

Esercizi Programmatori Mainframe

Contents

1. [COBOL](#)
2. [SQL](#)
3. [SQL - COBOL](#)
4. [TSO - 3270](#)

COBOL

Contents

1. [Easy](#)
2. [Medium](#)
3. [Hard](#)

Cobol - Easy

1. Creare un programma COBOL che accetti 3 input (che il programma legge tramite **ACCEPT**): due numeri interi ed una operazione (+, -, *, /). In base all'operazione fornita in input, il programma ritorna in output il corretto risultato dato dalla applicazione dell'operazione ai due dati numerici.

Esempio:

```
input:
12
10
+

output:
22
```

Considerare gli input numerici come massimo di 3 cifre.

Tenere inoltre presente che:

1. Con le divisioni, l'output può essere un numero non-intero e non è possibile dividere per zero.
 2. Con le sottrazioni, l'output può essere negativo.
2. Creare un programma COBOL che dato un numero intero in input (che il programma legge tramite **ACCEPT**) calcola il fattoriale.
 3. Creare un programma COBOL che dati due numeri in input (che il programma legge tramite **ACCEPT**) calcola il coefficiente binomiale ($C(n,k) = n! / k!(n-k)!$). Considerare il primo numero fornito come **n** ed il secondo come **k**. Il programma deve scrivere in output il coefficiente.
 4. Creare un programma COBOL che data una stringa in input (letta tramite **ACCEPT**), mi scriva in output i corrispettivi valori in ASCII dei suoi caratteri. Considerare che la stringa può essere al massimo lunga 100 caratteri e quando il programma incontra uno spazio, interrompe l'esecuzione perchè per lui lo spazio è sinonimo di fine della stringa (se dovessero esserci altri caratteri dopo lo spazio... come si dice qua a Bologna: *Polleg* - o come direbbe un nostro famoso personaggio contemporaneo: *doveva anna' così fratelli*).

Consiglio: Utilizzare la funzione **ORD** del COBOL, tenendo presente che questa ritorna il numero ordinale in cui sono definiti i caratteri in ASCII. Ciò vuol dire che, se il primo valore in ASCII è uno zero **0** (in decimale) ed ha significato di **null**, il suo corrispettivo numero ordinale (dunque quello fornito dalla funzione **ORD**) sarà **1**. Il carattere **A** in ASCII è **65** ed il corrispettivo numero ordinale sarà **66** (not that hard).

5. Creare un programma COBOL in cui si definisce e si riempie dinamicamente un'array (**table** in COBOL) con tutte le lettere dell'alfabeto (prima maiuscole poi minuscole). Le lettere non devono essere *hard-coded* ossia **non** deve essere presente una cosa del genere:

```
MOVE 'A' TO alfabeto(1)
MOVE 'B' to alfabeto(2)
.
.
.
```

Consiglio: la funzione **CHAR** del cobol prende in input un numero ordinale e ritorna in output il corrispettivo ASCII.
Ossia se le si da in pasto **66** mi tirerà fuori **A**.


```

06270000001EUR200N0000044
06270000002EUR500N0000412
06270000002EUR500N0000613
06270000002EUR500N0000713
06270000002EUR200N0000042
06270000002EUR200N0000038

```

Output:

	Abi		Stabilimento		Taglio		Totale	

	05080		000001		500		0000053	
	05080		000001		200		0000019	

	Totale Stab		000001				0000052	

	05080		000002		500		0001145	
	05080		000002		200		0000970	

	Totale Stab		000002				0002115	

	Totale Abi		05080		500		0001178	
	Totale Abi		05080		200		0000989	

Abi	Stabilimento	Taglio	Totale	

06270	000001	500	0000023	
06270	000001	200	0000059	

Totale Stab		000001	0000082	

06270	000002	500	0001738	
06270	000002	200	0000080	

Totale Stab		000002	0001818	

Totale Abi		06270	500	0001761
Totale Abi		06270	200	0000139

Esercizio 2

Considerare il file di input `file02.txt`. Ogni record (riga) è strutturato nel modo seguente:

posizione inizio	lunghezza	significato
1	12	numero identificativo dell'elaborazione
13	18	timestamp inizio della elaborazione
31	8	ora fine della elaborazione

Il formato dei Timestamp (che è una `data` + `orario`) è il seguente (che poi è praticamente lo stesso formato dei timestamp in MySQL):

```
AAAA-MM-GGHH:MM:SS
```

Quello che bisogna fare è calcolare quanti secondi è durata ogni elaborazione (dunque per ogni record). Per semplicità, ogni elaborazione sarà avvenuta all'interno della stessa giornata e l'output, dato il seguente input:

```
1234561234562020-02-0112:23:0012:27:22  
1234571234572020-02-0105:11:0205:12:54
```

dovrà avere la seguente forma (scritto in un file):

```
01/02/2020  
Elaborazione: 123456123456  
Durata (s): 00322  
  
02/03/2020  
Elaborazione: 123457123457  
Durata (s): 00112
```

Esercizio 3

Come miglioramento dell'esercizio precedente (2) considerare il file `file03.txt` e tenendo a mente le stesse direttive, raggruppare sotto la stessa data tutte le elaborazioni avvenute in quella giornata (il file non sarà ordinato per data) e scrivere il totale delle durate.

Esempio:

Input:

```
1234261234562020-02-0112:23:0012:27:22
1334571234562020-03-0105:11:0205:12:54
1234571234562020-02-0105:11:0205:18:54
1231574242422020-03-0105:11:0205:12:54
1244571212222020-02-0105:11:0205:12:54
1254571299762020-02-0105:11:0205:14:54
```

Output:

01/02/2020

```
-----
| Elaborazione | Durata |
|-----|
| 123426123456 | 00322 |
| 123457123456 | 00472 |
| 124457121222 | 00112 |
| 125457129976 | 00232 |
| TOTALE       | 00001138 |
-----
```

03/02/2020

```
-----
| Elaborazione | Durata |
|-----|
| 133457123456 | 00112 |
| 123157424242 | 00112 |
| TOTALE       | 00000224 |
-----
```

Tenere a mente che la somma di svariate durate che possono essere interi di 5 cifre (in quanto in una giornata ci sono al massimo $24*60*60 = 86400$ secondi), può risultare in un intero che sfora anche le 6 cifre.

Esercizio 4

Considerare il file `copy04.cpy`. Questa è una **COPY** ossia contiene un tracciato che rappresenta la definizione dei record per certi file sequenziali, le copy vengono usate per avere consistenza in differenti programmi. Per esempio, un programma COBOL mi può scrivere certi record che per esempio hanno il seguente formato:

```
01 tracciato.
   05 tracciato-abi      pic X(05).
   05 tracciato-num-stab pic X(06).
   05 tracciato-num-cart pic X(18).
```

In questo caso, un file sequenziale i quali record sono creati usando questo formato, potrebbe avere la seguente forma:

```
05080000001123456123456123456
05080000002123456123456323456
05080000003123456123456623456
05080000004123456123426123456
05080000005123456123446123456
05080000006123456123434123456
05080000007123456123454123456
```

Dunque, un altro programma COBOL che si dovesse trovare a leggere i record da questo file, userebbe la seguente sintassi:

```
...
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT INFILE ASSIGN TO "infile.txt"
           FILE STATUS IS fs-infile
...
WORKING-STORAGE SECTION.
...
COPY "copy.cpy".
...
PROCEDURE DIVISION.
...
   READ INFILE INTO tracciato
...
   STOP RUN.
```

In questo modo, non dovremo "indovinare" i campi del file `infile.txt` ma ci basterà usare la **COPY** (chiaramente bisogna essere al corrente del fatto che quel dato file di records è stato creato utilizzando quella data **COPY**).

Detto ciò, utilizzando la **COPY** `copy04.cpy` che rappresenta i record del file `file04.txt`, per ogni record bisogna creare un messaggio **XML** nel seguente formato:

```
<message>
  <from>*****</from>
  <to>----</to>
  <transactions>
```

```

    <amount id="%%%%" iso="$$$">#####</amount>
    <amount id="%%%%" iso="$$$">#####</amount>
  </transactions>
</message>

```

dove bisogna andare a mettere al posto dei simboli il contenuto dei corrispettivi campi:

Simbolo	Campo nella copy
*****	tracciato-from
-----	tracciato-to
%%%%%	tracciato-trx-id(i)
\$\$\$	tracciato-trx-iso(i)
#####	tracciato-trx-amount(i)

dove `tracciato-trx-id(i)`, `tracciato-trx-iso(i)` e `tracciato-trx-amount(i)` rappresentano l'i-esimo elemento dell'array (`table`) `tracciato-transactions`. Le transazioni, come è apprezzabile dalla `COPY`, possono essere al massimo 5 (e ve ne sarà almeno 1), nei record di input le transazioni non presenti saranno spazi e bisognerà aggiungere gli elementi `<amount>` solo per quelle transazioni che sono specificate.

Il file di output sarà una cosa del genere:

```

<message>
  <from>COMMAND FEED</from>
  <to>ALoud ATE</to>
  <transactions>
    <amount id="ec9b7b" iso="DLR">469</amount>
    <amount id="edc07c" iso="PND">183</amount>
    <amount id="288bf5" iso="EUR">51</amount>
  </transactions>
</message>
<message>
  ...
</message>
.
.
.

```

Notare che:

- Le stringhe `from` e `to` sono state 'trimmate' ossia sono stati rimossi gli spazi in eccesso (COBOL ha una funzione che fa questo: `TRIM`)
- Gli zero non significativi davanti nei valori `amount` sono stati rimossi.

COBOL - Hard

JSON Parser

Questo esercizio sarà diviso in vari step, permettendo di ragionare su differenti implementazioni.

Bisogna dunque creare un programma che faccia un 'parse' di un file in formato **JSON**. JSON sta per **JavaScript Object Notation** ed è un formato 'human readable' per salvare e passare dati in formato **key-value**. E' un'alternativa agli **XML** come mezzo di passaggio di informazioni. Esempio di JSON:

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "età": 27,
  "laurea": true,
  "ubicazione": {
    "via": "via dalle palle",
    "civico": 70,
    "città": "BOLOGNA",
    "cap": "40127",
  }
}
```

Come si può notare, un JSON può contenere diverse 'data structures' in particolare **string**, **number**, **boolean**, **object**, etc.. Tecnicamente non ci sarebbe limite alla 'profondità' nei vari 'nested objects', tuttavia è abbastanza chiaro che per applicazioni reali non si avranno mai troppi livelli (> 100 ?) in quanto inizierebbe a perdere di significato sia il suo carattere 'human readable' sia il fatto che per accedere a questi livelli bisognerebbe avere una serie di riferimenti ad elementi dentro oggetti del tipo **object.key1.key2.key3.key4...** (chiaramente in una applicazione reale, si definirebbero tanti oggetti - in punti diversi del programma in base alla necessità - del tipo: **value1 = object.key1; value2 = value1.key2; value3 = value2.key3; ...**).

Il programma che si andrà a costruire sarà composto da una **copy** di **working-storage** ed una **copy** di **procedure**. La copy di working-storage conterrà le variabili utilizzate dalle routine nella copy di procedure e la copy di procedure conterrà le routine per eseguire il parse ed eventuali altri metodi utili.

STEP 1

Per questa prima versione di **Parser**, implementare la lettura di JSON che **non contengono** 'nested objects' quindi saranno 'plain key-value objects' (non vi saranno altri oggetti annidati). I possibili tipi di valori associati alle chiavi saranno:

```
string
numeric
boolean
```

Dunque un JSON in input potrebbe essere:

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "età": 27,
  "laurea": true
}
```

Nota: Il nome della copy di working-storage dovrà essere **MNCWJSPA**.

Ogni variabile definita in questa copy dovrà essere chiamata con il prefisso **mncwjspa-**.

Nota: Il nome della copy di procedure dovrà essere **MNCPJSPA**.

Ogni metodo (routine) definito in questa copy, dovrà essere chiamato con il prefisso **mncpjspa-**.

NOTA SULLA NOMENCLATURA

MN --> Monetica (uno dei reparti CSE)

C --> Copy

W/P --> Working-Storage / Procedure

JSPA --> JSon PARser

Nota: Considerare come massimo numero di key-value possibili: 100

Nota: Considerare il file **.json** in input sempre formattato come l'esempio sopra. Dunque **non** considerare JSON del seguente tipo:

```
{ "nome": "JOHN", "cognome": "DOE", "età": 27, "laurea": true }
```

La copy di procedure dovrà contenere i seguenti metodi:

metodo	funzionalità
mncpjspa-parse-json	Questa routine andrà a salvarsi nella table mncwjspa-key-value le chiavi con i relativi valori e i tipi dei valori (dunque si avranno dei sottolivelli di mncwjspa-key-value che conterranno queste variabili. Potete chiamarli mncwjspa-key , mncwjspa-value e mncwjspa-value-type)
mncpjspa-search-for-value	Questa routine va a cercare se la chiave salvata nella variabile mncwjspa-key-to-search compare nel JSON, e se compare, salva il suo valore nella variabile mncwjspa-value-returned ed il tipo di variabile (string, numeric, boolean) in mncwjspa-value-type-returned . Se non dovesse esserci la chiave, si dovrà fare una display: La chiave '<nome della chiave>' non esiste.
mncpjspa-display-table	Questa routine andrà a fare una display di tutte le coppie key-value nel seguente formato: key --> value

NOTA: Considerare **rec-infile** come **PIC X(100)**.

Esempi di output:

- Per la **mncpjspa-search-for-value**:

chiave esiste nel JSON:

```
Chiave trovata:
nome --> JOHN
```

chiave non esiste nel JSON:

```
La chiave 'hello' non esiste
```

- Per la `mncpjspa-display-table`:

```
nome --> JOHN
cognome --> DOE
età --> 27
maschio --> true
```

('età' verà sfasata come si può vedere (in particolare il carattere à), questo direi che dipenda da che tipo di `encoding` supporta opencobol, ma non ci interessa in questo momento).

- Per un formato JSON Errato:
 - Manca una virgola in una entrata in mezzo (la virgola può non esserci per l'ultimo valore):

input:

```
{
  "nome": "JOHN",
  "cognome": "DOE"
  "età": 27,
  "laurea": true
}
```

output:

```
Errore nel formato JSON:
Manca una virgola dopo il valore '"DOE"'
```

- Un valore non ha la sintassi giusta:

input:

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "età": 234uuu,
  "laurea": true
}
```

output:

```
Errore nel formato JSON:
Valore '27uuu' non valido
```

STEP 2

Con qualche modifica, a questo punto, bisogna portare il programma a fare il parse di JSON con oggetti annidati. Con parse, in questo step, intendiamo che il programma debba riuscire a leggere correttamente e salvare nelle apposite variabili di working-storage, gli elementi del JSON e per ogni elemento vogliamo che salvi il suo **parent**.

Esempio **file_hard02.json**:

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "età": 27,
  "maggiorenne": true,
  "ubicazione": {
    "1": {
      "via": "via dalle palle",
      "civico": "1",
      "cap": "40068",
      "città": "San lazzaro di savena"
    },
    "2": {
      "via": "piazza la bomba e scappa",
      "civico": "88",
      "cap": "40127",
      "città": "Bologna"
    }
  }
}
```

Per gli elementi **nome**, **cognome**, **età**, .. Il loro **parent** sarà settato a **NULL**, mentre per **via** il suo **parent** sarà **1** e per **1** sarà **ubicazione** e così via.

NOTA: Il parent sarà salvato per ogni key-value come un **POINTER** che punti alla chiave del parente. Dunque nella **table** di working-storage **mncwjspa-key-value**, vi sarà un sottolivello aggiuntivo: **mncwjspa-parent-ptr POINTER**.

Dopo aver fatto girare **mncpjspa-parse-json**, la routine **mncpjspa-display-table** dovrà produrre una cosa del genere:

```
nome --> JOHN   parent: root
cognome --> DOE  parent: root
età --> 27      parent: root
maggiorenne --> true  parent: root
ubicazione -->      parent: root
1 -->      parent: ubicazione
via --> via dalle palle  parent: 1
civico --> 1      parent: 1
cap --> 40068     parent: 1
città --> San lazzaro di savena  parent: 1
2 -->      parent: ubicazione
via --> piazza la bomba e scappa  parent: 2
civico --> 88     parent: 2
cap --> 40127     parent: 2
città --> Bologna  parent: 2
```

STEP 3

A questo punto, bisogna modificare il metodo `mncpjspa-search-for-value` permettendogli di accettare input (nella variabile `mncwjspa-key-to-search`) del seguente tipo:

```
<key_1>.<key_2>.<key_3>...<key_n>
```

Come per un oggetto si accede ai suoi attributi con il punto `.`, qui cerchiamo di riprodurre la stessa cosa.

Esempio:

Considerare il JSON:

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "età": 27,
  "maggiorenne": true,
  "ubicazione": {
    "1": {
      "via": "via dalle palle",
      "civico": "1",
      "cap": "40068",
      "città": "San lazzaro di savena"
    },
    "2": {
      "via": "piazza la bomba e scappa",
      "civico": "88",
      "cap": "40127",
      "città": "Bologna"
    }
  },
  "istruzione": {
    "laurea": {
      "primo_livello": {
        "università": "Università degli studi di bologna",
        "corso": "fisica"
      },
      "secondo_livello": {
        "università": "Università degli studi di bologna",
        "corso": "fisica teorica"
      }
    }
  }
}
```

Vogliamo andare a cercare con la routine `mncpsjspa-search-for-value` il valore associato alla seguente chiave:

```
istruzione.laurea.primo_livello.corso
```

La routine dovrà ritornare nella variabile `mncwjspa-returned-value` il valore `fisica`.

Nel caso in cui non esista quella data chiave, il programma dovrà ritornare un messaggio di errore.

Nota: Se non già abbastanza capiente, aumentare la grandezza della variabile `mncwjspa-key-to-search` ad almeno `PIC X(10000)` per permettere di cercare valori annidati sufficientemente profondi (100 livelli possibili per 100 bytes massimi di ogni chiave (si potrebbe/dovrebbe abbassare il numero massimo di bytes per ogni chiave a tipo 30-50, ma vabbè, non è problema in questo momento)).

Nota: Sarà un 'degenero' di puntatori, probabilmente ci sarà un soluzione migliore, ma quella che ho trovato io consiste nel definire due puntatori nella `table`:

```
mncwjspa-key-value
```

uno che punta alla chiave del parent (`mncwjspa-parent-ptr`) ed uno che punta al puntatore del parent (`mncwjspa-parent-ptr-ptr`). Inoltre vi serviranno altri due puntatori: `mncwjspa-current-ptr` e `mncwjspa-current-ptr-ptr`. Infine, dato che non si può settare l'`ADDRESS` di variabili in working-storage, vi serviranno anche due puntatori nella `LINKAGE SECTION` (del main program): `mncwjspa-current-ptr-1` e `mncwjspa-current-ptr-ptr-1` più una variabile che servirà per accedere alla chiave puntata dal puntatore: `mncwjspa-current-linkage`.

Utilizzando il JSON sopra, potremo interrogare il programma per:

```
ubicazione.2.cap
```

che dovrà ritornare:

```
40127
```

oppure:

```
istruzione.laurea.primo_livello.corso
```

che dovrà ritornare:

```
fisica
```

Nota: Se la chiave che si cerca ha come valore un oggetto, in questo step, nel valore ritornato, va bene metterci spazi.

Esempio: se si riempie la variabile `mncwjspa-key-to-search` con `istruzione.laurea` e poi si fa girare la `mncpjspa-search-for-key`, la variabile `mncwjspa-returned-value` dovrà contenere spazi.

STEP 4

In questo step andremo ad implementare la possibilità di cercare chiavi che hanno come valori degli oggetti. Dunque, se per esempio avessimo il seguente JSON:

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "età": 27,
  "maggiorenne": true,
  "ubicazione": {
    "1": {
      "cap": "40068",
      "città": "San lazzaro di savena",
      "civico": "1",
      "via": "via dalle palle"
    },
    "2": {
      "cap": "40127",
      "città": "Bologna",
      "civico": "88",
      "via": "piazza la bomba e scappa"
    }
  }
}
```

Vogliamo fare in modo che la routine `mncpjspa-search-for-value` quando il campo `mncpjspa-key-to-search` è riempito nel seguente modo:

```
MOVE 'ubicazione.1` TO mncpjspa-key-to-search
```

mi ritorni tutte le chiavi + valori del dato oggetto. In che senso deve ritornare questi elementi? Si definisca una nuova table, chiamata `mncwjspa-returnd-obj` che conterrà come sottolivelli `mncwjspa-ret-key`, `mncwjspa-ret-value` e `mncwjspa-ret-value-type` in modo che questa table, alla fine della routine `mncpjspa-search-for-value`, dovrà contenere i seguenti valori:

```
cap --> 40068 --> string
città --> San lazzaro di savena --> string
civico --> 1 --> string
via --> via dalle palle --> string
```

Altro esempio:

Considerare il seguente JSON:

```
{
  "istruzione": {
    "laurea": {
      "primo_livello": {
        "università": "Università degli studi di bologna",
        "corso": "fisica"
      }
    }
  }
}
```

```
    },  
    "secondo_livello": {  
        "università": "Università degli studi di bologna",  
        "corso": "fisica teorica"  
    }  
  }  
}
```

se cerchiamo la seguente chiave:

```
MOVE 'istruzione.laurea' TO mncwjspa-key-to-search
```

la routine dovrà mettere i seguenti valori nella table:

```
primo_livello --> --> object  
secondo_livello --> --> object
```

dunque, chiaramente, se i valori ritornati sono degli oggetti, nel campo `mncpjspa-ret-value` si lasceranno degli spazi.

STEP 5

In questo step andremo a rifinire la gestione degli errori e del 'return' dei valori.

Definiremo una nuova variabile che verrà riempita in base all'esito delle varie elaborazioni:

```
01 mncwjspa-esito      PIC 9(4).
```

Ed un'altra variabile che verrà riempita con la descrizione dell'errore:

```
01 mncwjspa-err-desc   PIC X(1000).
```

Andremo a definire dei codici numerici che rappresenteranno ognuno un certo tipo di errore.

Per la routine `mncpjspa-parse-json` avremo i seguenti possibili esiti:

Codice	Codifica
0	Parse andato a buon fine
10	Mancanza della graffa { ad inizio JSON
20	Mancanza di una virgola , a fine di una riga che lo necessita
30	Mancanza di una graffa } in chiusura di un oggetto
40	Formato sbagliato di una chiave (mancano virgolette ")
50	Formato sbagliato di un valore nel JSON
60	Presenza di una virgola , dove non dovrebbe esserci

Per la routine `mncpjspa-search-for-key` avremo i seguenti possibili valori:

Codice	Codifica
0	Chiave trovata correttamente
100	Non è stato ancora fatto il parse del JSON
110	La chiave cercata non esiste

Esempi

```
"nome": "JOHN",
"cognome": "DOE",
"età": 27
}
```

variabile	contenuto
mncwjspa-esito	10
mncwjspa-err-desc	Manca la parentesi graffa in apertura del JSON

```
{
  "nome": "JOHN"
  "cognome": "DOE",
  "età": 27
}
```

variabile	contenuto
mncwjspa-esito	20
mncwjspa-err-desc	Manca una virgola nella riga 2

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "yo": {
    "hello": "world"
  }
}
```

variabile	contenuto
mncwjspa-esito	30
mncwjspa-err-desc	Manca una parentesi graffa in chiusura oggetto

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  yo: {
    "hello": "world"
  }
}
```

variabile	contenuto
mncwjspa-esito	40
mncwjspa-err-desc	Errore nel formato della chiave nella riga 4

```
{
  "nome": "JOHN",
  "cognome": "DOE",
  "yo": {
    "hello": "world"sdsdsd
  }
}
```

variabile	contenuto
-----------	-----------

variabile	contenuto
mncwjspa-esito	50
mncwjspa-err-desc	Errore nel formato del valore nella riga 5

```
{  
  "nome": "JOHN",  
  "cognome": "DOE",  
  "yo": {  
    "hello": "world"  
  },  
}
```

variabile	contenuto
mncwjspa-esito	60
mncwjspa-err-desc	Virgola in più nella riga 6

STEP 6

In questo step andremo ad implementare la possibilità di scrivere in output un JSON.

In questo step andremo ad implementare la possibilità di scrivere JSON senza oggetti annidati. Successivamente andremo ad implementare anche quella funzionalità.

Avremo dunque una nuova **table** che conterrà il JSON che successivamente andremo a scrivere. Questa table dovrà essere chiamata:

```
mncwjspa-json-out
```

E dovrà avere i seguenti sottolivelli:

Campo	Contenuto
mncwjspa-out-key	Chiave
mncwjspa-out-value	Valore associato alla chiave, sarà SPACES per una chiave che è un object
mncwjspa-out-value-type	Tipo della chiave: string , numeric , boolean or object come in precedenza

Inoltre dovranno essere definite anche le seguenti variabili (la prima servirà per i successivi step):

```
01 mncwjspa-out-level PIC X(10000).

01 mncwjspa-current-out.
  05 mncwjspa-c-out-key          PIC X(100).
  05 mncwjspa-c-out-value        PIC X(100).
  05 mncwjspa-c-out-value-type
```

Variabile	Contenuto
mncwjspa-out-level	Dove si specificherà il livello nel quale si vuole inserire la key-value che si specificherà negli altri campi
mncwjspa-out-key	Si specifica la chiave che si vorrà scrivere
mncwjspa-out-value	Conterrà il valore della chiave
mncwjspa-out-value-type	Conterrà il tipo del valore

NOTA: quando si vorrà inserire una key-value nel root, bisognerà mettere **SPACES** nella variabile **mncwjspa-out-level**. Dunque in questo step, quella variabile dovrà essere inizializzata con **SPACES** e non si dovranno considerare altri casi.

In conclusione dovranno esserci i seguenti metodi:

Metodo	Funzionalità
mncpjspa-fill-out-level	Prenderà la chiave, valore e tipo in mncwjspa-current-out e andrà a metterli nella table mncwjspa-json-out considerando che questo valore ha come parente il valore in mncwjspa-out-level

Metodo	Funzionalità
<code>mncpjspa-write-json</code>	Scriverà nel file di output <code>OUTFILE</code> il json che è salvato nella variabile <code>mncwjspa-json-out</code> con la corretta formattazione

ESEMPIO

```
MOVE SPACES    TO mncjwspa-out-level

MOVE "nome"    TO mncwjspa-c-out-key
MOVE "John"    TO mncwjspa-c-out-value
MOVE "string"  TO mncwjspa-c-out-value-type
PERFORM mncpjspa-fill-out-level

MOVE "cognome" TO mncwjspa-c-out-key
MOVE "Doe"     TO mncwjspa-c-out-value
PERFORM mncpjspa-fill-out-level

PERFORM mncpjspa-write-json
```

Questo dovrà produrre il seguente json:

```
{
  "nome": "John",
  "cognome": "Doe"
}
```

STEP 7

SQL

Contents

1. [Easy](#)
2. [Medium](#)
3. [Hard](#)

Risolvere gli esercizi utilizzando il database `sakila` in MySQL

SQL - Easy

Esercizio 1

Creare una Query che dato il nome e cognome di un attore, mi tiri fuori i titoli dei film in cui ha recitato.

Esempio:

Utilizzando `first_name = "JOE"` e `last_name = "SWANK"`, la Query dovrebbe ritornare:

```
ANYTHING SAVANNAH  
BIRCH ANTITRUST  
CHOCOLAT HARRY  
CHOCOLATE DUCK  
CROOKED FROGMEN  
CURTAIN VIDEOTAPE  
DALMATIONS SWEDEN  
HORROR REIGN  
LAWLESS VISION  
LEBOWSKI SOLDIERS  
MAJESTIC FLOATS  
PACIFIC AMISTAD  
PERDITION FARGO  
PRIMARY GLASS  
REEF SALUTE  
RUNNER MADIGAN  
SMILE EARRING  
SNATCHERS MONTEZUMA  
SUNRISE LEAGUE  
SWEETHEARTS SUSPECTS  
TIES HUNGER  
TRAFFIC HOBBIT  
UNTOUCHABLES SUNRISE  
WATERFRONT DELIVERANCE  
WILD APOLLO
```

Esercizio 2

Creare una Query che dato il nome e cognome di un attore, mi tiri fuori le lunghezze dei film che ha girato (chiaramente senza doppi) ordinate in modo decrescente.

Esempio:

Utilizzando `first_name = "JOE"` e `last_name = "SWANK"`, la Query dovrebbe ritornare:

```
181
162
144
143
139
135
133
132
130
123
120
111
108
106
101
99
82
74
69
61
60
53
```

Esercizio 3

Creare una Query che dato il nome e cognome di un cliente (**customer**) mi ritorni tutti i soldi spesi in noleggio (**rental**) da quel dato cliente.

Esempio:

Utilizzando **first_name = PATRICIA** e **last_name = JOHNSON**, la query dovrebbe ritornare:

128.73

Esercizio 4

Creare una Query che dato il nome e cognome di un cliente (**customer**) mi ritorni tutti i soldi spesi in noleggio (**rental**) da quel dato cliente che sono stati processati dal manager **Jon Stephens** (nella tabella **staff**).

Esempio:

Utilizzando **first_name = PATRICIA** e **last_name = JOHNSON**, la query dovrebbe ritornare:

67.88

Esercizio 5

Creare una Query che, tra i film in cui compaiono le scene tagliate (**Deleted Scenes** nella colonna **special_features**), mi estragga quello che è stato noleggiato per più tempo (conteggiare il tempo di noleggio in **ore**).

La query dovrebbe ritornare:

hours	title
4458	RIDGEMONT SUBMARINE

Tenere a mente che i film possono avere più copie, dunque ci possono essere più elementi nella tabella **inventory** associate allo stesso film. La relazione è dunque one-to-many per **film_id-to-inventory_id**

Esercizio 6

Creare una Query che ritorni il guadagno ricevuto dal noleggio dei film contenenti **Trailers** (come **special_features**). In output bisogna avere una tabella contenente in una colonna il titolo del film e nell'altra il guadagno totale dai noleggi.

La query dovrebbe ritornare:

guadagno	titolo
214.69	ZORRO ARK
198.72	TORQUE BOUND
191.74	INNOCENT USUAL
190.78	HUSTLER PARTY
180.71	ENEMY ODDS
179.73	RANGE MOONWALKER
175.77	FOOL MOCKINGBIRD
172.72	CLOSER BANG
...	...
...	...
...	...

Esercizio 7

Creare un Query che elenchi i clienti con i soldi spesi per il noleggio in base allo staff che li ha serviti (ordinati per nome e cognome e staff_id).

La query dovrebbe ritornare:

first_name	last_name	amount	staff_id
AARON	SELBY	46.90	2
AARON	SELBY	63.86	1
ADAM	GOOCH	51.89	1
ADAM	GOOCH	49.89	2
ADRIAN	CLARY	33.89	1
ADRIAN	CLARY	40.92	2
AGNES	BISHOP	53.87	2
AGNES	BISHOP	44.90	1
ALAN	KAHN	76.84	1
ALAN	KAHN	47.90	2
...
...
...

SQL - Medium

Esercizio 1

Creare una Query che ritorni i clienti (nome e cognome) che hanno noleggiato il film che è stato noleggiato più volte (non per più tempo) in cui siano presenti le scene tagliate (dovrebbe essere **BUCKET BROTHERHOOD** con 34 noleggi).

La Query dovrebbe ritornare (ordinato per nome, cognome)

```
ALFRED CASILLAS  
ALFREDO MCADAMS  
ANA BRADLEY  
ANTONIO MEEK  
ARMANDO GRUBER  
BEN EASTER  
BERNARD COLBY  
BOBBY BOUDREAU  
CHERYL MURPHY  
CLAYTON BARBEE  
DAN PAINE  
DANA HART  
DANNY ISOM  
DWIGHT LOMBARDI  
ENRIQUE FORSYTHE  
EVERETT BANDA  
JARED ELY  
JUAN FRALEY  
LORETTA CARPENTER  
MARIA MILLER  
MARIO CHEATHAM  
MONICA HICKS  
NANCY THOMAS  
NAOMI JENNINGS  
PATRICK NEWSOM  
RANDALL NEUMANN  
ROBERTO VU  
SETH HANNON  
TAMMY SANDERS  
TARA RYAN  
VINCENT RALSTON  
WILLIE MARKHAM  
YVONNE WATKINS
```

Nota: ANA BRADLEY ha noleggiato due volte questo film, quindi se dovesse comparire due volte è per quel motivo. L'output non dovrà contenere doppi, tuttavia.

Esercizio 2

Creare una Query che mostri quanto è stato guadagnato in noleggi giorno per giorno in modo progressivo in base ai giorni di pagamento (**payment_date** nella tabella **payment**).

La query dovrebbe ritornare:

date	total
2005-05-24	29.92
2005-05-25	603.55
2005-05-26	1357.81
2005-05-27	2043.14
2005-05-28	2847.18
2005-05-29	3495.64
2005-05-30	4124.06
2005-05-31	4824.43
2005-06-14	4882.27
2005-06-15	6258.79
...	...
...	...
...	...

Esercizio 3

Creare una Query che estragga le persone che hanno noleggiato film di categoria **Animation** e **Family** ordinati per nome, cognome, categoria e titolo del film.

La query dovrebbe ritornare:

first_name	last_name	title	category
AARON	SELBY	NETWORK PEAK	Family
AARON	SELBY	WILLOW TRACY	Family
ADAM	GOOCH	TITANIC BOONDOCK	Animation
ADAM	GOOCH	MUSIC BOONDOCK	Family
ADAM	GOOCH	SIEGE MADRE	Family
ADRIAN	CLARY	EFFECT GLADIATOR	Family
ADRIAN	CLARY	HALF OUTFIELD	Family
ADRIAN	CLARY	MAGUIRE APACHE	Family
ADRIAN	CLARY	ROBBERY BRIGHT	Family
AGNES	BISHOP	ANACONDA CONFESSIONS	Animation
...
...
...

Esercizio 4

Vogliamo una query che ritorni, per ogni film, il suo titolo e la lista delle date in cui è stato noleggiato (tenere a mente che ogni film può essere presente in più copie nell'inventario). Dovrà essere presente il numero della volte in cui il dato film è stato noleggiato (si dovrà ordinare il risultato per questo valore) più la lista delle date in cui è stato noleggiato:

title	count	rental_dates
HARDLY ROBBERS	4	2005-07-31,2005-08-22,2005-07-29,2005-08-18
MIXED DOORS	4	2005-07-08,2005-07-27,2005-08-21,2005-07-28
TRAIN BUNCH	4	2005-08-02,2005-08-23,2005-07-28,2005-08-22
BRAVEHEART HUMAN	5	2005-07-31,2005-08-19,2005-07-10,2005-08-01,2005-08-20
BUNCH MINDS	5	2005-07-09,2005-07-30,2005-08-21,2005-07-29,2005-08-18
...
...
...

Esercizio 5

Creare una Query che ritorni per ogni film, il suo titolo e la lista degli attori che vi hanno preso parte. Dovrà quindi esserci anche un contatore che mi contenga il numero di attori nella lista. La query dovrà ritornare i risultati ordinati in modo decrescente per questo contatore:

title	count	actors
LAMBS CINCINATTI	15	WOODY HOFFMAN,VAL BOLGER,REESE KILMER,JULIA BARRYMORE,MENA TEMPLE,CHRISTIAN NEESON,BURT POSEY,SCARLETT DAMON,WALTER TORN,CAMERON ZELLWEGER,LUCILLE DEE,FAY WINSLET,JAYNE NOLTE,MENA HOPPER,JULIA ZELLWEGER
BOONDOCK BALLROOM	13	HUMPHREY GARLAND,AUDREY BAILEY,ED CHASE,JENNIFER DAVIS,UMA WOOD,FRED COSTNER,KIRSTEN PALTROW,SANDRA PECK,DAN HARRIS,RAY JOHANSSON,KENNETH PESCI,CHRIS BRIDGES,WARREN JACKMAN
CHITTY LOCK	13	JOHNNY LOLLOBRIGIDA,LUCILLE TRACY,ELVIS MARX,SISSY SOBIESKI,VAL BOLGER,SUSAN DAVIS,RUSSELL TEMPLE,AL GARLAND,NICK DEGENERES,OLYMPIA PFEIFFER,LISA MONROE,HUMPHREY GARLAND,ROCK DUKAKIS
CRAZY HOME	13	SISSY SOBIESKI,WOODY JOLIE,MEG HAWKE,RUSSELL BACALL,MORGAN MCDORMAND,ALBERT NOLTE,CATE HARRIS,RUSSELL TEMPLE,VIVIEN BASINGER,HARVEY HOPE,WILL WILSON,ALAN DREYFUSS,GENE MCKELLEN
...
...
...

SQL - Hard

Esercizio 1

Creare una Query che ritorni il numero di noleggi progressivi effettuati da ogni membro dello staff giorno per giorno.

La query dovrebbe ritornare:

r_date	first_name	last_name	count
2005-05-24	Mike	Hillyer	5
2005-05-25	Mike	Hillyer	75
2005-05-26	Mike	Hillyer	153
2005-05-27	Mike	Hillyer	230
2005-05-28	Mike	Hillyer	318
...
...
2005-05-24	Jon	Stephens	3
2005-05-25	Jon	Stephens	70
2005-05-26	Jon	Stephens	166
2005-05-27	Jon	Stephens	255
...
...

Esercizio 2

Creare una funzione che dato in input un `film_id`, mi ritorni quanto è stato guadagnato grazie a quel film. Come calcolare questo valore? Per ogni giorno in cui è stato noleggiato, applicare il `rental_rate`. Bisogna poi vedere se vi sono stati dei ritardi nel ritornare i film. Per ogni giorno di ritardo, considerare 1 euro di addebito in più.

Esempi

Supponendo la funzione si chiami `earnings_by_film`, si dovrebbe avere:

```
earnings_by_film(1) --> 116.93
earnings_by_film(10) --> 496.04
earnings_by_film(12) --> 135.82
```

La seguente Query:

```
SELECT title, earnings_by_film(film_id) AS earnings
FROM film
ORDER BY earnings DESC
```

dovrà ritornare:

title	earnings
BUCKET BROTHERHOOD	799.42
MASSACRE USUAL	733.59
WITCHES PANIC	732.58
FORRESTER COMANCHEROS	680.66
CONFIDENTIAL INTERVIEW	673.69
LIES TREATMENT	646.73
...	...
...	...
...	...

Esercizio 3

Where are the people that rented the most rented movie?

Vogliamo una query che ritorni il titolo del film con un maggior numero di noleggi, più una lista delle città dei clienti che l'hanno noleggiato.

La query dovrebbe ritornare:

title	cities
BUCKET BROTHERHOOD	Oshawa,Almirante Brown,South Hill,Mukateve,Bag,al- Manama,Gorontalo,Teboksary,Changhwa,Serpuhov,Dhule (Dhulia),Yinchuan,Kirovo- Tepetsk,Kragujevac,Patras,Chatsworth,Kamyin,Ocumare del Tuy,Sawhaj,La Romana,Stockport,Southport,Allende,Hanoi,Mysore,Purwakarta,Memphis,Karnal,Bilbays,Memphis,Yamuna Nagar,Batna,Alvorada,Okara

Esercizio 4

Creare una query che ritorni per ogni film, il numero di ore totali in cui è stato noleggiato con la lista dei clienti che l'hanno preso. La query deve ritornare le colonne ordinate in maniera decrescente per il numero di ore (ricordarsi che ogni film è presente in più copie (ossia ad ogni film_id corrispondono più inventory_id)).

La query dovrebbe ritornare:

title	total_hours	customers
TRAIN BUNCH	320	BETTY WHITE,RHONDA KENNEDY,DAVID ROYAL,EDUARDO HIATT
DUFFEL APOCALYPSE	321	GLENN PULLEN,IVAN CROMWELL,TERRENCE GUNDERSON,BRIAN WYMAN,JUSTIN NGO,EDWIN BURK,SAMANTHA DUNCAN
FREEDOM CLEOPATRA	353	CLARA SHAW,LILLIAN GRIFFIN,GILBERT SLEDGE,MARVIN YEE,ALICIA MILLS
TRAFFIC HOBBIT	406	MAXINE SILVA,LEE HAWKS,CHRISTOPHER GRECO,JULIE SANCHEZ,JO FOWLER
INFORMER DOUBLE	406	DIANA ALEXANDER,MARGARET MOORE,ADAM GOOCH,HOLLY FOX,JEREMY HURTADO
...
...
...

SQL - COBOL

Contents

1. [Easy](#)
2. [Medium](#)

Risolvere gli esercizi utilizzando il database `sakila` in MySQL

SQL - COBOL - Easy

Esercizio 1

Creare un programma che estragga dal database tutti i clienti che hanno noleggiato almeno un film e creare un file in output dove i record hanno il seguente formato:

```
<nome><cognome><totale speso in noleggio>
```

Note:

- Mantenere lo stesso formato ('grandezza' delle variabili) delle colonne SQL (ossia, se la colonna `first_name` è un `varchar(45)`, la sua corrispondente variabile COBOL dovrà essere un `PIC X(45)`).
- I nomi delle colonne nel DB sono in **snake case**, ossia gli spazi sono sostituiti da *underscores* `_`, mentre le variabili COBOL sono in **kebab-case** (source <https://stackoverflow.com/questions/11273282/whats-the-name-for-hyphen-separated-case>) (lol) ossia gli spazi sono sostituiti da trattini (o *dashes* o *hyphens*) `-`.
- Il totale speso in noleggio può essere messo in un `PIC 9(04)V99`.
- I record dovranno essere ordinati per nome, cognome.

Esercizio 2

Creare un programma che mi legga dal file `file_sql02.txt` i record che avranno il seguente formato:

[INPUT]

posizione inizio	lunghezza	significato
1	45	nome dell'attore
46	45	cognome dell'attore

Il programma deve andare a vedere se nella tabella `actor`, l'attore letto dal file di input esiste. Nel file di output bisognerà scrivere una **S** se il record è in tabella oppure una **N** se il record non esiste nella tabella. Se il record esiste, ossia se l'attore è in tabella, bisognerà aggiungere anche il numero di film in cui il dato attore ha preso parte (se l'attore non è presente in tabella mettere `0`), il tutto secondo il seguente formato:

[OUTPUT]

posizione inizio	lunghezza	significato
1	45	nome dell'attore
46	45	cognome dell'attore
91	1	S o N
92	2	numero di film in cui l'attore ha preso parte

Esercizio 3

Come miglioramento dell'esercizio precedente (2), per ogni attore presente, invece di mettere il numero di film in cui il dato attore ha preso parte, mettere la lista dei suddetti film (titolo), separati da punto e virgola ; in una riga sotto. Il massimo numero di titoli che si deve mettere è 10 (prendere i primi ordinati in modo alfabetico). In base a questo, definire la corretta dimensione dei record del file di output (ossia nella **FILE SECTION**, definire il record in modo esatto - la massima lunghezza che può avere un record).

Note:

- Nel file di output, eliminare gli spazi finali in eccesso di ogni film.
- Se l'attore non è in tabella, non scrivere proprio la riga con i film

Esempio di output: `output02_example.txt`

Esercizio 4

Abbiamo un file (`file_sql04.txt`) in input i quali record sono nomi e cognomi di alcuni clienti. Vogliamo creare un programma che per ogni cliente, ci crei una mail (che sarà un file in output). La mail dovrà avere la seguente forma:

```
FROM:    <vostra email>
TO:      <email del cliente>
SUBJECT: Notifica dei pagamenti a vostro carico
BODY:
    Gentile <nome cliente>,

    Siamo qui a contattarvi per informarvi che i noleggi a vostro carico
    (già debitamente saldati) ammontano a euro:

        <totale pagato dal cliente>

    Vi ringraziamo per averci scelto e per il continuo supporto.

    Con i migliori auguri, vi mandiamo i nostri più cordiali saluti,
    Team noleggioFilm.it
```

NOTE:

- I record nel file di input saranno 10 (e dunque ci dovranno essere 10 file di output).

SQL - COBOL - Medium

Esercizio 1

Leggere il file di output generato dall'esercizio 3 della sezione [SQL-COBOL-Easy](#).

Per gli attori che sono nel database, bisognerà prendere i film specificati nel file e per ogni attore, il programma dovrà tirare fuori nel file di output una cosa del genere:

```
<NOME ATTORE>
FILM:          <NOME DEL FILM>
DESCRIZIONE:   <DESCRIZIONE DEL FILM>
NUMERO DI NOLEGGI: <NUMERO DI NOLEGGI>
---
FILM:          <NOME DEL FILM>
DESCRIZIONE:   <DESCRIZIONE DEL FILM>
NUMERO DI NOLEGGI: <NUMERO DI NOLEGGI>
---
FILM:          <NOME DEL FILM>
DESCRIZIONE:   <DESCRIZIONE DEL FILM>
NUMERO DI NOLEGGI: <NUMERO DI NOLEGGI>

<NOME ATTORE>
FILM:          <NOME DEL FILM>
DESCRIZIONE:   <DESCRIZIONE DEL FILM>
NUMERO DI NOLEGGI: <NUMERO DI NOLEGGI>
---
FILM:          <NOME DEL FILM>
DESCRIZIONE:   <DESCRIZIONE DEL FILM>
NUMERO DI NOLEGGI: <NUMERO DI NOLEGGI>
---
FILM:          <NOME DEL FILM>
DESCRIZIONE:   <DESCRIZIONE DEL FILM>
NUMERO DI NOLEGGI: <NUMERO DI NOLEGGI>
```

Dove chiaramente bisognerà andare a sostituire i placeholders <> con le relative stringhe.

Esercizio 2

Creare un programma che possa leggere file in input (un file XML) della seguente forma:

```
<messaggio>
  <email>mary.smith@sakilacustomer.org</email>
  [<amount />]
  [<rental />]
</messaggio>
```

Dove i tag tra parentesi quadre [] sono opzionali. In output dovrà esserci un altro file XML della seguente forma:

```
<output>
  <customer id="<ID del cliente>">MARY SMITH</customer>
  <address>1913 Hanoi Way</address>
  [<amount>118.68</amount>]
  [<rental>
    <title>PATIENT SISTER</title>
    <title>TALENTED HOMICIDE</title>
    <title>MUSKETEERS WAIT</title>
    <title>DETECTIVE VISION</title>
    <title>FERRIS MOTHER</title>
    <title>CLOSER BANG</title>
    <title>ATTACKS HATE</title>
    <title>SAVANNAH TOWN</title>
    <title>YOUTH KICK</title>
    <title>FIRE WOLVES</title>
    <title>SATURDAY LAMBS</title>
    <title>SNATCH SLIPPER</title>
    <title>CONFIDENTIAL INTERVIEW</title>
    <title>EXPECATIONS NATURAL</title>
    <title>LUCK OPUS</title>
    <title>DOORS PRESIDENT</title>
    <title>USUAL UNTOUCHABLES</title>
    <title>FROST HEAD</title>
    <title>WOMEN DORADO</title>
    <title>AMISTAD MIDSUMMER</title>
    <title>JEEPERS WEDDING</title>
    <title>ADAPTATION HOLES</title>
    <title>RACER EGG</title>
    <title>FINDING ANACONDA</title>
    <title>MINDS TRUMAN</title>
    <title>DALMATIONS SWEDEN</title>
    <title>PATIENT SISTER</title>
    <title>JUMANJI BLADE</title>
    <title>UNFORGIVEN ZOOLANDER</title>
    <title>FIREBALL PHILADELPHIA</title>
    <title>FIREBALL PHILADELPHIA</title>
    <title>BIKINI BORROWERS</title>
  </rental>]
</output>
```

Dove gli elementi tra parentesi dovranno essere presenti solo se i rispettivi tag sono presenti nel messaggio in input.

All'interno del tag `<amount>` nel file di output, dovrà esserci l'ammontare speso dal cliente in noleggi, mentre nei tag `<title>` all'interno del tag `<rental>` dovranno essere presenti i titoli dei film noleggiati dal suddetto cliente. Infine, nel tag `<address>` dovrà essere presente l'indirizzo di residenza del cliente.

TSO - 3270

Contents

- [Easy](#)
- [Medium](#)

TSO - Easy

Esercizio 1 - guidato

Seguire il file [create_first_program.pdf](#)

Esercizio 2

TSO - Medium