Query JQ per Dati Geografici Italiani

Struttura Dati

Il file JSON presenta la seguente struttura:

```
{
  "regioni": [
    {
      "_key": "nome_regione",
      "id": 123,
      "nome": "Nome Regione",
      "latitudine": 42.0,
      "longitudine": 13.0,
      "province": [
        {
          "_key": "nome_provincia",
          "id": 456,
          "nome": "Nome Provincia",
          "sigla_automobilistica": "XX",
          "latitudine": 42.0,
          "longitudine": 13.0,
          "comuni": [
            {
              "_key": "nome_comune",
              "id": 789,
              "nome": "Nome Comune",
              "codice_catastale": "A123",
              "latitudine": 42.0,
              "longitudine": 13.0
            }
          ]
        }
      ]
    }
  ]
}
```

PARTE 1: Query di Estrazione e Stampa

1.1 Estrarre tutti i comuni di una provincia specifica

```
jq '.regioni[].province[] | select(.nome == "Chieti") | .comuni[].nome' italy_fixed.json
```

Commento passo passo:

1. `.regioni[]` - Si itera su tutte le regioni dell'array principale
2. `.province[]` - Per ogni regione, si itera su tutte le sue province
3. `select(.nome == "Chieti")` - Si filtra selezionando solo la provincia con nome "Chieti"
4. `.comuni[]` - Si accede all'array dei comuni della provincia selezionata
5. `.nome` - Si estrae il campo "nome" di ogni comune
6. Il risultato è una lista di nomi di comuni, uno per riga

Esempio con sigla automobilistica:

```
jq '.regioni[].province[] | select(.sigla_automobilistica == "CH") | .comuni[].nome'
italy_fixed.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `.province[]` - Iterazione su tutte le province
3. `select(.sigla_automobilistica == "CH")` - Filtro per sigla automobilistica
4. `.comuni[].nome` - Estrazione dei nomi dei comuni

1.2 Estrarre i comuni con informazioni complete

```
jq '.regioni[].province[] | select(.nome == "Chieti") | .comuni[] | {nome, codice_catastale, id}'
italy_fixed.json
```

Commento passo passo:

1. `.regioni[].province[] | select(.nome == "Chieti")` - Selezione della provincia (come prima)
2. `.comuni[]` - Iterazione su tutti i comuni
3. `{nome, codice_catastale, id}` - Costruzione di un nuovo oggetto JSON contenente solo i campi specificati
4. Il risultato è una serie di oggetti JSON, uno per ogni comune

1.3 Trovare la regione che contiene una provincia con sigla automobilistica specifica

```
jq '.regioni[] | select(.province[].sigla_automobilistica == "RM") | .nome' italy_fixed.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `select(.province[].sigla_automobilistica == "RM")` - Filtro che verifica se almeno una provincia ha la sigla "RM"
3. `.nome` - Estrazione del nome della regione trovata

Versione alternativa con maggiore precisione:

```
jq '.regioni[] | select(any(.province[]; .sigla_automobilistica == "RM")) | .nome' italy_fixed.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `any(.province[]; .sigla_automobilistica == "RM")` - Funzione che verifica se esiste almeno una provincia con sigla "RM"
3. `select(...)` - Filtro che mantiene solo le regioni dove la condizione è vera
4. `.nome` - Estrazione del nome della regione

1.4 Contare il numero di comuni per provincia

```
jq '.regioni[].province[] | {provincia: .nome, numero_comuni: (.comuni | length)}' italy_fixed.json
```

Commento passo passo:

1. `.regioni[].province[]` - Iterazione su tutte le province di tutte le regioni
2. `{provincia: .nome, ...}` - Creazione di un oggetto con il nome della provincia
3. `(.comuni | length)` - Calcolo della lunghezza dell'array comuni (numero di elementi)
4. Il risultato è un oggetto JSON per ogni provincia con nome e conteggio comuni

1.5 Elencare tutte le province di una regione specifica

```
jq '.regioni[] | select(.nome == "Abruzzo") | .province[] | {nome, sigla: .sigla_automobilistica}'
italy_fixed.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `select(.nome == "Abruzzo")` - Selezione della regione Abruzzo

3. `.province[]` - Iterazione su tutte le province della regione
4. `{nome, sigla: .sigla_automobilistica}` - Creazione di oggetti con nome provincia e sigla

1.6 Trovare tutti i comuni con un nome specifico in tutta Italia

```
jq '.regioni[].province[].comuni[] | select(.nome == "San Giovanni") | {comune: .nome, provincia: ..._key, codice: .codice_catastale}' italy_fixed.json
```

Commento passo passo:

1. `.regioni[].province[].comuni[]` - Navigazione a tutti i comuni in tutte le province
2. `select(.nome == "San Giovanni")` - Filtro per nome comune
3. Creazione di oggetto con dettagli del comune trovato

Nota: Per includere anche la provincia/regione di appartenenza serve una query più complessa con path().

1.7 Elencare comuni con coordinate geografiche

```
jq '.regioni[].province[].comuni[] | select(.nome | startswith("Roma")) | {nome, lat: .latitudine, lon: .longitudine}' italy_fixed.json
```

Commento passo passo:

1. `.regioni[].province[].comuni[]` - Accesso a tutti i comuni
2. `select(.nome | startswith("Roma"))` - Filtro per comuni il cui nome inizia con "Roma"
3. `startswith()` - Funzione che verifica se una stringa inizia con il prefisso specificato
4. `{nome, lat: .latitudine, lon: .longitudine}` - Creazione oggetto con coordinate

1.8 Trovare la provincia e la regione di un comune specifico

```
jq '.regioni[] as $regione | $regione.province[] as $provincia | $provincia.comuni[] | select(.nome == "Milano") | {comune: .nome, provincia: $provincia.nome, regione: $regione.nome}' italy_fixed.json
```

Commento passo passo:

1. `.regioni[] as $regione` - Iterazione sulle regioni, salvando ogni regione nella variabile `$regione`
2. `$regione.province[] as $provincia` - Iterazione sulle province, salvando ogni provincia in `$provincia`
3. `$provincia.comuni[]` - Iterazione sui comuni della provincia corrente
4. `select(.nome == "Milano")` - Filtro per il comune cercato

5. `{comune: .nome, provincia: $provincia.nome, regione: $regione.nome}` - Costruzione dell'oggetto risultato usando le variabili salvate
6. L'uso di variabili permette di mantenere i riferimenti ai livelli superiori della gerarchia

---

PARTE 2: Query di Manipolazione e Trasformazione

2.1 Estrazione di un sotto-albero: Singola Regione

Obiettivo: Estrarre tutti i dati di una regione specifica in un nuovo file JSON.

```
jq '{regioni: [.regioni[] | select(.nome == "Lombardia")]}' italy_fixed.json > lombardia.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `select(.nome == "Lombardia")` - Selezione della regione Lombardia
3. `[...]` - Le parentesi quadre creano un array contenente la regione selezionata
4. `{regioni: [...]}` - Creazione di un nuovo oggetto JSON con la stessa struttura dell'originale
5. `> lombardia.json` - Reindirizzamento dell'output in un nuovo file

Risultato: File `lombardia.json` con struttura:

```
{
  "regioni": [
    {
      "nome": "Lombardia",
      "province": [...],
      ...
    }
  ]
}
```

2.2 Estrazione di un sotto-albero: Singola Provincia

Obiettivo: Estrarre tutti i dati di una provincia specifica in un nuovo file JSON.

```
jq '{provincia: (.regioni[].province[] | select(.nome == "Milano"))}' italy_fixed.json >
milano.json
```

Commento passo passo:

1. `.regioni[].province[]` - Navigazione a tutte le province
2. `select(.nome == "Milano")` - Selezione della provincia Milano

3. `{provincia: (...)}` - Creazione di un nuovo oggetto con chiave "provincia"
4. Il risultato è un oggetto JSON con la provincia e tutti i suoi comuni

Risultato: File `milano.json` con struttura:

```
{
  "provincia": {
    "nome": "Milano",
    "sigla_automobilistica": "MI",
    "comuni": [...],
    ...
  }
}
```

Variante alternativa per mantenere anche il contesto regionale:

```
jq '.regioni[] | select(.province[].nome == "Milano") | {regione: .nome, provincia: (.province[] | select(.nome == "Milano"))}' italy_fixed.json > milano_con_regione.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione sulle regioni
2. `select(.province[].nome == "Milano")` - Selezione della regione che contiene la provincia Milano
3. `{regione: .nome, ...}` - Inclusione del nome della regione nel risultato
4. `.province[] | select(.nome == "Milano")` - Estrazione della provincia specifica

2.3 Estrazione di tutte le province senza i comuni

Obiettivo: Creare un file con solo le province, eliminando l'array dei comuni.

```
jq '{regioni: [.regioni[] | {nome, province: [.province[] | del(.comuni)]}]}' italy_fixed.json > province_senza_comuni.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `{nome, province: [...]}` - Creazione di nuovo oggetto regione con solo nome e province
3. `.province[]` - Iterazione su tutte le province della regione
4. `del(.comuni)` - Eliminazione del campo "comuni" da ogni provincia
5. `[...]` - Creazione di array per contenere le province modificate
6. Il risultato esterno è wrappato in `{regioni: [...]}` per mantenere la struttura

Risultato: File con struttura semplificata:

```
{
  "regioni": [
    {
```

```
      "nome": "Abruzzo",
      "province": [
        {
          "nome": "Chieti",
          "sigla_automobilistica": "CH",
          "latitudine": ...,
          "longitudine": ...,
          "id": ...
        }
      ]
    }
  ]
}
```

Variante più completa con selezione campi:

```
jq '{regioni: [.regioni[] | {nome: .nome, id: .id, province: [.province[] | {nome,
sigla_automobilistica, id}]}]}' italy_fixed.json > province_essenziali.json
```

Commento passo passo:

1. Simile alla query precedente ma con controllo esplicito dei campi
2. `{nome, sigla_automobilistica, id}` - Selezione esplicita dei campi da mantenere per ogni provincia
3. Questo approccio è più chiaro e predicibile nei risultati

2.4 Estrazione di un elenco piatto di tutte le province

Obiettivo: Creare un array semplice con tutte le province, perdendo la struttura gerarchica.

```
jq '{province: [.regioni[].province[] | {provincia: .nome, sigla: .sigla_automobilistica, regione:
..._key}]}' italy_fixed.json > elenco_province.json
```

Commento passo passo:

1. `.regioni[].province[]` - Navigazione a tutte le province
2. Creazione di oggetti semplificati per ogni provincia
3. `{province: [...]}` - Wrapping in un oggetto con array "province"

Risultato: Lista piatta di province:

```
{
  "province": [
    {"provincia": "Chieti", "sigla": "CH", "regione": "Abruzzo"},
    {"provincia": "L'Aquila", "sigla": "AQ", "regione": "Abruzzo"},
    ...
  ]
}
```

2.5 Estrazione di statistiche aggregate

Obiettivo: Creare un documento con statistiche per ogni regione.

```
jq '{statistiche: [.regioni[] | {regione: .nome, numero_province: (.province | length),
numero_comuni: ([.province[].comuni[]] | length)}]}' italy_fixed.json > statistiche.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione su tutte le regioni
2. `(.province | length)` - Conteggio del numero di province
3. `[.province[].comuni[]]` - Creazione di un array con tutti i comuni di tutte le province
4. `| length` - Conteggio del numero totale di comuni
5. Creazione di oggetto con statistiche per ogni regione

Risultato:

```
{
  "statistiche": [
    {
      "regione": "Abruzzo",
      "numero_province": 4,
      "numero_comuni": 305
    },
    ...
  ]
}
```

2.6 Trasformazione in formato CSV-like

Obiettivo: Creare un formato tabulare per l'import in altri sistemi.

```
jq -r '.regioni[].province[].comuni[] | [.nome, .codice_catastale, .id, .latitudine, .longitudine]
| @csv' italy_fixed.json > comuni.csv
```

Commento passo passo:

1. `.regioni[].province[].comuni[]` - Navigazione a tutti i comuni
2. `[.nome, ...]` - Creazione di un array con i campi desiderati nell'ordine specificato
3. `@csv` - Formatter che converte l'array in una riga CSV
4. `-r` (flag di jq) - Output in formato raw, senza escape JSON

Con intestazioni:

```
jq -r '["nome","codice_catastale","id","latitudine","longitudine"] | @csv,
(.regioni[].province[].comuni[] | [.nome, .codice_catastale, .id, .latitudine, .longitudine] |
@csv)' italy_fixed.json > comuni.csv
```

Commento passo passo:

1. `["nome",...]` - Primo array con i nomi delle colonne
2. `| @csv` - Conversione intestazione in formato CSV
3. `,` - Operatore che concatena gli output
4. Poi segue la query per i dati effettivi

2.7 Filtro e trasformazione combinati

Obiettivo: Estrarre solo i comuni di province con un certo numero minimo di comuni.

```
jq '{regioni: [.regioni[] | {nome, province: [.province[] | select((.comuni | length) > 100) | {nome, sigla_automobilistica, numero_comuni: (.comuni | length)}]}] | select(.province | length > 0)}' italy_fixed.json > province_grandi.json
```

Commento passo passo:

1. `.regioni[]` - Iterazione sulle regioni
2. `.province[]` - Iterazione sulle province
3. `select((.comuni | length) > 100)` - Filtro per province con più di 100 comuni
4. `{nome, sigla_automobilistica, numero_comuni: ...}` - Creazione oggetto provincia semplificato
5. `select(.province | length > 0)` - Filtro finale per escludere regioni senza province che soddisfano il criterio
6. Questa query combina filtri a più livelli della gerarchia

2.8 Raggruppamento per sigla automobilistica

Obiettivo: Creare un dizionario indicizzato per sigla automobilistica.

```
jq '[.regioni[].province[] | {(.sigla_automobilistica): {nome, regione: ..._key, numero_comuni: (.comuni | length)}}] | add' italy_fixed.json > province_per_sigla.json
```

Commento passo passo:

1. `.regioni[].province[]` - Navigazione a tutte le province
2. `{(.sigla_automobilistica): {...}}` - Creazione di oggetto con chiave dinamica (la sigla)
3. `[...]` - Creazione di array di oggetti
4. `add` - Funzione che unisce tutti gli oggetti dell'array in un unico oggetto
5. Il risultato è un dizionario dove ogni sigla è una chiave

Risultato:

```
{
  "CH": {
    "nome": "Chieti",
    "regione": "Abruzzo",
    "numero_comuni": 104
  },
  "MI": {
    "nome": "Milano",
    "regione": "Lombardia",
    "numero_comuni": 133
  },
  ...
}
```

PARTE 3: Query Avanzate

3.1 Ricerca fuzzy (approssimativa)

```
jq '.regioni[].province[].comuni[] | select(.nome | test(".*San.*"; "i")) | .nome' italy_fixed.json
```

Commento passo passo:

1. `.regioni[].province[].comuni[]` - Navigazione a tutti i comuni
2. `test(".*San.*"; "i")` - Ricerca con regex, case-insensitive
3. `"i"` - Flag per ignorare maiuscole/minuscole
4. `.*San.*` - Pattern che cerca "San" in qualsiasi posizione

3.2 Join di informazioni da livelli diversi

```
jq '[.regioni[] as $r | $r.province[] as $p | $p.comuni[] | {comune: .nome, provincia: $p.nome,
regione: $r.nome, sigla: $p.sigla_automobilistica}] | sort_by(.comune)' italy_fixed.json >
comuni_completo.json
```

Commento passo passo:

1. `.regioni[] as $r` - Salvataggio regione corrente in variabile
2. `$r.province[] as $p` - Salvataggio provincia corrente in variabile
3. `$p.comuni[]` - Iterazione sui comuni
4. Creazione di oggetto che combina informazioni da tutti i livelli
5. `sort_by(.comune)` - Ordinamento alfabetico per nome comune
6. `[...]` - Wrapper in array per l'output finale

3.3 Generazione di un indice gerarchico

```
jq '{indice: {regioni: [.regioni[] | {nome, numero_province: (.province | length), province:
[.province[] | {nome, sigla_automobilistica, numero_comuni: (.comuni | length)}]}]}}'
italy_fixed.json > indice.json
```

Commento passo passo:

1. Creazione di una struttura ad albero semplificata
2. Ogni livello contiene informazioni aggregate
3. Utile per navigazione e overview dei dati

Note Tecniche e Best Practices

Uso delle variabili

Le variabili in jq (sintassi `as $nome`) sono fondamentali per:

- Mantenere riferimenti a livelli superiori della gerarchia
- Evitare ricalcoli di espressioni complesse
- Rendere le query più leggibili

Operatori utili

- `select()` - Filtraggio condizionale
- `map()` - Trasformazione di array
- `length` - Conteggio elementi
- `add` - Somma o merge di oggetti
- `sort_by()` - Ordinamento
- `group_by()` - Raggruppamento
- `del()` - Eliminazione di campi

Performance

Per file JSON molto grandi:

- Usare `--stream` per processing incrementale
- Limitare la profondità di iterazione quando possibile

- Evitare ricerche esaustive se non necessarie

Output formatting

- `-r` per output raw (senza quote)
- `-c` per output compatto (una riga)
- `-M` per output monocromatico
- Default: output pretty-printed

Esempi di Uso Combinato con Shell

Contare province per regione

```
jq -r '.regioni[] | "\(.nome): \(.province | length) province"' italy_fixed.json
```

Pipeline complessa

```
jq '.regioni[].province[].comuni[] | select(.latitudine > 45) | .nome' italy_fixed.json | sort |
uniq | wc -l
```

Questo conta i comuni unici sopra il 45° parallelo (nord Italia).

Esportazione multipla

```
for regione in $(jq -r '.regioni[].nome' italy_fixed.json); do
  jq --arg reg "$regione" '{regioni: [.regioni[] | select(.nome == $reg)]}' italy_fixed.json >
"output/${regione}.json"
done
```

Questo crea un file separato per ogni regione