Relazione del progetto di Gobbi Giovanni (832014),
Fiordarancio Matteo (825597)e Lunadei Jacopo (839278)

Lo scopo dell'elaborato è quello di simulare un database Sql. Per soddisfare la consegna sono state assunte le seguenti limitazioni nelle lunghezze delle stringhe dovute al linguaggio di sviluppo:

- La lunghezza massima di una stringa in generale è stata assunta di 128 caratteri
- La lunghezza massima di una riga nel file è stata assunta di 1024 caratteri

Entrambe le lunghezze sono modificabili semplicemente modificando i valori nelle #define MAX_LEN e #define MAX_ROW.

Sono inoltre state usate le seguenti librerie:

- #include <string.h>
- #include <stdio.h>
- #include <stdlib.h>
- #include <wchar.h>

Ove possibile, si è inoltre cercato di utilizzare allocazioni dinamiche, in modo da non imporre limitazioni di spazio nel caso di un numero elevato di righe o di colonne. Si è fatta inoltre attenzione a non incorrere in problemi di overhead allocando dinamicamente troppe volte.

Sono stati anche aggiunti numerosi controlli sulla correttezza della query, in modo da evitare il più possibile il problema di inconsistenza dei dati.

Come si può notare, si è cercato per quanto possibile di dividere il programma in più funzioni. Quelle che riteniamo vengano chiamate più spesso sono state opportunamente inserite in modo tale da poter essere richiamate quando necessarie.

Abbiamo inoltre aggiunto nel programma numerosi commenti al fine di semplificare la comprensione del codice, che ci rendiamo conto possa essere complessa. Per ogni funzione è stato indicato il tempo necessario all'esecuzione della stessa nel caso peggiore.

Qui di seguito andremo ora a spiegare le scelte algoritmiche fatte per le varie funzioni richieste.

READ FROM FILE: Tempo O(n righe * m colonne)
E' la funzione che viene chiamata da tutti i filtri select. Lo scopo di questa funzione è salvare in un struttura tridimensionale dinamica i dati letti dal file. La

tridimensionalità è dovuta al fatto che abbiamo 3 valori in gioco: la lunghezza di ogni singola stringa (fissata a MAX_LEN) all'interno di una data riga e colonna, il numero di righe (dinamico) e il numero di colonne (dinamico). Il tempo impiegato per questa lettura è il minimo possibile, ovvero il tempo necessario a leggere tutte le righe del file e per ogni riga le colonne. Essendo tale lettura non influenzata dall'ordine in cui si parte (in quanto vanno comunque letti tutti i dati) abbiamo deciso di partire semplicemente dall'alto.

CREATE TABLE: Tempo O(m colonne)
Riguardo questa funzione non sono state fatte particolari scelte architetturali. La funzione si limita semplicemente a creare il file e a scrivere la riga di introduzione della tabella come da query. Fa uso della funzione aggiungi txt che aggiunge l'omonima estensione in tempo O(1). Si è inoltre prestata attenzione che alla fine della funzione il file venisse chiuso.

INSERT: Tempo O(m colonne)
La funzione si occupa di inserire i dati presi della query di input ordinatamente all'interno del file. Il tempo di esecuzione di tale query è pari al numero di colonne inserite O(m colonne). Si è prestata particolare attenzione che le colonne inserite in input fossero conformi a quelle della tabella.

SELECT NO FILTER: Tempo O(n righe * m colonne)
La funzione si occupa di eseguire tutte le query select in cui non siano presenti filtri. Per scriverla, sono state fatte a priori delle considerazioni riguardo il costo minimo possibile di un algoritmo che risolvesse il problema. A seguito di tali considerazioni, si è osservato che il costo minimo per selezionare i risultati conformi alle richieste dell'utente è quantomeno il costo di lettura di tutte le righe e colonne. Per tale ragione, l'algoritmo da noi creato ha costo O(n righe * m colonne). Tale costo è determinato principalmente dalla funzione read from file.

SELECT WHERE: Tempo O(n righe * m colonne)
La funzione si occupa di eseguire tutte le query select in cui sia presente il filtro WHERE. Per lo svolgimento di questa funzione, sono state svolte le stesse considerazioni della funzione select no filter. Non possiamo escludere a priori alcun tipo di dato, pertanto è necessaria una lettura completa di tutte le righe. Per l'applicazione del criterio selezionato, una volta lette le righe una per volta, si va semplicemente a controllare che ciascuna riga rispetti la condizione. Il costo di tale funzione è lo stesso della select no filter, ovvero O(n righe * m colonne).

SELECT ORDER: Tempo O(n righe^2 * strlen + n righe * m colonne)
La funzione si occupa di eseguire tutte le query select in cui sia presente il filtro ORDER BY. E' la funzione computazionalmente più pesante, in quanto il

problema postoci è quello di ordinare, nella maggior parte dei casi, un array di stringhe. Per risolvere tale problema abbiamo deciso di rifarci a un algoritmo preso dalla letteratura, il bubble sort. L'algoritmo è stato scelto perché di facile implementazione. Infatti, l'algoritmo funziona scambiando due righe se il numero della prima è maggiore della seconda (nel caso in cui si voglia un ordinamento crescente). Scambiando questo criterio con quello della strcmp, siamo riusciti ad ottenere un algoritmo semplice in grado di ordinare una serie, anche lunga, di stringhe. Nel caso in cui il criterio di ordinamento sia numerico, si usa un bubble sort classico. Il costo dell'algoritmo è O(n righe * n righe * lunghezza stringhe da ordinare), il quale è maggiore del costo teorico di un merge sort O(n righe * log(n righe) * lunghezza stringhe), ma che nei dettagli implementativi risulta più veloce su un numero modesto di dati come quelli che ci si aspetta debba gestire questo programma.

SELECT GROUP: **Tempo O(n righe * n colonne + n righe * n gruppi)**

L'ultima funzione richiestaci di implementare è stata la select per raggruppamento. Per svolgere tale richiesta, si è creata una struttura di supporto come quella che si può vedere nella tabella.

Nome del gruppo	Occorrenze
Rossi	1
Bianchi	1
Fiordarancio	3
Gobbi	1
Lunadei	2

Ad ogni nuova riga letta, si va a controllare se la stessa è già stata inserita nella struttura attraverso un ciclo. Se essa è già stata inserita si incrementa il numero di occorrenze. In caso contrario la riga viene inserita all'interno del gruppo con valore Occorrenze pari ad 1. Il costo dell'algoritmo è pari a O(n righe * n colonne + n righe * n gruppi). Si sarebbe potuto ottenere un costo minore utilizzando funzioni di hashing perfetto, ma sarebbe stato molto complesso in quanto non ci è permesso di utilizzare funzioni di hashing presenti in librerie C già scritte.