📦 Marrel S.r.l. - JSON to SQL Import Script

Script Python per convertire i dati JSON del gestionale in statement SQL per l'import su Supabase.

🎯 Caratteristiche

- ✅ Conversione automatica da **camelCase** (JSON) a **snake_case** (SQL)
- ✅ Gestione corretta delle **foreign keys** (ordine di inserimento)
- ✅ Conversione **array PostgreSQL** per campi multipli
- ✅ Escape automatico delle virgolette singole
- ✅ Gestione valori NULL
- ✅ Transazione BEGIN/COMMIT per atomicità
- ✅ Gestione campi obbligatori con valori di default
- ✅ Supporto per strutture nested (rateizzi + rate)
- ✅ Conversione presenze da dizionario a tabella

📋 Requisiti

- Python 3.6+
- File JSON esportato dal gestionale

🚀 Utilizzo

Sintassi Base

```bash
python json_to_sql.py <file_json> [file_output.sql]
```

Esempi

```bash
# Output automatico (crea gestionale-edile-2025-11-14_import.sql)
python json_to_sql.py gestionale-edile-2025-11-14.json

# Output personalizzato
python json_to_sql.py gestionale-edile-2025-11-14.json marrel_import.sql
```

📊 Tabelle Supportate

Lo script converte le seguenti tabelle con le mappature corrette:

JSON Key	Tabella SQL	Note
lavoratori	lavoratori	Dipendenti dell'azienda
fornitori	fornitori	Fornitori con default per ragione_sociale
subappaltatori	subappaltatori	Subappaltatori
cantieri	cantieri	Cantieri e commesse
automezzi	veicoli	Parco mezzi
unilav	unilav	Comunicazioni obbligatorie
corsiVisite	certificazioni	Corsi e certificazioni
presenze	presenze	Convertite da dict a records
notePresenze	note_presenze	Note sulle presenze
sal	sal	Stati di avanzamento lavori
fattureEmesse	fatture_emesse	Fatture clienti
acconti	acconti	Acconti pagati/ricevuti
rateizzi	rateizzi + rate	Rateizzazioni (nested)
movimentiContabili	movimenti_contabili	Movimenti di cassa
storicoPaghe	storico_paghe	Storico paghe
documenti	documenti	Documenti aziendali

📝 Mappature Colonne Principali

Lavoratori

codiceFiscale → codice_fiscale
ibanPrimario → iban_primario
ibanSecondario → iban_secondario

dataNascita → data_nascita
luogoNascita → luogo_nascita

Cantieri

clienteId → cliente_id
clienteNome → cliente_nome
cassaEdile → casse_edile
dataInizio → data_inizio
dataFinePrevista → data_fine_prevista
importoContratto → importo_contratto
importoLavori → importo_lavori
codiceCommessa → codice_commessa
rupNome → rup_nome
cseNome → cse_nome
dlNome → dl_nome
dataComunicazioneDNLT → data_comunicazione_dnlt
scadenzaDNLT → scadenza_dnlt

Fornitori

ragioneSociale → ragione_sociale
piva → partita_iva
codiceFiscale → codice_fiscale

UniLav

tipoUnilav → tipo_unilav
lavoratoreId → lavoratore_id
cantiereId → cantiere_id
dataComunicazione → data_comunicazione
dataInizio → data_inizio

dataFine → data_fine
tipoContratto → tipo_contratto
oreSettimanali → ore_settimanali

Presenze (Speciale)

Le presenze sono convertite da formato dizionario:

```json
{
  "1759331131292-2025-07-31": {
    "tipo": "lavoro",
    "ore": "8",
    "cantiere": "1759333459303"
  }
}
```

A formato tabella:

```sql
INSERT INTO presenze (id, lavoratore_id, data, tipo, ore, cantiere_id)
VALUES ('1759331131292-2025-07-31', '1759331131292', '2025-07-31', 'lavoro', '8', '1759333459303');
```

🔧 Gestione Campi Obbligatori

Lo script gestisce automaticamente i campi obbligatori mancanti:

- **Fornitori senza ragione_sociale**: Usa `nome` o genera `"Fornitore {id}"`
- **Cantieri senza nome**: Genera `"Cantiere {id}"`
- **Lavoratori senza nome/cognome**: Usa valori di default

📤 Import su Supabase

Passo 1: Generare il file SQL

```bash
python json_to_sql.py gestionale-edile-2025-11-14.json
```

Passo 2: Verificare il file

Apri `gestionale-edile-2025-11-14_import.sql` e verifica che contenga:

- BEGIN all'inizio
- COMMIT alla fine
- INSERT statements corretti

Passo 3: Eseguire su Supabase

1. Vai su **Supabase Dashboard**
2. Apri **SQL Editor**
3. Copia il contenuto del file SQL
4. Clicca **Run** per eseguire
5. Verifica che tutti gli INSERT siano andati a buon fine

Passo 4: Verificare i dati

```sql
```

```sql
-- Conta i record importati
SELECT 'lavoratori' as tabella, COUNT(*) as records FROM lavoratori
UNION ALL
SELECT 'fornitori', COUNT(*) FROM fornitori
UNION ALL
SELECT 'cantieri', COUNT(*) FROM cantieri
UNION ALL
SELECT 'presenze', COUNT(*) FROM presenze
UNION ALL
SELECT 'unilav', COUNT(*) FROM unilav;
```

⚠️ Note Importanti

1. **Backup**: Fai sempre un backup del database prima dell'import
2. **Transazione**: Lo script usa BEGIN/COMMIT, quindi o tutto va a buon fine o nulla viene importato
3. **Foreign Keys**: L'ordine di inserimento rispetta le dipendenze tra tabelle
4. **IDs**: Gli ID vengono mantenuti dal JSON originale
5. **Duplicati**: Se esegui lo script due volte, otterrai errori di chiave duplicata

🐛 Troubleshooting

Errore: "null value in column X violates not-null constraint"
Lo script ora gestisce automaticamente i campi obbligatori mancanti. Se vedi questo errore, contatta lo sviluppatore.

Errore: "duplicate key value violates unique constraint"
Significa che i dati sono già stati importati. Per reimportare:

```sql
```

```
-- ATTENZIONE: Questo cancella TUTTI i dati!
BEGIN;
TRUNCATE TABLE lavoratori, fornitori, cantieri, presenze, unilav, certificazioni, sal, fatture_emesse, acconti, rateizz
COMMIT;
```

Errore: "foreign key constraint fails"

L'ordine di inserimento è corretto nello script. Verifica che lo schema SQL sia stato creato correttamente.

📈 Statistiche Import

Dalle statistiche dell'ultimo import:

- **909 INSERT statements** totali
- **30 lavoratori**
- **226 fornitori**
- **4 cantieri**
- **Migliaia di presenze**
- **SAL, fatture, acconti, rateizzi, ecc.**

🔄 Aggiornamenti Futuri

Per aggiornare le mappature delle colonne, modifica il dizionario `COLUMN_MAPPINGS` nello script:

```
python
```

```
COLUMN_MAPPINGS = {
    'tabella': {
        'campoJSON': 'campo_sql',
        # ...
    }
}
```

📞 Supporto

Per problemi o domande, contatta lo sviluppatore del gestionale.

Versione Script: 2.0
Data: Novembre 2025
Compatibile con: Supabase PostgreSQL