Il Formato JSON

Struttura, sintassi, esempi ed utilizzi

Francesco Gobbi

23 febbraio 2026

Cos'è JSON

Definizione

JSON (**JavaScript Object Notation**) è un formato standard aperto per:

- memorizzare informazioni in modo strutturato
- scambiare dati tra applicazioni (web e non)

- È semplice da scrivere e da analizzare
- È indipendente dalla piattaforma e dal linguaggio
- È ampiamente supportato nelle applicazioni web moderne

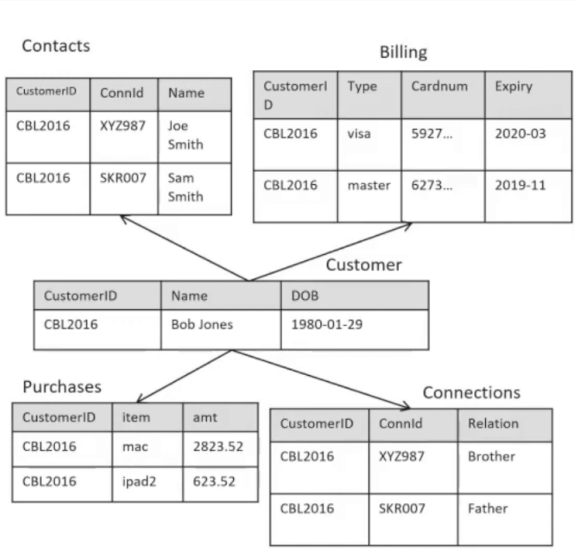
Perché JSON è così diffuso

- È compatto rispetto a XML (meno struttura ripetuta)
- Si adatta bene allo scambio dati (risposte di servizi web / API)
- Qualsiasi linguaggio che gestisce stringhe può analizzarlo

Database e JSON

- DB relazionali (es. **MySQL**, **PostgreSQL**) supportano campi JSON
- DB NoSQL (es. **MongoDB**) usano documenti in stile JSON per i record

JSON nei Database Relazionali



DocumentKey: **CBL2016**

```
{
  "Name" : "Bob Jones",
  "DOB" : "1980-01-29",
  "Billing" : [
    {
      "type"   : "visa",
      "cardnum" : "5927-2842-2847-3909",
      "expiry" : "2020-03"
    },
    {
      "type"   : "master",
      "cardnum" : "6273-2842-2847-3909",
      "expiry" : "2019-11"
    }
  ],
  "Connections" : [
    {
      "CustId"  : "XYZ987",
      "Relation"    : "Brother"
    },
    {
      "CustId"  : "PQR823",
      "Relation"    : "Father"
    }
  ],
  "Purchases" : [
    { "id":12, item: "mac",  "amt": 2823.52 },
    { "id":19, item: "ipad2", "amt": 623.52 }
  ]
}
```

Differenze tra XML e JSON

XML

- Molto flessibile: tag personalizzabili
- Verboso: tag aperti/chiusi
- Utile in contesti complessi e documentali

JSON

- Più compatto e leggibile
- Struttura immediata (oggetti/array)
- Ideale per scambio dati in applicazioni web

Idea chiave

JSON tende a essere più semplice e immediato da leggere; XML più descrittivo ma spesso più complesso.

Differenze tra XML e JSON

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endereco>
    <cep>31270901</cep>
    <city>Belo Horizonte</city>
    <neighborhood>Pampulha</neighborhood>
    <service>correios</service>
    <state>MG</state>
    <street>Av. Presidente Antônio Carlos, 6627</street>
</endereco>
```

vs.

JSON

```json
{
    "endereco": {
        "cep": "31270901",
        "city": "Belo Horizonte",
        "neighborhood": "Pampulha",
        "service": "correios",
        "state": "MG",
        "street": "Av. Presidente Antônio Carlos, 6627"
    }
}
```

Formato di JSON

JSON è costituito da **due sole strutture**:

- **insieme di coppie** (nome, valore) = oggetti
- **lista ordinata** di valori = array

Esempio (oggetto)

```
{
  "nome": "Andrea",
  "eta": 18
}
```

Oggetti letterali JSON

Un **oggetto** è un insieme di coppie `chiave: valore`:

- racchiuso tra { e }
- coppie separate da virgole
- le chiavi sono stringhe

Sintassi

```
{
  "proprieta1": "valore",
  "proprieta2": "valore"
}
```

Nota importante

JSON ammette solo valori: stringhe, numeri, booleani, array, oggetti, `null`. Non contiene funzioni.

Tipi di dato supportati

JSON supporta i seguenti tipi:

- **Number** (interi e decimali)
- **String**
- **Boolean** (`true`/`false`)
- **Array**
- **Object**
- **null**

Nota sui numeri

In JSON non sono previsti formati come ottale o esadecimale, e non esistono `NaN` o `Infinity`.

Esempi rapidi di tipi

String e Number

```
{
  "nome": "Andrea",
  "eta": 18
}
```

Boolean

```
{
  "promosso": true
}
```

Array

```
{
  "voti": [6, 7, 8]
}
```

null

```
{
  "valore": null
}
```

Array e Object

Array: lista ordinata di valori

```
{
  "books": [
    {"linguaggio": "Java",   "edizione": "seconda"},
    {"linguaggio": "Python", "edizione": "prima"}
  ]
}
```

Object: insieme di coppie chiave-valore

```
{ "id": "01669",
  "linguaggio": "JAVA",
  "prezzo": 15.50 }
```

Whitespace (spazi)

Gli spazi (whitespace) possono essere inseriti per migliorare la leggibilità:

- non cambiano il significato del JSON
- rendono il documento più facile da leggere

Creare oggetti JSON in JavaScript

In JavaScript possiamo creare oggetti (in stile JSON) in tre modi:

1. Creare un oggetto vuoto
2. Creare un oggetto con il costruttore `Object()`
3. Creare un oggetto assegnando valori direttamente

Esempi

```
var oggetto1 = {};
var oggetto2 = new Object();
var oggetto3 = { "titolo":"TPSIT vol.3", "prezzo":23.50 };
```

Scenario didattico con l'uso di JSON

Obiettivo

Una pagina web mostra le **proiezioni disponibili** (film + data/ora + posti liberi).

- I dati sono in un **database MySQL**
- Un servizio (API) interroga MySQL e restituisce **JSON**
- La pagina web usa JavaScript (**fetch**) per richiedere i dati e visualizzarli

Visione d'insieme: tutti i passaggi

1. **DB MySQL**: tabelle e dati
2. **Query SQL**: seleziono ciò che serve (JOIN + aggregazioni)
3. **API (server)**: espone un endpoint `/api/proiezioni`
4. **Risposta JSON**: formato standard di scambio dati
5. **Front-end JS**: `fetch()` chiama l'endpoint
6. **Rendering HTML**: mostro i risultati nella pagina

Idea chiave

Il browser **non parla direttamente con MySQL**: serve un **server/API** in mezzo.

Passaggio 1 — Struttura del database (MySQL)

Tabelle minime

- `film`: informazioni del film
- `proiezioni`: data/ora, sala, capienza
- `prenotazioni`: quante persone hanno prenotato

DDL (creazione tabelle) — pulito e commentato

```
-- Tabella film: un record per ogni film
CREATE TABLE film (
  id INT AUTO_INCREMENT PRIMARY KEY,
  titolo VARCHAR(100) NOT NULL,
  durata_min INT NOT NULL
);
```

Passaggio 1 — Struttura del database (MySQL)

DDL (creazione tabelle) — pulito e commentato

```
-- Tabella proiezioni: ogni proiezione riferisce un film
CREATE TABLE proiezioni (
  id INT AUTO_INCREMENT PRIMARY KEY,
  film_id INT NOT NULL,
  data_ora DATETIME NOT NULL,
  sala VARCHAR(20) NOT NULL,
  capienza INT NOT NULL,
  FOREIGN KEY (film_id) REFERENCES film(id)
);
```

Passaggio 1 — Struttura del database (MySQL)

DDL (creazione tabelle) — pulito e commentato

```
-- Tabella prenotazioni: 1 record = 1 prenotazione (n posti)
CREATE TABLE prenotazioni (
  id INT AUTO_INCREMENT PRIMARY KEY,
  proiezione_id INT NOT NULL,
  posti INT NOT NULL,
  FOREIGN KEY (proiezione_id) REFERENCES proiezioni(id)
);
```

Passaggio 2 — Query SQL: cosa serve al sito?

Richiesta della pagina

Per ogni proiezione vogliamo visualizzare:

- **titolo** del film
- **data/ora** della proiezione
- **sala**
- **posti prenotati** (totale)

N.B.

Ci basta una query con **JOIN** (per il titolo) e **SUM** (per sommare i posti).

Passaggio 2 — Query MySQL

Query (JOIN + SUM)

```
-- N.B. LEFT JOIN + COALESCE per gestire il caso "zero prenotazioni"
SELECT
  p.id        AS proiezione_id,
  f.titolo    AS titolo,
  p.data_ora  AS data_ora,
  p.sala      AS sala,
  COALESCE(SUM(pr.posti), 0) AS prenotati
FROM proiezioni p
JOIN film f ON f.id = p.film_id
LEFT JOIN prenotazioni pr ON pr.proiezione_id = p.id
GROUP BY p.id, f.titolo, p.data_ora, p.sala
ORDER BY p.data_ora;
```

Passaggio 3 — API: chi parla con MySQL

Ruolo dell'API

- riceve una richiesta HTTP dal browser
- esegue la query in MySQL sul DataBase
- restituisce una risposta in JSON

Nota

Possiamo implementare l'API con PHP, Node.js, Python,... Con vari linguaggi quindi.

Passaggio 3 — Esempio API (PHP) che restituisce JSON

api/proiezioni.php

```
<?php
// 1) Dichiarazione per poter rispondere in JSON
header('Content-Type: application/json; charset=utf-8');

// 2) Connessione al DB (in produzione usare variabili d'ambiente)
$pdo = new PDO(
  "mysql:host=localhost;dbname=cinema;charset=utf8",
  "user",
  "password",
  [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
);
```

Passaggio 3 — Esempio API (PHP) che restituisce JSON

```
// 3) Creo la query: JOIN + SUM
$sql = "SELECT
          p.id AS proiezione_id,
          f.titolo AS titolo,
          p.data_ora AS data_ora,
          p.sala AS sala,
          COALESCE(SUM(pr.posti), 0) AS prenotati
        FROM proiezioni p
        JOIN film f ON f.id = p.film_id
        LEFT JOIN prenotazioni pr ON pr.proiezione_id = p.id
        GROUP BY p.id, f.titolo, p.data_ora, p.sala
        ORDER BY p.data_ora";
```

Passaggio 3 — Esempio API (PHP) che restituisce JSON

```
// 4) Conversione in JSON e successivo invio al client
echo json_encode($rows, JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT);
?>
```

Passaggio 4 — Il JSON che arriva al browser

Risposta tipica

Un array di oggetti: ogni oggetto rappresenta una proiezione.

Esempio JSON (un solo oggetto)

```
[
  {
    "proiezione_id": 12,
    "titolo": "Interstellar",
    "data_ora": "2026-03-10 20:30:00",
    "sala": "A",
    "prenotati": 45
  }
]
```

Passaggio 5 — La pagina web chiede i dati (fetch)

Cosa fa il front-end

- invia una richiesta HTTP all'endpoint `/api/proiezioni.php`
- riceve JSON
- lo trasforma in oggetti JS e li usa per costruire HTML

Passaggio 5 — La pagina web chiede i dati (fetch)

JavaScript

```
// Carico le proiezioni dall'API
async function caricaProiezioni() {
  // 1) Request
  const response = await fetch("/api/proiezioni.php");

  // 2) Parsing JSON (stringa -> array di oggetti)
  const proiezioni = await response.json();

  // 3) Visualizzo nella pagina
  renderProiezioni(proiezioni);
}
caricaProiezioni();
```

Passaggio 6 — Visualizzare i dati nella pagina

```
<div id="listaProiezioni"></div>
<script>
function renderProiezioni(lista) {
  const box = document.getElementById("listaProiezioni");
  box.innerHTML = ""; // box pulito, posso quindi aggiungere
  for (const p of lista) {
    const div = document.createElement("div");
    div.innerHTML = `
      <h3>${p.titolo}</h3>
      <p>Quando: ${p.data_ora} | Sala: ${p.sala}</p>
      <p>Posti prenotati: <b>${p.prenotati}</b></p>
      <hr>`;
    box.appendChild(div); } }
</script>
```

Riassumendo: cosa devono portarsi a casa

Il flusso completo dei dati

1. **Database (MySQL)** I dati sono salvati in tabelle strutturate (film, proiezioni, prenotazioni).
2. **Query SQL** La query seleziona e combina i dati necessari (JOIN + SUM).
3. **API (server)** Il server interroga il database e prepara la risposta.
4. **JSON** I dati vengono convertiti in formato JSON per essere inviati al browser.
5. **JavaScript (fetch)** Il browser richiede i dati e li riceve come oggetti JavaScript.
6. **HTML (visualizzazione)** I dati vengono mostrati nella pagina web.

Idea fondamentale

Il browser **non accede mai direttamente al database**: c'è sempre un **server** che fa da intermediario.