

GomorraSQL

Neapolitan Dialect Compiler

Quando 'o SQL parla napoletano



Angelo Alberico
Matr. NF22500104

Implementation Report
11 Gennaio 2026

Introduzione: Progetto Originale

GomorraSQL: DSL dichiarativo ispirato a SQL con dialetto napoletano.

Caratteristiche principali:

- ▶ **Traduttore runtime:** GomorraSQL → SQL standard
- ▶ Esecuzione su database esistenti (MySQL, PostgreSQL, SQLite)
- ▶ Implementato in Java con JDBC
- ▶ Nessuna analisi semantica o ottimizzazione

"Chest' è 'o progetto nuost'."



Ispirazione:

 github.com/auraspHERE/gomorra-sql

Introduzione: Il nostro progetto

Architettura:

- ▶ Parsing e validazione semantica di query SELECT
- ▶ Esecuzione diretta su **file CSV**, non database
- ▶ Generazione codice LLVM IR

"Chest' è 'o progetto nuost'"

Esempio:

```
ripigliammo * mmiez 'a "guaglioni.csv"
```



```
SELECT * FROM guaglioni
```



Generazione LLVM IR

Strategia di Compilazione:

La generazione di codice LLVM IR si concentra **esclusivamente** sulla clausola WHERE.

“Facimm ‘o core d’ ‘a query.”

Cosa viene compilato:

- ▶ Condizioni logiche (e, o)
- ▶ Operatori di confronto (>, <, =, <>)
- ▶ Controlli NULL (è nisciun, nun è nisciun)

Resto della query: Parsing, validazione semantica, caricamento CSV in Python.



Strumenti e Tecnologie

{ }

Parser

Lark (Python)
Grammatica EBNF
Lexer + Parser

>>

Code Generator

llvmlite
LLVM IR
Ottimizzazione

Py

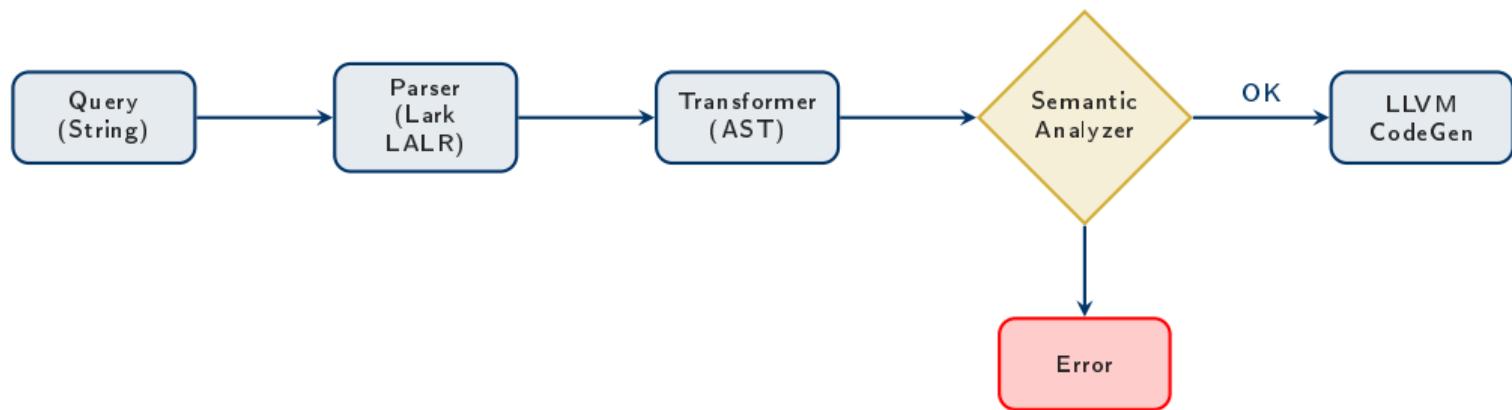
Linguaggio

Python 3.11
Implementazione
Testing (pytest)

Pipeline di Compilazione

Fasi di Compilazione: dalla query in formato testuale fino alla generazione di codice LLVM IR.

La pipeline prevede una fase di **analisi semantica** che valida la correttezza della query prima di procedere alla generazione del codice.



Analisi Lessicale

Token Recognition: Keywords napoletane → Token

| | |
|---------------|------------------|
| ripigliammo | → SELECT_KW |
| mmiez 'a | → FROM_KW |
| arò | → WHERE_KW |
| e / o | → AND_KW / OR_KW |
| è nisciun | → IS_KW NULL_KW |
| >=, <=, <>, = | → COMP_OP |

“O Lexer nun scherza.”



Analisi Sintattica: Grammatica EBNF

Parser LALR con Lark

```
select_stmt: SELECT_KW projection  
           from_clause [where_clause]
```

```
projection: ALL_COLS | column_list
```

```
from_clause: FROM_KW table_ref join_clause*
```

```
where_clause: WHERE_KW condition
```

```
condition: logic_term (OR_KW logic_term)*
```

```
logic_term: logic_factor (AND_KW logic_factor)*
```

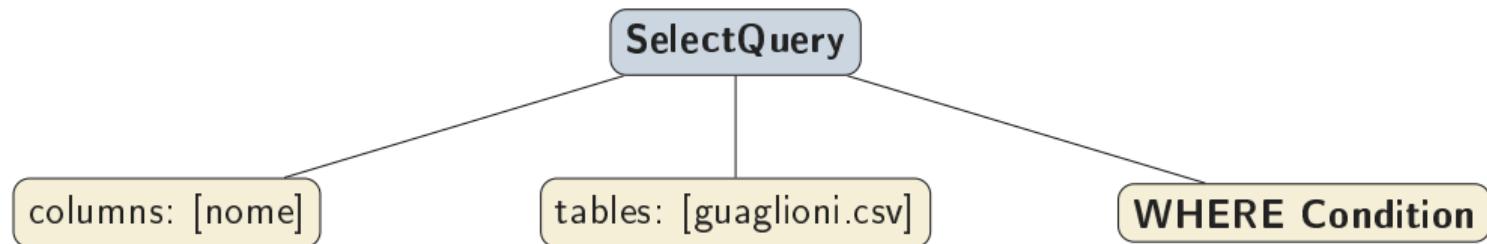
```
logic_factor: comparison | null_check  
           | "(" condition ")"
```

'A grammatica è 'a legge.



Visualizzazione AST (1/2): Struttura SelectQuery

Query: RIPIGLIAMMO nome MMIEZ 'A "guaglioni.csv" arò (eta > 18 e zona = "Scampia") o nome = "Ciro"

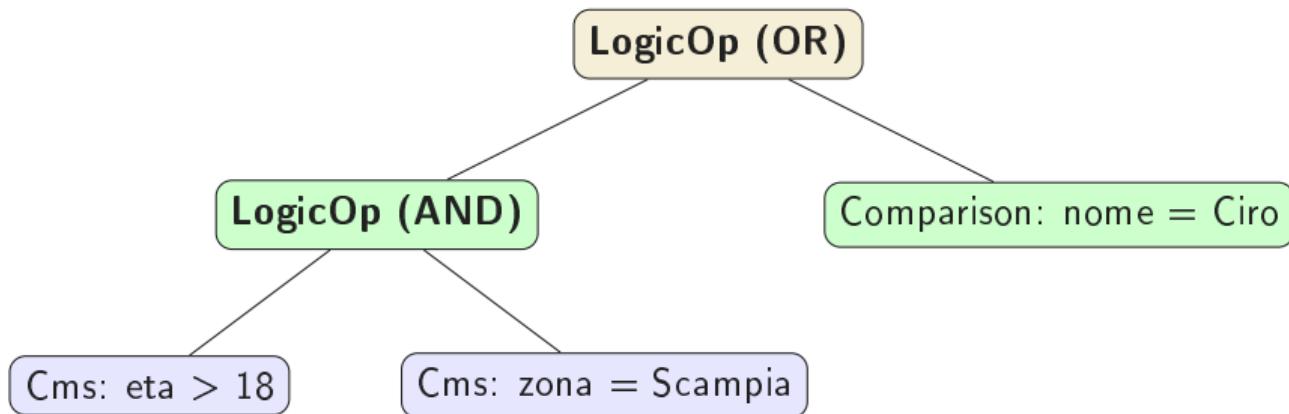


Componenti principali:

- ▶ **Projection:** Lista di colonne da selezionare
- ▶ **From:** Tabelle sorgente (CSV files)
- ▶ **Where:** Albero condizioni logiche

Visualizzazione AST (2/2): Albero Condizioni WHERE

Condizione: $(\text{eta} > 18 \text{ e } \text{zona} = \text{"Scampia"}) \text{ o } \text{nome} = \text{"Ciro"}$



Struttura gerarchica:

- ▶ **OR esterno:** Valutazione short-circuit
- ▶ **AND interno:** Entrambe condizioni devono essere vere
- ▶ **Confronti:** Foglie dell'albero

Analisi Semantica: Symbol Table

Validazione Colonne tramite CSV Header

Step 1: Carica schema tabelle

guaglioni.csv header → {nome, zona, eta}

'E colonne se devono truvà.

Step 2: Verifica esistenza colonne

- ▶ SELECT: nome ∈ {nome, zona, eta} ✓
- ▶ WHERE: eta ∈ {nome, zona, eta} ✓
- ▶ WHERE: zona ∈ {nome, zona, eta} ✓



Errore Semantico: RIPIGLIAMMO cognome MMIEZ 'A

"guaglioni.csv" ... → SemanticError: Colonna 'cognome' non
trovata.

Generazione LLVM: Visitor Pattern

Pattern Utilizzato: Visitor Pattern per attraversare l'AST

Implementazione:

- ▶ LLVMCodeGenerator estende ASTVisitor
- ▶ Ogni nodo AST ha un metodo `visit_*` dedicato
- ▶ Il metodo `visit()` fa dispatch automatico

*“O Visitor cammina
pe’ ll’AST.”*

Metodi Visitor:

- ▶ `visit_comparison()` → Genera codice per confronti
- ▶ `visit_logic_op()` → Genera AND/OR
- ▶ `visit_null_check()` → Genera controlli NULL

Vantaggi: Separazione tra struttura AST e generazione co



Generazione LLVM: Struttura IR

Struttura del Codice LLVM IR:

Il compilatore genera una **singola funzione LLVM** per valutare la clausola WHERE.

“A funzione è semplice.”

Architettura:

- ▶ **Un solo blocco "entry":** Tutta la logica WHERE in un blocco
- ▶ **IRBuilder:** Costruisce istruzioni sequenzialmente
- ▶ **Nessun branching:** Le condizioni AND/OR usano operazioni bit-a-bit
- ▶ Termina con `ret i1` (ritorna true/false)



Type Inference: Regole di Inferenza

Analisi Automatica CSV → Tipo LLVM:

Il compilatore analizza i valori del CSV per determinare il tipo di ogni colonna.

| Condizione | Tipo Inferito | Esempio |
|---------------------------|---------------|---------|
| Valore vuoto | NULL | "" |
| Contiene . e convertibile | float | "19.99" |
| Convertibile a intero | int | "35" |
| Altrimenti | str | "Ciro" |

"E tipi se scoprono da soli."

Strategia:

- ▶ Campiona prime 100 righe del CSV
- ▶ float prevale su int se presente
- ▶ Tipo predominante determina il tipo della colonna



Esempio LLVM IR: Operatori Logici AND/OR

Query: arò (eta > 18 e zona = "Scampia") o nome = "Ciro"

LLVM IR Generato:

```
define i1 @evaluate_row(i32 %.1") {entry:  
    ; Primo confronto: eta > 18  
    ; icmp_signed('>') → icmp sgt (signed greater than)  
    %.3" = icmp sgt i32 35, 18  
  
    ; AND logico con secondo confronto (zona = "Scampia")  
    ; builder.and_() → and i1 (bitwise AND su booleani)  
    %.4" = and i1 %.3", 1  
  
    ; OR logico con terzo confronto (nome = "Ciro")  
    ; builder.or_() → or i1 (bitwise OR su booleani)  
    %.5" = or i1 %.4", 1  
ret i1 %.5"}
```

Testing e Code Coverage

19 test che coprono tutti i casi d'uso principali:

- ▶ SELECT semplice e complesso
- ▶ WHERE con operatori logici (AND/OR)
- ▶ JOIN con disambiguazione
- ▶ Controlli NULL
- ▶ Errori semantici e sintattici
- ▶ Type inference e generators

"E test nun mentono."



Risultato: Tutti i 19 test passati ✓

Grazie per l'attenzione

Angelo Alberico

Matr. NF22500104



github.com/Nakura125



GomorraSQL - Quando 'o SQL parla napoletano