai stato molto vicino allo standard SQL ANSI, ma la versione 5.0 colma la maggioranza delle lacune. Il committment è di aderire allo standard se non vi sono svantaggi prestazionali!
La versione 5.0 aderisce in modo molto stretto allo standard ANSI comprese le recenti nuove funzionalità previste nell'ultimo rilascio SQL:2003. Da questo punto di vista MySQL risulta più conforme allo standard rispetto ad altri RDBMS commerciali.
E' anche possibile scegliere tra diversi *dialetti* SQL impostando SET sql_mode= ai diversi mode disponibili (molto comodi sono gli alias ANSI, ORACLE, TRADITIONAL, ...). In questo modo il controllo all'aderenza allo standard risulta semplificato. [6]

3.3 – Calcolo del tempo di esecuzione:

Per eseguire le query abbiamo usato la console di MySQL Server 5.0.22, che insieme ai risultati dell'interrogazione ci ritorna anche un valore, che indica il tempo di esecuzione in secondi. Oltre alla console è possibile usare un programma appartenente ai tools di MySQL, ovvero MySQL Query Browser che ci permette di eseguire query e ci mostra i tempi di esecuzione.

4 – ANALISI DELLE PRESTAZIONI DEI 2 DBMS:

4.1 – Descrizione delle tabelle utilizzate:

In entrambi i DBMS è stato creato un database con al suo interno 7 tabelle chiamate studente21k, studente84k, studente147k, studente210k, esame, corso, docente.
Le tabelle della famiglia STUDENTE hanno 5 attributi: matricola valore auto-incrementato, nome e cognome che sono sempre ‘aaa’ e ‘bbb’, città di provenienza e anno di corso; i record, 21mila in studente21k, 84mila in studente84k, 147mila in studente147k e 210mila in studente210k, sono stati inseriti con dei cicli effettuati in java e gli attributi sono stati settati in modo che le query eseguite per il calcolo dei tempi di risposta ritornassero il 20%, il 50% e l’80% di quelli totali.

```
create table studente (
        matricola int not null  auto_increment/identity,
        nome varchar(20) not null,
        cognome varchar(20) not null,
        città char(2),
        acorso int,
        Primary key(matricola)
)
```

La tabelle DOCENTE contiene 4 attributi: cod_docente valore auto-incrementato, nome e cognome sempre ‘ccc’e ’ddd’, città di provenienza. Il numero dei record è di 110 docenti, con città casuali.

```
create table docente (
        cod_docente int not null auto_increment/identity,
        nome varchar(20) not null,
        cognome varchar(20) not null,
        città char(2),
        Primary key(cod_docente)
)
```

La tabella che riguarda i CORSI è stata l’unica ad essere riempita con nomi reali, infatti rappresentano alcuni esami della facoltà di ingegneria informatica; il campo cod_corso è un valore auto-incrementato e il docente è casuale. E’ presente una foreign key sulla colonna docente, che si riferisce alla chiave primaria della tabella docente. In totale i record sono 72.

```
create table corso (
        cod_corso int not null auto_increment/identity,
        nome varchar(30) not null,
        docente int not null,
        Primary key(cod_corso),
```

```
        Foreign key(docente) references docente(cod_docente)
)
```

La tabella ESAME è stata riempita con 23196 record sempre con cicli in java e in modo assolutamente casuale; i suoi attributi sono: matricola foreign key sulla tabella studente, corso foreign key sulla tabella corso, data del giorno in cui è stato sostenuto l'esame e il voto; la primary key è data dalla coppia matricola,corso.

```
create table esame (
        matricola int not null,
        corso int not null,
        data date,
        voto int,
        Primary key(matricola,corso),
        Foreign key(matricola) references studente(matricola),
        Foreign key(corso) references corso(cod_corso)
)
```

4.1.1 – Connessione JDBC:ODBC

Come abbiamo appena detto sopra, per riempire le tabelle con le migliaia di record necessarie, abbiamo dovuto formulare dei cicli, che ripetevano -n- volte le operazioni di insert e popolando così la tabella velocemente; abbiamo deciso di fare ciò utilizzando Java e la sua possibilità di interfacciarsi con i nostri dbms grazie al driver jdbc:odbc che effettua un "ponte" tra le chiamate alle API JDBC (Java DataBase Connectivity) Java e le chiamate alle API ODBC. Tale driver sfrutta ODBC (Open DataBase Connectivity), uno standard Microsoft consolidato e diffuso che consente di astrarre dai protocolli specifici dei produttori di database, fornendo un'interfaccia di programmazione comune alle applicazioni dei client di database. Per fare ciò è sufficiente creare un DSN (Data Source Name) nel quale bisogna completare alcuni campi, come il nome del DSN, il database a cui ci vogliamo collegare, il nome utente e la password, dopodiché si scrive qualche riga di codice in java (nel nostro caso abbiamo usato JCreator) nel quale utilizziamo i metodi messi a disposizione dalle interfacce e dalle classi del package java.sql che servono per creare una connessione con il DBMS selezionato nel DSN.
Ad esempio il codice utilizzato per collegarci al DSN di nome "myodbc" è il seguente:

```
import java.sql.*;
public class MySql1 {
        public static void main(String[] args) {
                Connection con = null;
                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        String url = "jdbc:odbc:myodbc";
                        con = DriverManager.getConnection(url, "root", "*******");
                    }
        catch(Exception e) { System.out.println("errore di connessione");    }
```

Creata la connessione possiamo eseguire le operazioni di INSERT:

```
        try {
                Statement query = con.createStatement();
                int n = 0;
                while (n<1000) {
                        int a = query.executeUpdate("insert into
studente(nome,cognome,città,acorso) values ('aaa',’bbb’,’RE’,3);
                        n=n+1;  }
            } catch(Exception e) { System.out.println("errore nella query");
```

Questa 2° parte ci permette di inserire nella tabella studente 1000 record che differenziano solamente per la matricola, che qua non inseriamo perché definita IDENTITY e quindi si incrementa di 1 automaticamente per ogni record inserito.

4.2 – Set delle query utilizzate:

Per l’analisi dei tempi di risposta abbiamo interagito solo con le tabelle di tipo studente e come detto in precedenza abbiamo scelto 3 query che ci ritornano il 20%, il 50% e l’80% dei record, questo per poter studiare le prestazioni dei dbms a seconda della grandezza delle tabelle su cui eseguono i calcoli. Le altre tabelle saranno impiegate per le query con join che vedremo nel prossimo paragrafo.
La query n.1, che ritorna circa il 20% dei record è:

```
select * from studente
where (citta='MO' or citta='RE' or citta='BO' or citta='FI')
and (acorso=1 or acorso=2)
```

per studente21k abbiamo 4000 record
- studente84k 16000
- studente147k 28000
- studente210k 40000

La query n.2, che ritorna il 50% dei record è:

```
select * from studente
where acorso=1 or acorso=2 or acorso=3
```

per studente21k abbiamo 10500 record
- studente84k 42000
- studente147k 73500
- studente210k 105000

La query n.3, che ritorna circa l'80% dei record è:

```
select * from studente
where citta is not null and (citta <> 'MO' or acorso <> 2)
```

per studente21k abbiamo 17000 record
- studente84k 68000
- studente147k 119000
- studente210k 170000

4.2.1 – Query con join:

Oltre alle query appena descritte, abbiamo voluto testare i tempi di risposta dei 2 dbms sottoponendoli all'esecuzione di query con join, cioè query su più tabelle intersecate; con questa operazione le tabelle coinvolte vengono unite attraverso un attributo che hanno in comune e che andiamo ad inserire nella clausola di join.
Le operazioni di join usate nelle query sono quelle fondamentali: inner join, right join, left join e full join e si passa dal join a 2 vie(2 tabelle) a quello a 4 vie(4 tabelle).
Per i join a 2 vie usiamo le tabelle studente210k e esame, per quelli a 3 vie aggiungiamo la tabella corso e per quello a 4 vie la tabella docente.
Abbiamo un problema quando dobbiamo fare il full join in MySQL, perché non è supportato, quindi dobbiamo fare il right join unito al left join.
La query base da cui partiamo è quella che interseca le tabelle studente210k e esame attraverso la foreign key matricola di esame:

```
select s.matricola, e.voto,
from esame as e join studente210k as s on e.matricola=s.matricola
```

poi aggiungendo al join le tabelle corso e docente, con clausola di join sempre sulle foreign key, abbiamo:

```
select s.matricola, e.voto, c.cod_corso, d.cod_docente
from docente as d join corso as c on d.cod_docente=c.docente
                        join esame as e on c.cod_corso=e.corso
                        join studente210k as s on e.matricola=s.matricola
```

Le altre query con gli altri join saranno viste direttamente nel capitolo dei risultati.

4.2.2 – Inserimento degli indici:

Le tabelle su cui effettuiamo i join vengono ora indicizzate per analizzare i tempi di esecuzione e confrontarli tra i 2 dbms, ma per fare questo abbiamo bisogno di nuove query da eseguire, che introdurremo direttamente nel paragrafo dei risultati.
Per quanto riguarda l'indicizzazione delle tabelle da notare che su ogni primary key viene creato dal dbms un indice, chiamato clustered in SQLServer e primary in MySQL, unico per tabella. Oltre a questo indice noi andremo ad aggiungerne altri, unclustered (SQLServer) o index (MySQL), sulle colonne che riterremo più adatte per ottimizzare il tempo di esecuzione delle query. Tutti questi indici sono di tipo B-Tree.

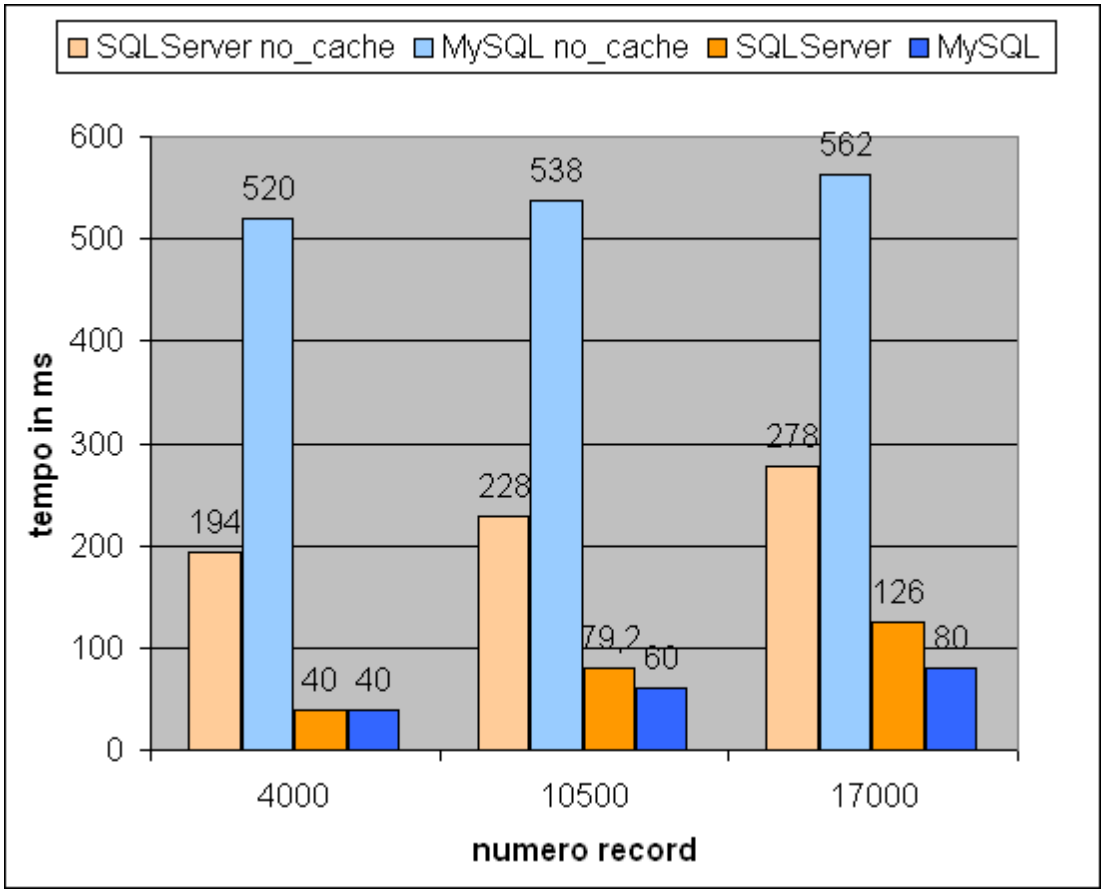
4.3 – Risultati:

Mostreremo gli esiti del confronto tra i 2 dbms a partire dalle query su singola tabella per poi passare ai join ed infine ai join con le tabelle indicizzate. In tutte e 3 le tipologie di analisi è da sottolineare il fatto che i tempi di esecuzione sono divisi in 2 categorie chiamate "con cache" e "senza cache"; la differenza tra le due sta nel fatto che un dbms quando esegue una query tiene in memoria, nella sua query-cache, dei dati che riguardano le interrogazioni effettuate, tra cui, oltre a record veri e propri, anche informazioni su come accedere ai dati del database sul disco fisso, ad esempio il piano di esecuzione.
Questo perchè quando il dbms esegue la query controlla se ha delle informazioni che gli possono essere utili nella query cache e solo se non le trova se le calcola andando a prendere i dati necessari in memoria secondaria.
Questa strategia viene adottata perché quando il dbms deve andare in memoria secondaria a prendere i dati il tempo di risposta aumenta a dismisura e fare un veloce controllo nella sua cache può fargli risparmiare molto se trova informazioni sulla query che sta eseguendo; inoltre, questo è sicuramente utile perché nella realtà un database che viene interrogato è facile che ritorni spesso gli stessi dati, cioè quelli più richiesti e che quindi le query da effettuare siano sempre più o meno quelle, poi se gli viene chiesto qualcosa che non esegue spesso e che quindi non ha in cache, ci metterà di più per rispondere.
Quindi la distinzione tra i 2 tempi di esecuzione con cache e senza cache è necessaria per confrontare a pieno il funzionamento dei 2 dbms.
I tempi di esecuzione senza cache, sono ottenuti riavviando ogni volta il dbms, in questo modo facciamo si che la cache si svuoti e che il tempo non sia influenzato da informazioni già presenti; invece per i tempi "con cache" è sufficiente eseguire la stessa query più volte, ovviamente escludendo il primo tempo di risposta.

4.3.1 – Query su singola tabella:

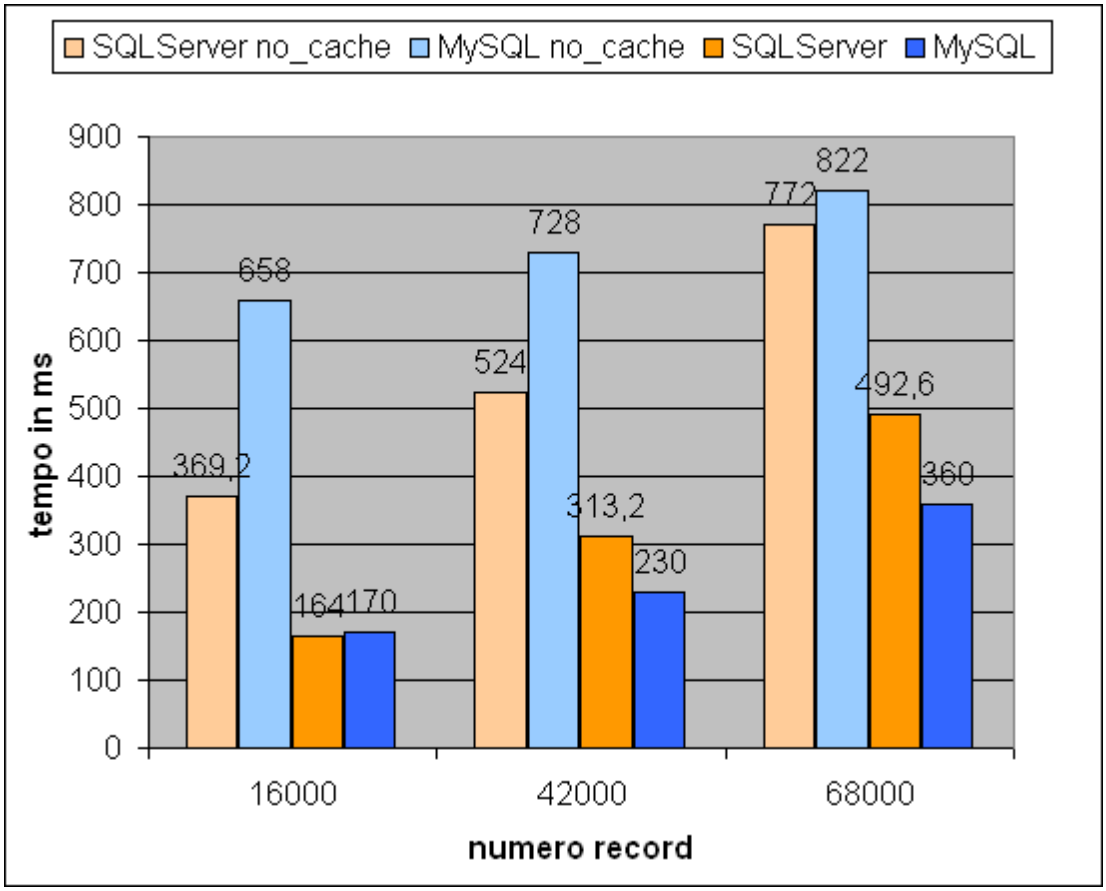
La prima tabella su cui andiamo ad eseguire le query descritte nel paragrafo 4.2 è studente21k, partiamo da questa perché è la più piccola e procederemo in ordine crescente. In tutte e 3 le query eseguite senza cache, possiamo osservare che SQLServer è sempre più veloce di circa 3 decimi di secondo; per quanto riguarda le query con cache i tempi sono più o meno uguali anche se con l'aumentare del numero di record ritornati MySQL si avvantaggia anche se di pochissimo.

		SQLServer			MySQL		
tabella: **studente21k**							
		20%	50%	80%	20%	50%	80%
senza cache		180	210	290	520	540	570
		190	240	260	520	530	560
		210	210	270	530	550	570
		200	260	300	510	530	550
		190	220	270	520	540	560
	MEDIA	**194 ms**	**228 ms**	**278 ms**	**520 ms**	**538 ms**	**562 ms**
con cache		40	83	130	40	60	80
		40	70	130	40	60	80
		40	80	120	40	60	80
		40	80	130	40	60	80
		40	83	120	40	60	80
	MEDIA	**40 ms**	**79,2 ms**	**126 ms**	**40 ms**	**60 ms**	**80 ms**



Per la tabella studente84k il tempo di risposta senza cache è sempre minore per SQLServer anche se con l'aumentare del numero dei record il divario diminuisce;
con la cache i tempi partono quasi uguali, ma MySQL diventa un po' più veloce con l'aumentare dei record.

		SQLServer			MySQL		
tabella: **studente84k**							
		20%	50%	80%	20%	50%	80%
senza cache		353	550	790	670	730	810
		283	570	770	680	740	790
		560	470	760	630	720	830
		300	510	780	660	730	850
		350	520	760	650	720	830
	MEDIA	**369,2 ms**	**524 ms**	**772 ms**	**658 ms**	**728 ms**	**822 ms**
con cache		170	310	490	170	230	350
		160	310	490	170	230	360
		160	313	490	170	230	360
		170	310	493	170	230	370
		160	323	500	170	230	360
	MEDIA	**164 ms**	**313,2 ms**	**492,6 ms**	**170 ms**	**230 ms**	**360 ms**

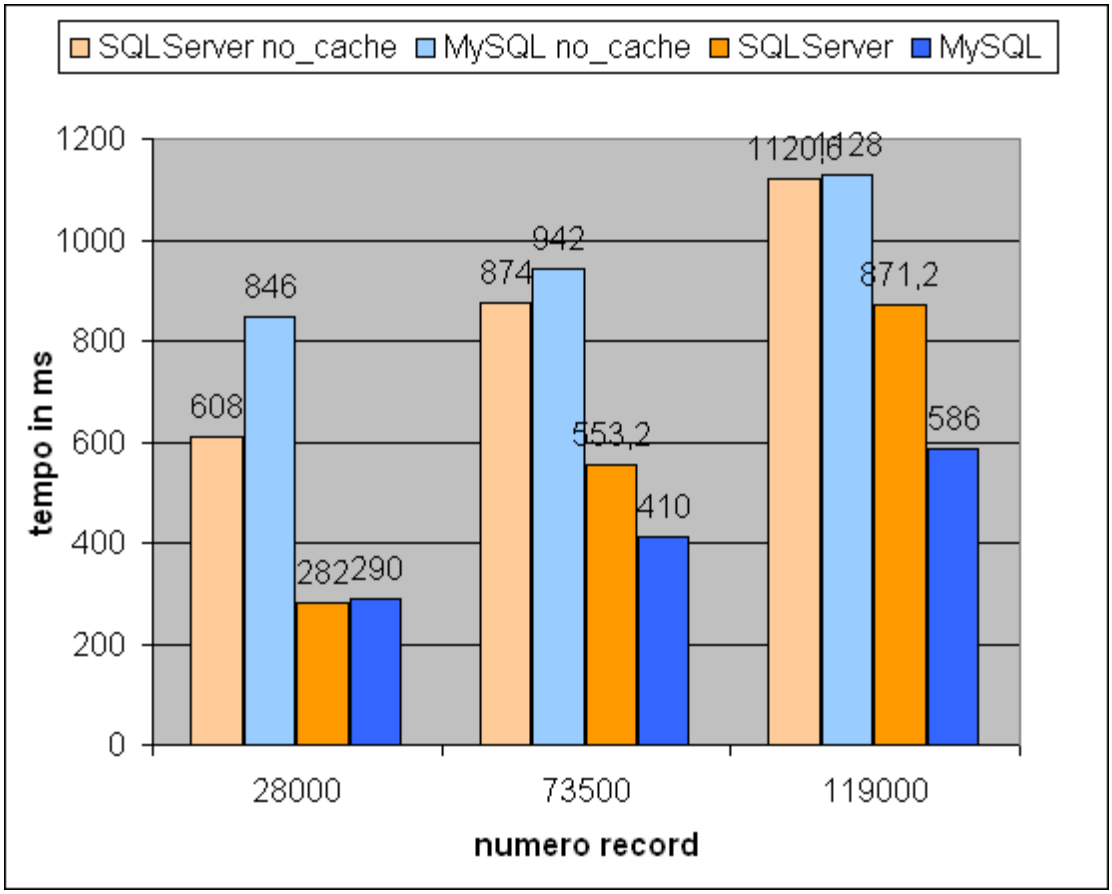


Per la tabella studente147k l’andamento dei tempi è uguale a quello di studente84k, notando che per l’ 80% dei record i tempi senza cache sono molto vicini, ma con SQLServer ancora in vantaggio, e il divario tra i tempi con cache è sempre più a favore di MySQL.

SQLServer MySQL

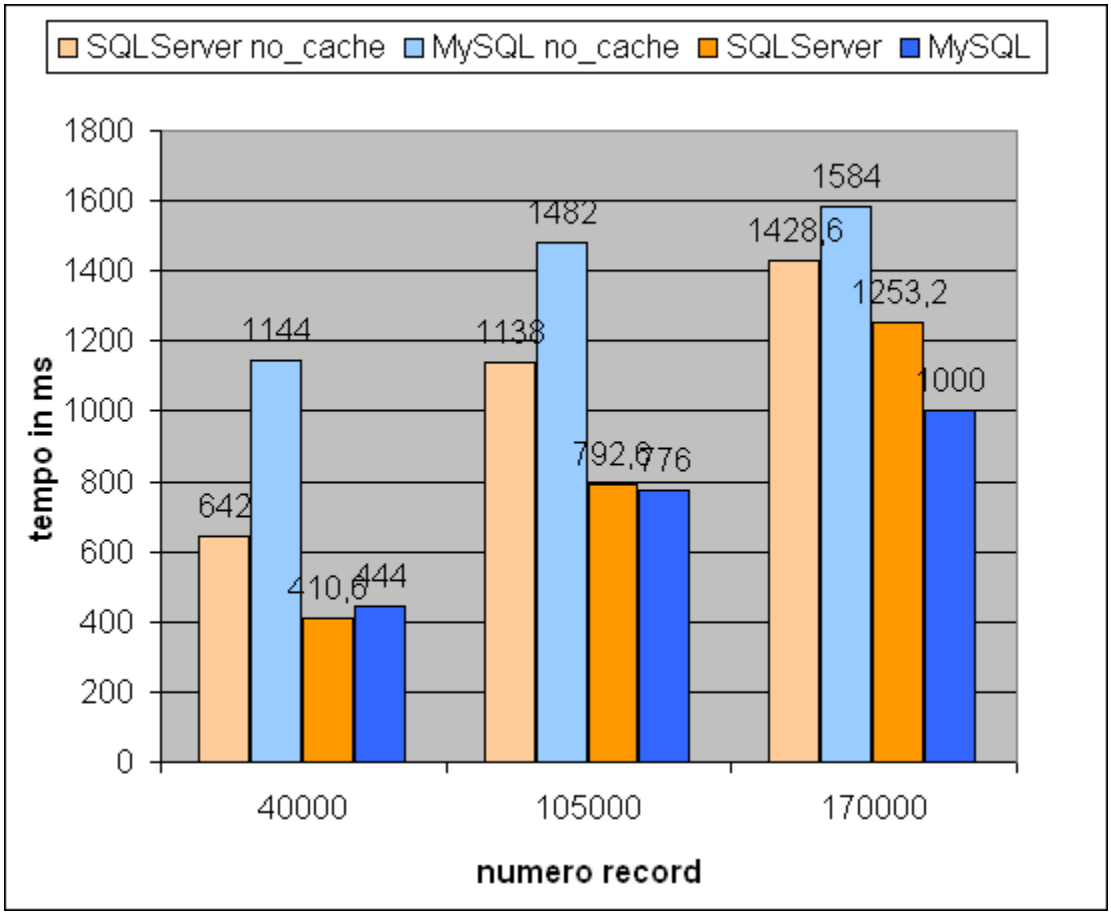
tabella: **studente147k**

		20%	50%	80%	20%	50%	80%
senza cache		590	830	1130	840	950	1140
		620	970	1150	850	950	1130
		620	810	1083	850	960	1140
		600	810	1090	830	930	1110
		610	950	1150	860	920	1120
	MEDIA	**608 ms**	**874 ms**	**1120 ms**	**846 ms**	**942 ms**	**1128 ms**
con cache		290	560	873	290	410	580
		280	553	870	290	410	590
		280	560	873	290	410	580
		280	550	860	290	410	600
		280	543	880	290	410	580
	MEDIA	**282 ms**	**553,2 ms**	**871,2 ms**	**290 ms**	**410 ms**	**586 ms**



Nella tabella studente210k SQLServer rimane sempre più veloce nelle query senza cache, mentre nelle query con cache i tempi sono circa uguali fino all’ 80% dove MySQL diventa più veloce.

		SQLServer			MySQL		
tabella: **studente210k**							
		20%	50%	80%	20%	50%	80%
senza cache		640	1190	1420	1160	1490	1570
		640	1150	1423	1140	1480	1600
		640	1130	1430	1150	1470	1560
		650	1110	1410	1140	1470	1600
		640	1110	1460	1130	1500	1590
	MEDIA	**642 ms**	**1138 ms**	**1428 ms**	**1144 ms**	**1482 ms**	**1584 ms**
con cache		400	790	1250	460	800	1010
		410	790	1250	450	770	1000
		423	790	1263	460	760	980
		410	790	1250	430	770	990
		410	803	1253	420	780	1020



4.3.2 – JOIN:

Adesso guardiamo i tempi di risposta ai join, partendo dal join a 2 vie che coinvolge le tabelle studente210k e esame che hanno uguale la colonna matricola.
Ecco le query che analizziamo:

```
select s.matricola,e.voto,e.corso
from esame as e [inner] / left / right / full join studente210k as s on e.matricola=s.matricola
```

Come già detto, MySQL non supporta il full join, ma dobbiamo fare la UNION tra il left join e il right join:

```
select s.matricola,e.voto,e.corso
from esame as e left join studente210k as s on e.matricola=s.matricola
union
select s.matricola,e.voto,e.corso
from esame as e right join studente210k as s on e.matricola=s.matricola
```

Con inner join (che da adesso chiameremo solamente join) e left join il numero di record che ritornano è 23196, mentre con right join e full join è 227696.
SQLServer risulta molto più veloce per le query senza cache, soprattutto nel right join e nel full join, anche se per il full join MySQL ha la "scusante" che deve unire 2 operazioni.
MySQL tiene testa a SQLServer solo nelle query con cache usando il join e il left join, mentre per il resto è molto più lento.
Nel grafico consideriamo solo il join e il full join, poiché il left e il right hanno andamenti simili.

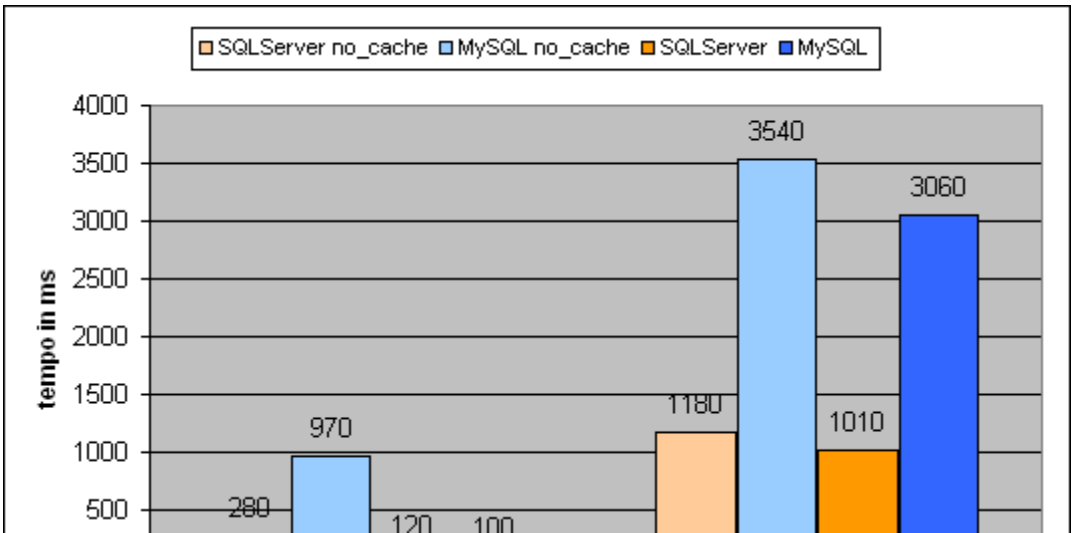
tabelle: esame, studente210k

senza cache

	SQLServer	MySQL	SQLServer	MySQL
	join		full join	
	250	1000	1190	3720
	340	960	1180	3450
	250	950	1170	3450
MEDIA	**280 ms**	**970 ms**	**1180 ms**	**3540 ms**
	left join		right join	
	400	1060	1193	2860
	280	950	1170	3080
	250	950	1120	3110
MEDIA	**310 ms**	**986,6 ms**	**1161 ms**	**3016 ms**

con cache

	SQLServer	MySQL	SQLServer	MySQL
	join		full join	
	120	100	1000	2960
	120	100	1000	3090
	120	100	1030	3130
MEDIA	**120 ms**	**100 ms**	**1010 ms**	**3060 ms**
	left join		right join	
	120	100	980	2210
	120	100	990	2350
	120	100	983	2320
MEDIA	**120 ms**	**100 ms**	**984 ms**	**2293 ms**



Per le query di join a 3 vie viene aggiunta la tabella corso che ha la stessa colonna di esame per quanto riguarda il corso, essendo foreign key.
In questo join a 3 vie considereremo solo il join e il full join tra tutte e 3 le tabelle:

```
select s.matricola,e.voto,c.cod_corso
from corso as c join / full join esame as e on c.cod_corso=e.corso
                  join / full join studente210k as s on e.matricola=s.matricola
```

In MySQL il full join sarà:

```
select s.matricola,e.voto,c.cod_corso
from corso as c left join esame as e on c.cod_corso=e.corso
                left join studente210k as s on e.matricola=s.matricola
union
select s.matricola,e.voto,c.cod_corso
from corso as c right join esame as e on c.cod_corso=e.corso
                right join studente210k as s on e.matricola=s.matricola
```

Anche in questo caso, come in quello precedente, SQLServer è molto superiore a MySQL e l'unico caso in cui i 2 dbms hanno tempi simili è il join con cache.

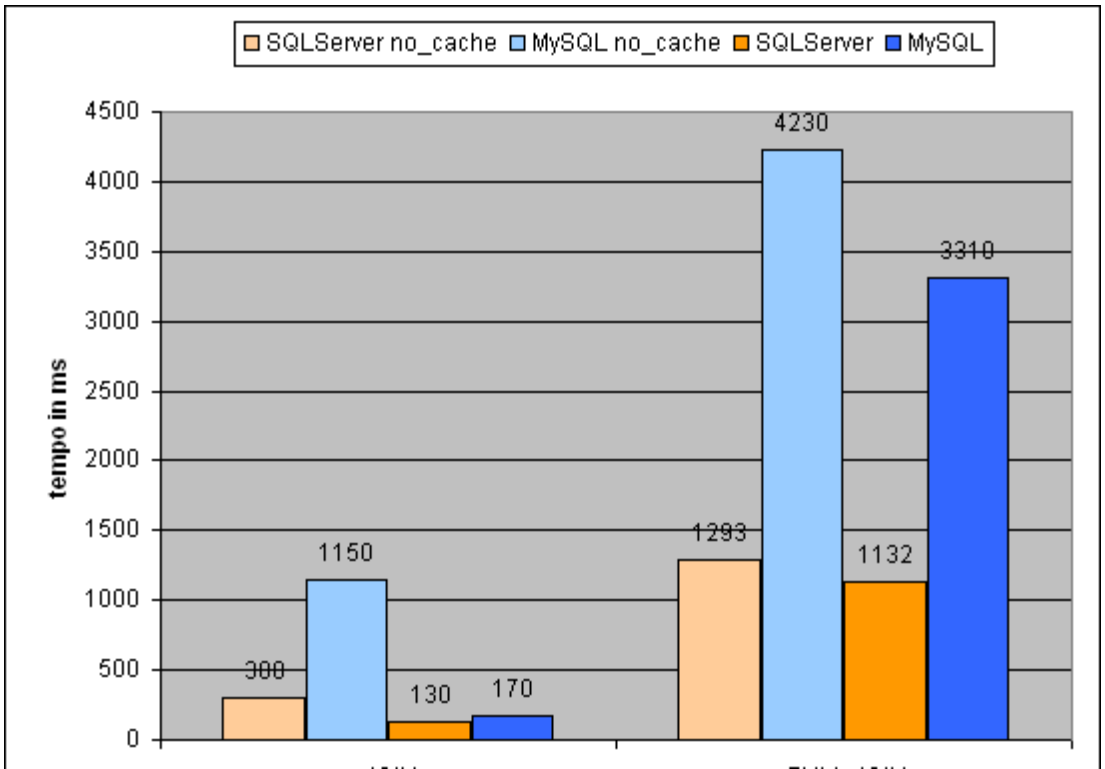
tabelle: corso, esame, studente210k

senza cache

	SQLServer	MySQL	SQLServer	MySQL
	join		full join	
	280	1130	1453	4280
	340	1150	1303	4120
	280	1170	1123	4290
MEDIA	**300 ms**	**1150 ms**	**1293 ms**	**4230 ms**

con cache

	SQLServer	MySQL	SQLServer	MySQL
	join		full join	
	130	170	1140	3350
	130	170	1123	3250
	130	170	1133	3330
MEDIA	**130 ms**	**170 ms**	**1132 ms**	**3310 ms**



Nei join a 4 vie inseriamo l'ultima tabella, docente che si collega a corso tramite la colonna cod_docente. Come nel join a 3 vie le query saranno di soli join e full join:

```
select s.matricola,e.voto,c.cod_corso,d.cod_docente
from corso as c join / full join docente as d on c.docente=d.cod_docente
                join / full join esame as e on c.cod_corso=e.corso
                join / full join studente210k as s on e.matricola=s.matricola
```

Per il full join in MySQL:

```
select s.matricola,e.voto,c.cod_corso,d.cod_docente
from corso as c left join docente as d on c.docente=d.cod_docente
                left join esame as e on c.cod_corso=e.corso
                left join studente210k as s on e.matricola=s.matricola
union
select s.matricola,e.voto,c.cod_corso,d.cod_docente
from corso as c right join docente as d on c.docente=d.cod_docente
                right join esame as e on c.cod_corso=e.corso
                right join studente210k as s on e.matricola=s.matricola
```

I valori riportati nel grafico hanno lo stesso andamento del join a 3 vie, infatti SQLServer è sempre più veloce, nel full join la differenza inizia ad essere troppo evidente, e MySQL riesce ad avere le stesse prestazioni solo nel join con cache.

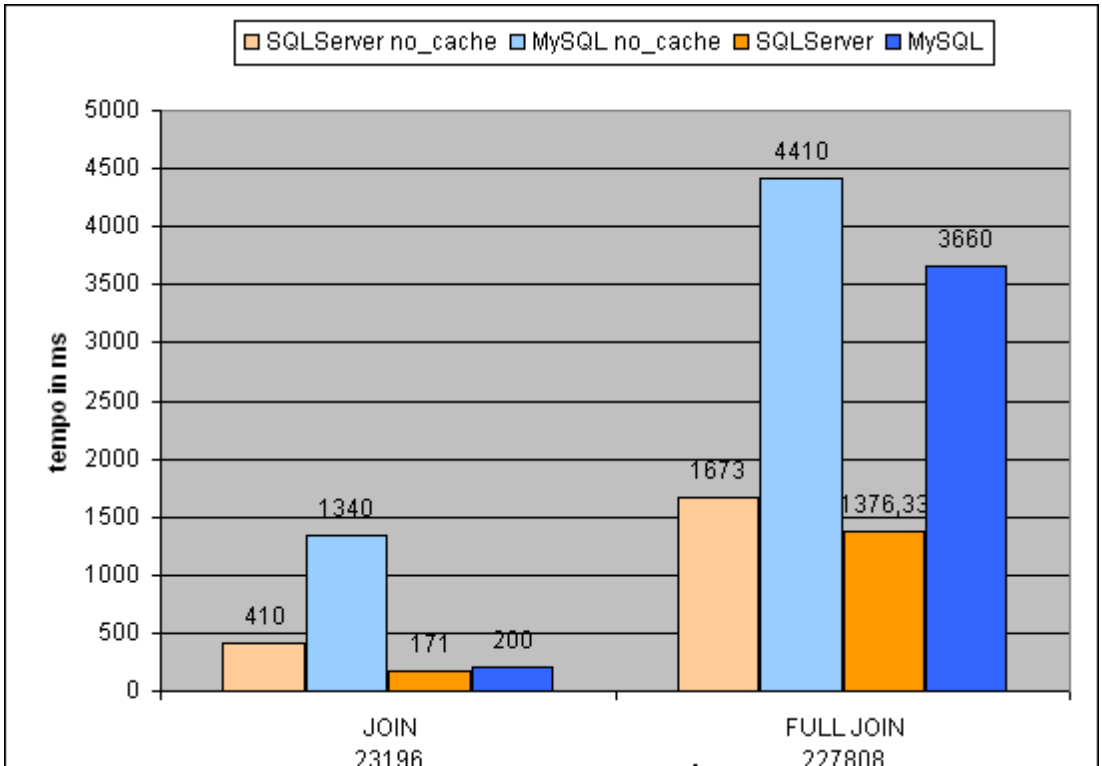
tabelle: corso, docente, esame, studente210k

senza cache

	SQLServer	MySQL	SQLServer	MySQL
	join		full join	
	440	1040	1693	4360
	420	1450	1653	4420
	370	1530	1673	4450
MEDIA	**410 ms**	**1340 ms**	**1673 ms**	**4410 ms**

con cache

	SQLServer	MySQL	SQLServer	MySQL
	join		full join	
	173	200	1373	3520
	170	200	1383	3670
	170	200	1373	3790
MEDIA	**171 ms**	**200 ms**	**1376,3 ms**	**3660 ms**



4.3.3 – JOIN con tabelle indicizzate:

In questo paragrafo andiamo ad eseguire alcune query sulle tabelle su cui abbiamo inserito degli indici su una o più colonna e in modo diverso, per vedere come cambiano i risultati all'interno di uno stesso dbms, per poi passare ad un confronto tra i 2.
Partiamo con il join a 2 vie dove la query che eseguiamo è:

```
select s.matricola,e.voto,s.citta
from esame as e full join studente210k as s on e.matricola=s.matricola
where e.corso=5
```

Prima di tutto però è necessario sapere quando un dbms usa o no un indice: i dbms hanno diverse tabelle di sistema, che utilizzano per memorizzare informazioni su tutte le operazioni fatte sui database; in alcune di queste, ad esempio in SQLServer la tabella sysindexes, sono memorizzate le informazioni che riguardano gli indici sulle tabelle come la densità, cioè quanti valori ci sono per chiave di indice.
Con dati di questo genere e con dati sulle grandezze delle tabelle, il dbms crea delle statistiche utili durante l'esecuzione di una query, le analizza e in base a quelle decide se gli conviene usare uno o più indici presenti oppure fare una scansione sequenziale.
Creiamo sulla tabella esame diversi indici (ricordandoci che matricola,corso sono chiave primaria e formano l'indice cluster e che su studente210k matricola è indice cluster perché chiave primaria) per ottenere e confrontare le diverse soluzioni e i tempi di risposta:

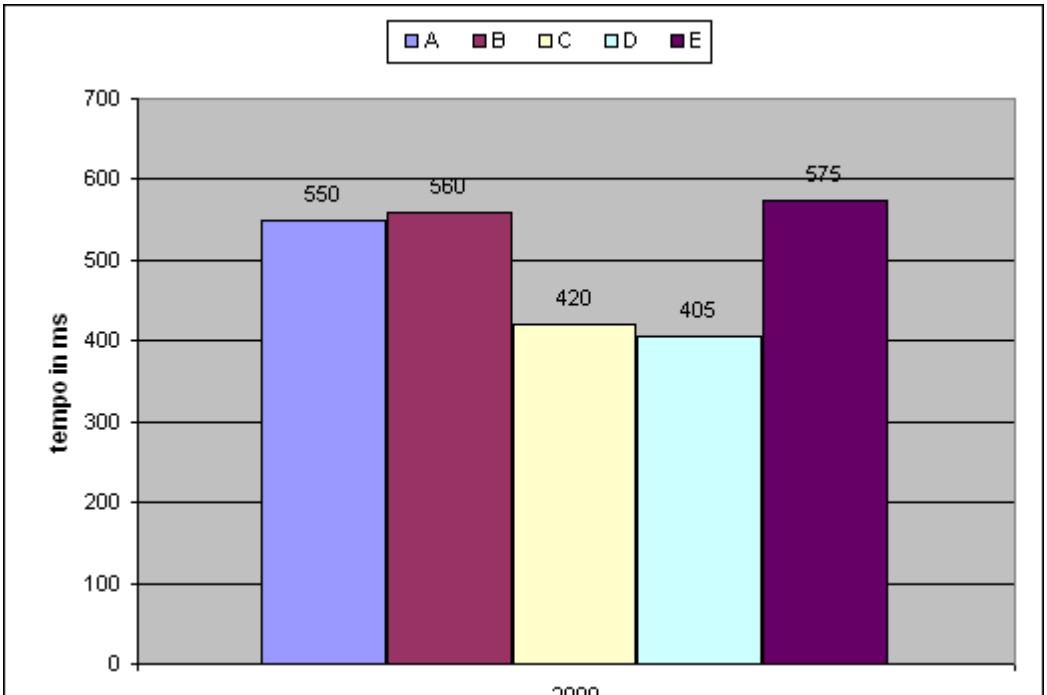
A: indice su corso
B: indice su matricola e su corso
C: indice su matricola e sulla coppia corso,voto
D: indice sulla coppia corso,voto
E: nessun indice

Dai risultati delle nostre prove abbiamo ottenuto che i 2 dbms agiscono allo stesso modo, nel senso che le query più veloci risultano la C e la D, anche se SQLServer si dimostra sempre più veloce di MySQL (si ricorda che MySQL non supporta il full join); questa infatti è la scelta giusta, poiché dovendo selezionare 2000 righe su 227696 se c'è un indice viene usato di sicuro, perché converrà sempre piuttosto che una scansione sequenziale: con le soluzioni C e D, abbiamo il miglior vantaggio perché definiamo un indice unclustered sulla coppia

corso,voto che sono proprio le colonne coinvolte nella tabella Esame, sulle quali il dbms farà "poca fatica" ad accedere perché indicizzate: avremo infatti nel piano di esecuzione un index seek esame where e.corso=5, mentre per la tabella Studente210k avendo solo l'indice cluster esegue una clustered index scan.
In MySQL non è possibile la soluzione E, perché dichiarando come primary key la coppia matricola,corso e poi le foreign key su matricola e su corso, il dbms crea automaticamente un indice unclustered su corso, che non possiamo eliminare se non eliminiamo la foreign key.
Nei grafici successivi abbiamo riportato un solo valore per tipo di query, che è la media calcolata su 5 esecuzioni; inoltre rappresenteremo nei grafici solo i tempi di risposta delle query senza cache, che riteniamo essere di maggiore importanza.

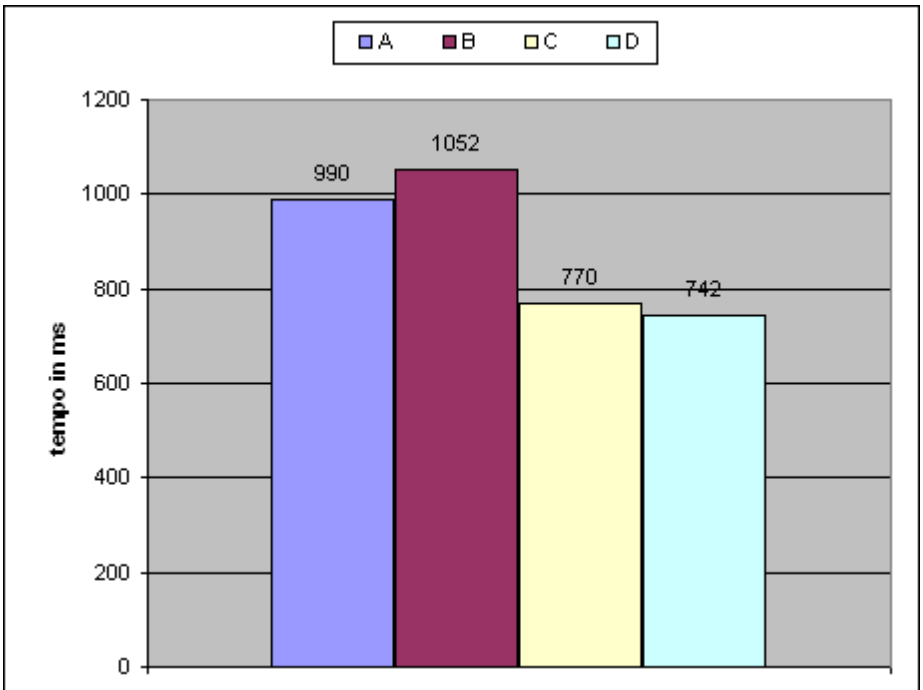
JOIN a 2 vie con INDICI in SQLServer

	senza cache	**con cache**
tipo A	550 ms	20 ms
tipo B	560 ms	20 ms
tipo C	420 ms	20 ms
tipo D	405 ms	20 ms
tipo E	575 ms	20 ms



JOIN a 2 vie con INDICI in MySQL

	senza cache	**con cache**
tipo A	990 ms	38 ms
tipo B	1052 ms	32 ms
tipo C	770 ms	20 ms
tipo D	742 ms	20 ms



Procediamo ad analizzare il join a 3 vie aggiungendo la tabella corso su cui creiamo un indice unclustered sulla colonna docente che è anche foreign key ed ha come indice cluster la chiave primaria cod_corso; definiamo 2 nuove query:

```
select s.matricola,e.voto,c.docente,s.citta
from corso as c full join esame as e on c.cod_corso=e.corso
                full join studente210k as s on e.matricola=s.matricola
where e.corso=5 / c.cod_corso=5
```

con F1: e.corso, nessun indice
F2: c.cod_corso, nessun indice
G1: e.corso, indice sulla coppia corso,voto e su docente
G2: c.cod_corso, indice sulla coppia corso,voto e su docente

```
select s.matricola,e.voto,c.docente,s.citta
from corso as c full join esame as e on c.cod_corso=e.corso
                full join studente210k as s on e.matricola=s.matricola
where c.docente=9
```

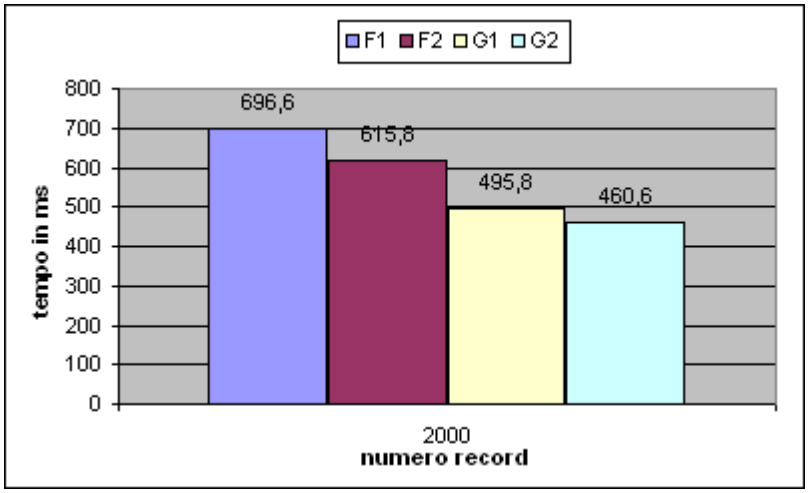
con H: indice culla coppia corso,voto
I: indice sulla coppia corso,voto e docente
L: nessun indice

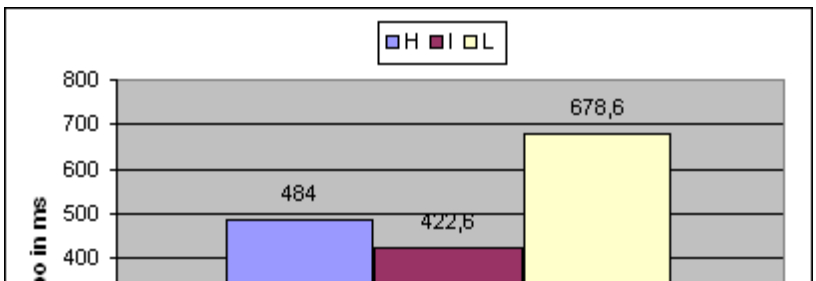
Nella prima query abbiamo voluto evidenziare che se possiamo scegliere tra clausola di where su un indice unclustered (e.corso=5) e su una chiave primaria (c.cod_corso=5) conviene una ricerca sulla chiave primaria, infatti la G2 è più veloce di G1 per questo motivo, mentre F2 è più veloce di F1 perché non essendoci indici oltre quelli clustered, una ricerca sulla chiave primaria è ovviamente migliore.
La differenza tra F1 e F2 sta nel fatto che nella prima abbiamo una clustered index scan sia su esame che su corso, mentre nella seconda clustered index seek c.cod_corso=5 su corso e una clustered index scan e.corso=c.cod_corso su esame; se poi mettiamo gli indici, in G1 abbiamo index seek e.corso=5 su esame e clustered index scan su corso, mentre in G2 clustered index seek c.cod_corso=5 su corso e index seek e.corso=c.cod_corso su esame.
Sulla tabella studente210k è sempre clustered index scan.
In conclusione, una ricerca su chiave primaria è conveniente, perché se la chiave primaria è implicata in una clausola di join la ricerca sulla tabella che ha la foreign key diventa una index

seek che restringe il campo dei valori, piuttosto che una clustered index scan (ovviamente se abbiamo definito un indice sulla colonna della foreign key).
Se invece nella select vogliamo una colonna che non appartiene agli indici unclustered, ad esempio nel nostro caso e.data, allora è indubbiamente migliore una ricerca con clausola where su chiave primaria, perché in questo caso l'indice unclustered non servirebbe a nulla e il dbms effettuerebbe una clustered index scan e.corso=5 su esame.
Nella seconda query sul join a 3 vie, vogliamo vedere lo stesso discorso fatto per il join a 2 vie, cioè quanto conviene utilizzare gli indici su una query che ritorna 1002 record su 227754. Anche in questo caso da notare che MySQL non ci permette di togliere tutti gli indici unclustered e neanche l'indice su docente nella tabella corso, perché foreign key, quindi non abbiamo potuto eseguire le query F1, F2, H e L.

JOIN a 3 vie con INDICI in SQLServer

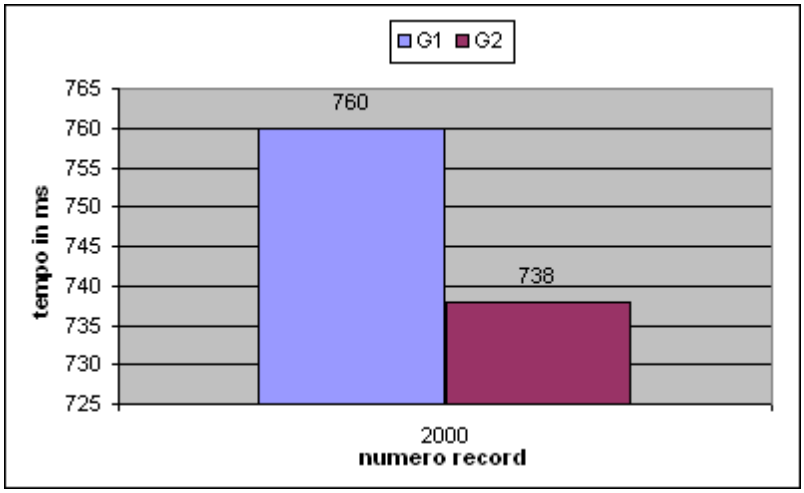
	senza cache	**con cache**
tipo F1	696,6 ms	30 ms
tipo F2	615,8 ms	28 ms
tipo G1	495,8 ms	28 ms
tipo G2	460,6 ms	20 ms
tipo H	484 ms	25 ms
tipo I	422,6 ms	13 ms
tipo L	678,6 ms	13 ms





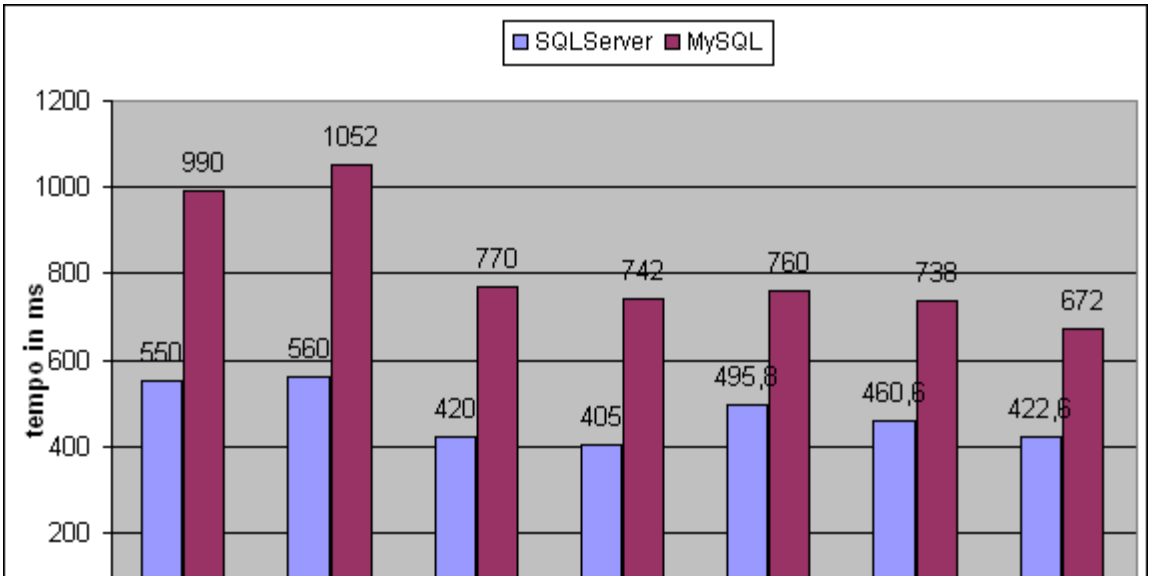
JOIN a 3 vie con INDICI in MySQL

	senza cache	**con cache**
tipo G1	760 ms	27 ms
tipo G2	738 ms	30 ms
tipo I	672 ms	10 ms



CONFRONTO QUERY CON INDICI

	SQLServer	MySQL
tipo A	550 ms	990 ms
tipo B	560 ms	1052 ms
tipo C	420 ms	770 ms
tipo D	405 ms	742 ms
tipo E	575 ms	--------
tipo F1	696,6 ms	--------
tipo F2	615,8 ms	--------
tipo G1	495,8 ms	760 ms
tipo G2	460,6 ms	738 ms
tipo H	484 ms	-------
tipo I	422,6 ms	672 ms
tipo L	678,6 ms	-------



5 – CONCLUSIONI:

Prima delle conclusioni finali sulle prove effettuate, dobbiamo trattare una differenza significativa che c'è tra i 2 dbms che abbiamo confrontato, cioè il costo del software, ricordando che mentre MySQL è disponibile sia per i sistemi Unix che per Windows, anche se prevale un suo utilizzo in ambito Unix, SQLServer è possibile usarlo solo su piattorme del suo produttore Microsoft, cioè la famiglia Windows.
Partiamo parlando di MySQL che possiamo trovare in 2 versioni: MySQL Community Server e MySQL Enterprise che è l'offerta più completa di software, servizi e supporto MySQL; mentre la prima edizione è totalmente gratuita ed ha ovviamente delle restrizioni, la seconda è a pagamento e si divide a sua volta in 4 livelli che si differenziano per i servizi che possiedono: basic, silver, gold, platinum (vedi https://shop.mysql.com/enterprise/); a seconda di quella scelta il prezzo varia tra i 600 $ e i 5000 $.
SQLServer è tutt'altro che gratuito e a seconda dell'edizione cambia il prezzo:

Livello	Funzionalità	Prodotti di esempio	Prezzo
Gratuita	Funzionalità database limitate Limitazioni di memoria, dimensioni del database e funzionalità	SQL Server Express Edition	$0
Di base	Funzionalità database di base Strumenti di gestione semplificati Sicurezza di base Fino a 2 CPU, limitazioni di memoria	Microsoft SQL Server Workgroup Edition	$500-$5.000 per CPU
Standard	Disponibilità Funzionalità database complete Strumenti di gestione di base Fino a 4 CPU o nodi	Microsoft SQL Server Standard Edition	$5.000-$15.000 per CPU
Enterprise	Disponibilità elevata Scalabilità Strumenti di gestione di alto livello Sicurezza enterprise Nessuna limitazione di CPU	Microsoft SQL Server Enterprise Edition	$25.000-$40.000 per CPU

Nota. Tutti gli importi sono basati sul modello Per processore, si riferiscono ai prezzi applicati negli Stati Uniti e sono espressi in dollari. I prezzi sono stati ricavati dalle informazioni pubblicate sui siti Web dei produttori. [7]

In genere, oltre alle licenze per i database i clienti acquistano anche la manutenzione e il supporto. Spesso il costo di questi servizi viene determinato in base a una percentuale del prezzo di listino, solitamente pari al 20-25%.
Grazie alla manutenzione, i clienti hanno diritto alle versioni più recenti del software.

Per quanto riguarda le prove da noi effettuate possiamo arrivare alle seguenti conclusioni:

- nelle query “semplici”, che non contengono join, senza l’utilizzo della cache, SQLServer è sempre superiore, anche se quando viene usata la cache, cioè nella maggior parte dei casi poiché qualche informazione sulle tabelle implicate è facile da trovare, MySQL tiene il passo di SQLServer e alla lunga, con l’aumentare dei record risultanti, è addirittura migliore.

- appena le query si complicano, facendo dei join tra una o più tabelle, SQLServer è molto più veloce di MySQL in tutte le circostanze e il divario aumenta con l’aumentare dei record e soprattutto con le operazioni di full join che da MySQL non sono supportate.

- nelle query con tabelle indicizzate, sempre premettendo che SQLServer è più veloce, entrambi i dbms riescono ad ottenere vantaggi significativi rispetto alle stesse tabelle non indicizzate, anche se MySQL crea automaticamente alcuni indici su foreign key che non è possibile togliere e questo può dare fastidio ad uno “spirito libero”.

Per riassumere se dobbiamo effettuare una scelta tra i 2, SQLServer sarebbe sicuramente la nostra se avessimo un budget illimitato; ma se dovessimo tenere soprattutto d’occhio questo aspetto allora la scelta ricaderebbe sul ruolo che assume il dbms nel nostro impiego: se vogliamo essere sicuri di una efficienza e di una tecnologia che sia tra le prime a livello mondiale dobbiamo mettere in preventivo un non indifferente costo di licenza, se invece per quello che facciamo è sufficiente un dbms leggero, che non ci interessi il tempo di risposta per query complicate e una piccola spesa per la licenza allora MySQL farebbe di sicuro al caso nostro. L’unico altro punto che darei a favore di MySQL oltre al basso costo, è la possibilità di poterlo installare non solo su sistemi Windows, quindi di non poter essere vincolati dal sistema operativo.

Ovviamente queste considerazioni fatte da me sono basate solo sulle prove effettuate e su ricerche correlate ad esse; un confronto più approfondito tra i 2 dbms porterebbe di sicuro alla luce tantissime altre differenze per le quali la scelta di uno tra i due diventerebbe sempre più complicata e difficile.

Ultimo auspicio dedicato alla tecnologia che non si fermi mai e che possa veramente dare una mano a tutti senza avere in testa solo il business.

BIBLIOGRAFIA:

Il materiale e la documentazione necessari alla realizzazione di questa tesi consistono in una varietà di documenti ed articoli reperiti sul Web, oltre a diversi manuali e testi cartacei.

Principali documenti online:

- [1] “Breve storia di SQL”, http://database.html.it/guide/lezione/1309/breve-storia-di-sql/
- [2] “Il modello relazionale”, http://database.html.it/guide/lezione/1308/il-modello-relazionale/
- [3] “SQL Server da Sybase a Microsoft”, http://database.html.it/articoli/leggi/1657/sql-server-da-sybase-a-microsoft/
- [4] “Microsoft SQL Server”, http://it.wikipedia.org/wiki/Microsoft_SQL_Server
- [5] “MySQL chi sei?”, http://asp.html.it/guide/lezione/2066/mysql-chi-sei/
- [6] “MySQL”, http://213.92.21.88/meo/white/oracle/mys.htm
- [7] “SQL_UnderstandingDBPricing2005.doc”, http://www.microsoft.com/sql/howtobuy/

Ulteriore documentazione elettronica:

- SQL Server 2000 Books Online
- MySQL 5.0 Reference Manual

Testi e manuali cartacei:

- “Progetto di Basi di Dati Relazionali”, Beneventano, Bergamaschi, Vincini, Pitagora Editrice, Bologna 2000
-“Lezioni di Basi di Dati”, Ciaccia, Maio, Società Editrice Esculapio, Bologna 1995