Progetto Algoritmi e Strutture Dati

Anno Accademico 2017/18

Docente di Laboratorio: Dott.ssa Greta Sasso

Sommario

1 Descrizione Generale

L'obbiettivo del progetto è quello di simulare la gestione di un *Database* utilizzando dei file di testo (***.txt***) per la memorizzazione dei dati. Di seguito alcuni dettagli scelti per semplificare la gestione:

- Ogni tabella del DB sarà memorizzata in un diverso file di testo.
- La gestione di ogni tabella del DB avverrà utilizzando dei comandi in un linguaggio fornito nelle specifiche di progetto, che riprende la sintassi SQL.
- Per rappresentare il DB nei file di testo (*output*) verranno fornite nelle specifiche di progetto delle istruzioni precise, atte a facilitare la correzione e la verifica del lavoro svolto. Se, inoltre, si vogliono creare ed utilizzare delle strutture dati per gestire il DB, si utilizzi la modalità che si ritiene più opportuna. Si possono utilizzare tutte le strutture dati astratte affrontate dal corso (*alberi, hash table, code, pile, liste, grafi,…*).

2 Nozioni di Base

In questa sezione si vogliono fornire le informazioni di base per la comprensione del concetto di Database, sufficienti per lo svolgimento di questo progetto.

2.1 Definizione di Database

Un **Database** (*detto anche **DB***) è un archivio, una collezione di dati in formato elettronico, persistente. Esso raccoglie i dati organizzandoli logicamente in modo strutturato. L'utilizzo di un DB favorisce quindi operazioni quali l'inserimento, la cancellazione, l'aggiornamento e la ricerca di informazioni (e*ventualmente anche applicando dei filtri sui dati*).

2.2 Tabelle

Un DB si compone di diverse **tabelle**, ovvero delle strutture utili per organizzare i dati. Pensando, ad esempio, ad un DB per la gestione del sistema universitario, tramite le tabelle è possibile suddividere per categorie i dati presenti (*studenti, professori, corsi, aule, ecc…*). Ogni tabella raccoglie quindi un certo gruppo di dati.

Per meglio figurarsi la concezione di tabella, riporto un esempio tipico che solitamente si usa, ovvero un foglio di calcolo Excel, in cui sono memorizzati i dati organizzati per righe e colonne, in fig. 1 e fig. 2 vengono riportati due esempi pratici.

Matricola	Nome	Cognome	Età
12345	Mario	Rossi	21
54312	Paolo	Bianchi	20
123123	Giovanni	Verdi	21
1231	Paolo	Rossi	25

Figura 1 – *Tabella Studente interna al DB di un Università*

Cognome	Nome	Telefono
Bianchi	Mario	123456
Rossi	Paolo	43215
Verdi	Chiara	654654

Figura 2 – *Tabella Rubrica*

2.3 Linguaggio SQL

Il linguaggio SQL è un linguaggio che permette di effettuare le operazioni sul DB. Si compongono delle stringhe, dette **query**, con delle keyword in base alle operazioni che si vogliono effettuare, per il quale verranno forniti degli esempi in seguito.

In questo progetto si sfrutteranno solo alcune delle operazioni disponibili, utilizzando alcune particolari stringhe che verranno indicate nelle specifiche di progetto, in un linguaggio il quanto più simile all'SQL.

3 Specifiche di Progetto

Il progetto consiste nel simulare la gestione di un Database. In particolare è richiesta la realizzazione delle seguenti funzionalità:

1. Creazione di una nuova tabella
2. Inserimento di nuovi dati nella tabella
3. Ricerca all'interno delle tabelle (SELECT)
    a. Ricerca senza filtri
    b. Ricerca con filtri
        i. Filtro "WHERE"
        ii. Filtro di Ordinamento "ORDER BY"
        iii. Filtro di Raggruppamento "GROUP BY";

3.1 Gestione della Memorizzazione

Per garantire la persistenza, ogni tabella del DB deve essere memorizzata in un file testuale (*.txt*) diverso, utilizzando come nome del file quello della tabella. Tutti i file di testo saranno pertanto raccolti in una directory che quindi conterrà tutte le tabelle del DB in questione.

La struttura dei file di testo deve seguire le precise indicazioni che verranno fornite in seguito.

3.2 Funzionalità

Lo studente deve implementare la seguente funzione:

```
bool executeQuery(char*);
```

Questa prende in input la stringa contenente la query, la stringa, per eseguire le operazioni indicate sopra. La funzione restituirà un valore booleano (**`true, false`**) che indicherà se la query è andata a buon fine o meno.

In questa sezione si descrive per ogni funzionalità quale **query** sarà necessario utilizzare. Una query è una stringa che utilizza delle keyword, tenendo fede al linguaggio SQL.

N.B. Ricordarsi di considerare gli spazi presenti all'interno delle query!

3.2.1 Creazione di una Nuova Tabella

Per la creazione di una nuova tabella occorre utilizzare la query:

```
CREATE TABLE nome_tabella (nome_colonna1,nome_colonna2,…)
```

La creazione di una tabella corrisponde alla creazione di un file di testo in formato *.txt* il cui nome corrisponde a `nome_tabella`.

Nella fase di creazione saranno specificati:

- Il **nome della tabella** ("*nome_tabella" nella query soprastante*)
- I **nomi delle colonne** ("*nome_colonna1, ecc…" nella query soprastante*) separando ogni colonna con una virgola.

Richiamando la funzione "executeQuery(char*)" con questa stringa, dovrà essere creato un file di testo chiamato "*nome_tabella.txt*", contenente una sola riga di testo in questo formato:

```
TABLE nome_tabella COLUMNS nome_colonna1,nome_colonna2,…;
```

1 - nome_tabella.txt

Esempi

Di seguito alcuni esempi di creazione di tabelle:

- `CREATE TABLE rubrica (cognome,nome,telefono)`
- `CREATE TABLE studente (matricola,nome,cognome,età)`

3.2.2 Inserimento di Nuovi Dati nella Tabella

Le tabelle appena create non contengono dati, questi vengono inseriti per righe, una alla volta, utilizzando la query seguente:

```
INSERT INTO nome_tabella (nome_colonna1,nome_colonna2,…) VALUES (valore1,valore2,…)
```

Il numero di valori interni alle parentesi **DEVE** corrispondere alle colonne dichiarate della tabella al momento della creazione. I nomi delle colonne indicate in questa query devono essere **uguali e nello stesso ordine** dei nomi delle colonne dichiarate alla creazione.

Per ogni query di inserimento eseguita, si controlli che esista la tabella e che non ci siano errori come indicato sopra.

Se non ci sono errori, si aggiunga in coda al file di testo una riga formattata nel seguente modo:

```
ROW valore1,valore2,valore3,…;
```

Esempi

Riprendendo gli esempi della sezione precedente, ecco alcuni esempi di inserimento dati:

- `INSERT INTO rubrica (cognome,nome,telefono) VALUES (Bianchi,Mario,3333333333)`
- `INSERT INTO rubrica (cognome,nome,telefono) VALUES (Rossi,Paolo,4444444444)`
- `INSERT INTO studente (matricola,nome,cognome,età) VALUES (1234,Mario,Rossi,21)`

In seguito a questi esempi, i file di testo *rubrica.txt* e *studente.txt* dovrebbero contenere questo testo:

```
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Bianchi,Mario,3333333333;
ROW Rossi,Paolo,4444444444;
```

2 - rubrica.txt

```
TABLE studente COLUMNS matricola,nome,cognome,età;
ROW 1234,Mario,Rossi,21;
```

3 - studente.txt

3.2.3 Ricerca all'interno delle tabelle

Quando si effettua una ricerca, di qualsiasi tipo (*con o senza filtri*) su una qualsiasi tabella, occorre aggiornare (*o creare*) un file di testo chiamato "*query_results.txt*". Sarà presente **UN UNICO** file per tutto il DB che racchiuderà tutti i risultati di tutte le query eseguite. Il formato da utilizzare è il seguente:

```
<QUERY ESEGUITA>;
TABLE nome_tabella COLUMNS nome_colonna1,nome_colonna2,…;
ROW valore1,valore2,…;
ROW valore1,valore2,…;
…
```

4 – query_results.txt

Per eseguire una ricerca si utilizza il comando **SELECT.**

Per le ricerche con filtri è possibile utilizzare **UN SOLO TIPO DI FILTRO** alla volta.

Ricerca senza filtri – 1° Versione

Con questa 1° versione dell'operazione si restituiscono come risultato tutti i dati della tabella.

Per selezionare tutti i dati di una tabella (*tutte le righe e tutte le colonne*) si utilizza l'operatore '*' come segue:

```
SELECT * FROM nome_tabella
```

Esempio

A partire dalla tabella in figura 2, si otterranno esattamente tutti i dati della tabella.

```
SELECT * FROM rubrica
```

Eseguendo questa query si ottiene un file di questo tipo:

```
SELECT * FROM rubrica;
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Bianchi,Mario,123456;
ROW Rossi,Paolo,43215;
ROW Verdi,Chiara,654654;
```

5 – query_results.txt

Ricerca senza filtri – 2° Versione

La 2° versione dell'operazione consente di specificare quali colonne si vogliono restituire come risultato della **SELECT**.

```
SELECT nome_colonna1,nome_colonna3,… FROM nome_tabella
```

Esempio

A partire dalla tabella in figura 2 e dalla seguente query:

```
SELECT cognome FROM rubrica
```

si ottiene un aggiornamento del file di questo tipo:

```
SELECT * FROM rubrica;
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Bianchi,Mario,123456;
ROW Rossi,Paolo,43215;
ROW Verdi,Chiara,654654;

SELECT cognome FROM rubrica;
TABLE rubrica COLUMNS cognome;
ROW Bianchi;
ROW Rossi;
ROW Verdi;
```

6 – query_results.txt

Mentre eseguendo questa seconda query:

```
SELECT cognome, telefono FROM rubrica
```

Il file si aggiorna nuovamente in questo modo:

```
SELECT * FROM rubrica;
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Bianchi,Mario,123456;
ROW Rossi,Paolo,43215;
ROW Verdi,Chiara,654654;

SELECT cognome FROM rubrica;
TABLE rubrica COLUMNS cognome;
ROW Bianchi;
ROW Rossi;
ROW Verdi;

SELECT cognome,telefono FROM rubrica;
TABLE rubrica COLUMNS cognome,telefono;
ROW Bianchi,123456;
ROW Rossi,43215;
ROW Verdi,654654;
```

7 – query_results.txt

Ricerca con filtri – Filtro WHERE

Il primo tipo di filtro che può essere applicato alla **`SELECT`** è il filtro **`WHERE`** che permette di selezionare solo i dati che rispettano una condizione, come di seguito:

```
SELECT … FROM nome_tabella WHERE condition
```

La **`condition`** può essere espressa con uno dei seguenti cinque operatori visti in linguaggio ANSI C:

- `==`
- `>`
- `>=`
- `<`
- `<=`

Esempio

A partire dalla tabella in figura 1 e figura 2 e dalle seguenti query:

```
SELECT * FROM rubrica WHERE cognome == Rossi

SELECT * FROM studente WHERE età >= 21

SELECT cognome,nome FROM studente WHERE età == 21
```

si ottiene un aggiornamento del file di questo tipo:

```
…

SELECT * FROM rubrica WHERE cognome == Rossi;
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Rossi,Paolo,43215;

SELECT * FROM studente WHERE età >= 21;
TABLE studente COLUMNS matricola,nome,cognome,età;
ROW 12345,Mario,Rossi,21;
ROW 123123,Giovanni,Verdi,21;
ROW 1231,Paolo,Rossi,25;

SELECT nome,cognome FROM studente WHERE età == 21;
TABLE studente COLUMNS nome,cognome;
ROW Mario,Rossi;
ROW Giovanni,Verdi;
```

8 – query_results.txt

Ricerca con filtri – Filtro di Ordinamento ORDER BY

Il secondo tipo di filtro che può essere applicato alla **SELECT** è il filtro **ORDER BY** che permette di ordinare i dati utilizzando una colonna della tabella come "pivot", come di seguito:

```
SELECT … FROM nome_tabella ORDER BY nome_colonna ASC
```

Utilizzando il filtro di ordinamento è possibile avere i risultati ordinati (*in ordine crescente o decrescente*) in base alla colonna specificata. Per scegliere se ordinare in ordine crescente o decrescente bisogna specificarlo in fondo alla query indicando **ASC** (*crescente*) o **DESC** (*decrescente*).

Esempio

A partire dalla tabella in figura 2 e dalle seguenti query:

```
SELECT * FROM rubrica ORDER BY nome ASC

SELECT * FROM rubrica ORDER BY cognome DESC

SELECT nome,telefono FROM rubrica ORDER BY nome DESC
```

si ottiene un aggiornamento del file di questo tipo:

```
…

SELECT * FROM rubrica ORDER BY nome ASC;
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Verdi,Chiara,654654;
ROW Bianchi,Mario,123456;
ROW Rossi,Paolo,43215;

SELECT * FROM rubrica ORDER BY cognome DESC;
TABLE rubrica COLUMNS cognome,nome,telefono;
ROW Verdi,Chiara,654654;
ROW Rossi,Paolo,43215;
ROW Bianchi,Mario,123456;

SELECT cognome,telefono FROM rubrica ORDER BY nome DESC;
TABLE rubrica COLUMNS cognome,telefono;
ROW Rossi,43215;
ROW Bianchi,123456;
ROW Verdi,654654;
```

9 – query_results.txt

Ricerca con filtri – Filtro di Raggruppamento GROUP BY

Il terzo tipo di filtro che può essere applicato alla **SELECT** è il filtro **GROUP BY** che permette di raggruppare i dati utilizzando una colonna della tabella come "pivot", introducendo nel file di testo con il risultato della query una nuova colonna denominata **COUNT** utile a indicare quante righe si stanno aggregando, come di seguito:

```
SELECT nome_colonnaX FROM nome_tabella GROUP BY nome_colonnaX
```

Utilizzando il filtro di raggruppamento è possibile contare quante occorrenze di uno stesso dato sono presenti all'interno della tabella.

N.B. per poter utilizzare questo filtro occorre selezionare solo una delle colonne della tabella.

Esempio

A partire dalla tabella in figura 1 e dalle seguenti query:

```
SELECT cognome FROM studenti GROUP BY cognome

SELECT età FROM studenti GROUP BY età
```

si ottiene un aggiornamento del file di questo tipo:

```
…

SELECT cognome FROM studente GROUP BY cognome;
TABLE studente COLUMNS cognome,COUNT;
ROW Rossi,2;
ROW Bianchi,1;
ROW Verdi,1;

SELECT età FROM studente GROUP BY età;
TABLE età COLUMNS età,COUNT;
ROW 21,2;
ROW 20,1;
ROW 25,1;
```

10 – query_results.txt