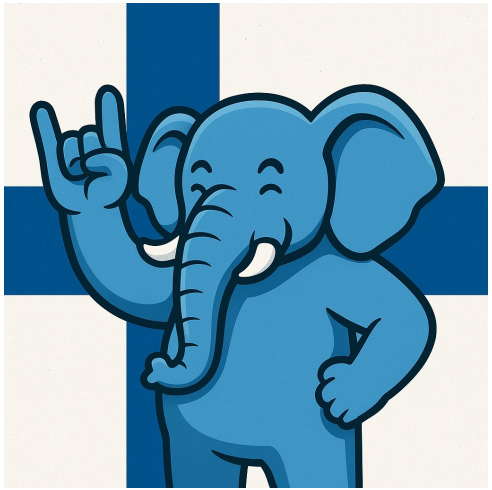
SQL compiler (sqlc): code generation matters



Nikolay Kuznetsov

Helsinki 4.11.2025

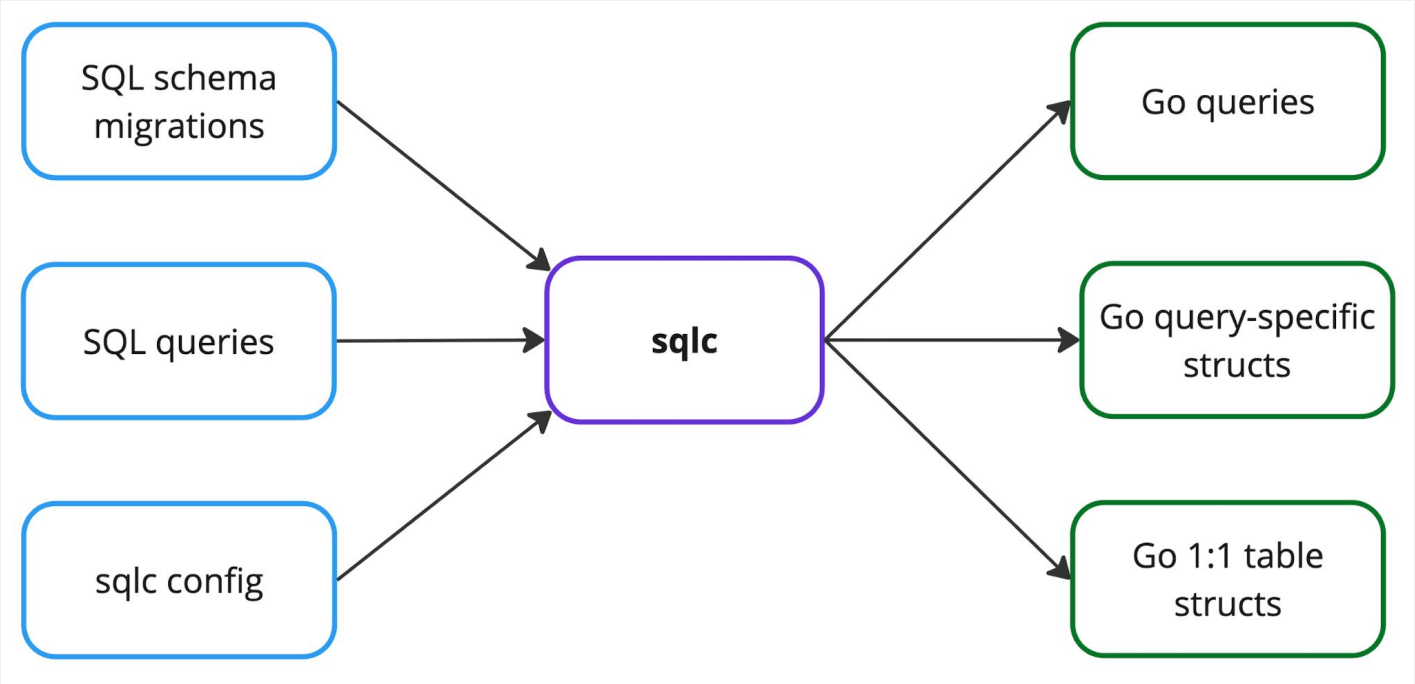
About me

- Senior Software Engineer @Zalando Helsinki 🇫🇮
- Author of ***pgx-outbox*** and ***sqlc++*** projects
- Speaker ’25: ContainerDays Hamburg🇩🇪, GoLab Florence🇮🇹
- C → Java → Kotlin → **Go**

SQL compiler (sqlc)

- SQL-first code generator
  - Go, Kotlin, Python, TypeScript
  - PostgreSQL, MySQL, SQLite
- Written in Go, 16K⭐ on GitHub
- Compile-time query safety, less boilerplate

sqlc in a nutshell



Schema migration: cart items

```
CREATE TABLE IF NOT EXISTS cart_items (

    owner_id        VARCHAR(255)                              NOT NULL,
    product_id      UUID                                      NOT NULL,
    price_amount    DECIMAL                                   NOT NULL,
    price_currency  VARCHAR(3)                                NOT NULL,

    created_at      TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,

    PRIMARY KEY (owner_id, product_id)
);
```

SQL queries + sqlc annotations

```sql
-- name: GetCart :many
SELECT product_id, price_amount, price_currency, created_at
FROM cart_items
WHERE owner_id = $1;

-- name: DeleteItem :execrows
DELETE FROM cart_items WHERE owner_id = $1 AND product_id = $2;
```

Supported annotations: exec, execresults, execrows, many, one, *etc*

Generated code

```
const DeleteItem = `-- name: DeleteItem :execrows
DELETE FROM cart_items WHERE owner_id = $1 AND product_id = $2`

type DeleteItemParams struct {
  OwnerID   string
  ProductID uuid.UUID
}

func (q *Queries) DeleteItem(ctx context.Context, arg DeleteItemParams) (int64, error) {
  result, err := q.db.Exec(ctx, DeleteItem, arg.OwnerID, arg.ProductID)
  if err != nil {
    return 0, err
  }
  return result.RowsAffected(), nil
}
```

sqlc config

```yaml
version: "2"
sql:
  - engine: "postgresql"
    schema: "internal/migrations"
    queries: "internal/db/queries"
    gen:
      go:
        package: "db"
        out: "internal/db"
        sql_package: "pgx/v5"
        emit_exported_queries: true
        overrides:
          - db_type: "uuid"
            go_type:
              import: "github.com/google/uuid"
              type: "UUID"
```

sqlc under the hood

- Uses PostgreSQL parser to convert migrations and queries into *Abstract Syntax Tree* (AST)
  - *pg_query_go* / *go-pgquery* (with *CGO* and without)

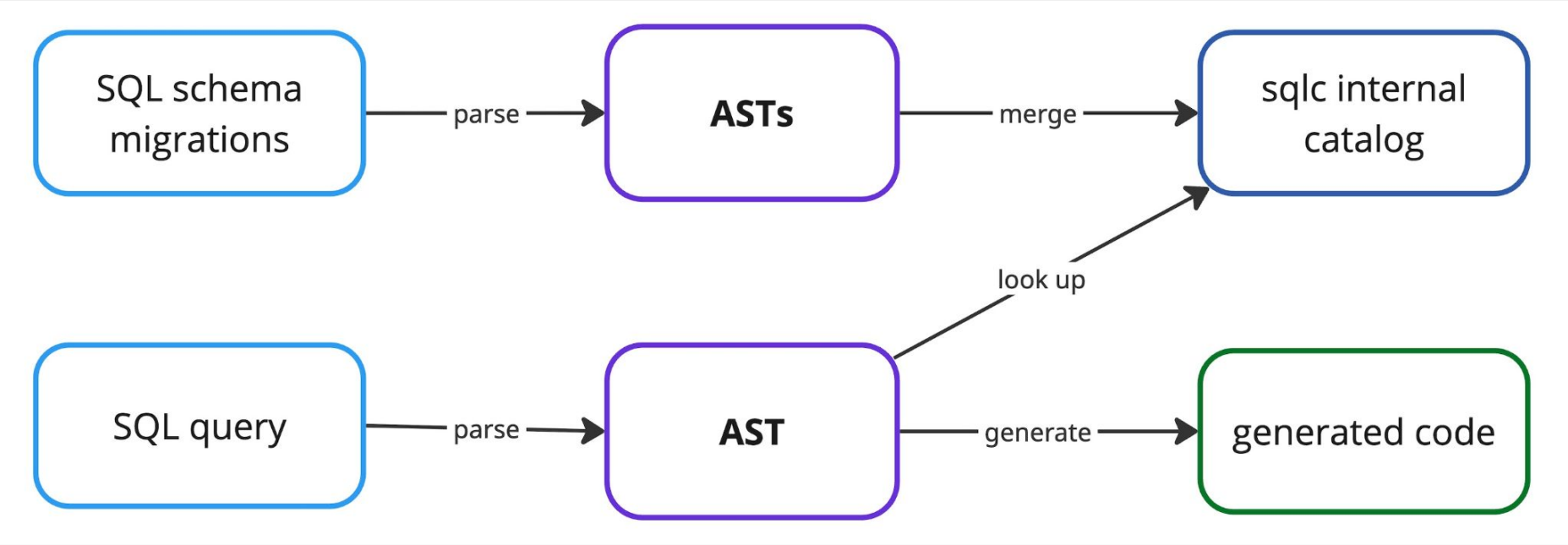
- Migration ASTs → *sqlc* **internal catalog**
  - info about tables, columns, types, etc
  - used to infer types, validate queries

Simplified AST

```
-- name: DeleteItem :execrows
DELETE FROM cart_items WHERE owner_id = $1 AND product_id = $2;
```

```
DeleteStmt
  ├ Relation: cart_items
  └ WhereClause (BoolExpr: AND)
      ├ left: ColumnRef { Name: "owner_id" } = ParamRef{1}
      └ right: ColumnRef { Name: "product_id" } = ParamRef{2}
```

Leveraging ASTs



sqlc challenges

- Dynamic conditional queries
- SQL-first meaning confusion
- Mapping between generated structs and domain models
- Some advanced SQL features, etc

Dynamic conditional queries

```sql
SELECT *
FROM cart_items
WHERE
    (
        -- condition is true / ignored if product_ids is NULL
        (@product_ids::UUID[] IS NULL OR product_id = ANY(@product_ids))

        -- sqlc.narg() is a helper function to hint nullable params
        AND (sqlc.narg(price_currency)::VARCHAR IS NULL
                OR price_currency = sqlc.narg(price_currency))
    )
ORDER BY created_at DESC;
```

SQL-first meaning confusion

✅ In control of SQL queries and schema

- opposite of ORM / DSL

❌ Schema-first application design

- **domain** drives application design

Domain cart

```
type Cart struct {
    OwnerID string
    Items   []CartItem
}

type CartItem struct {
    ProductID uuid.UUID // google/uuid
    Price     Money

    CreatedAt time.Time
}

type Money struct {
    Amount   decimal.Decimal   // shopspring/decimal
    Currency currency.Unit     // x/text/currency
}
```

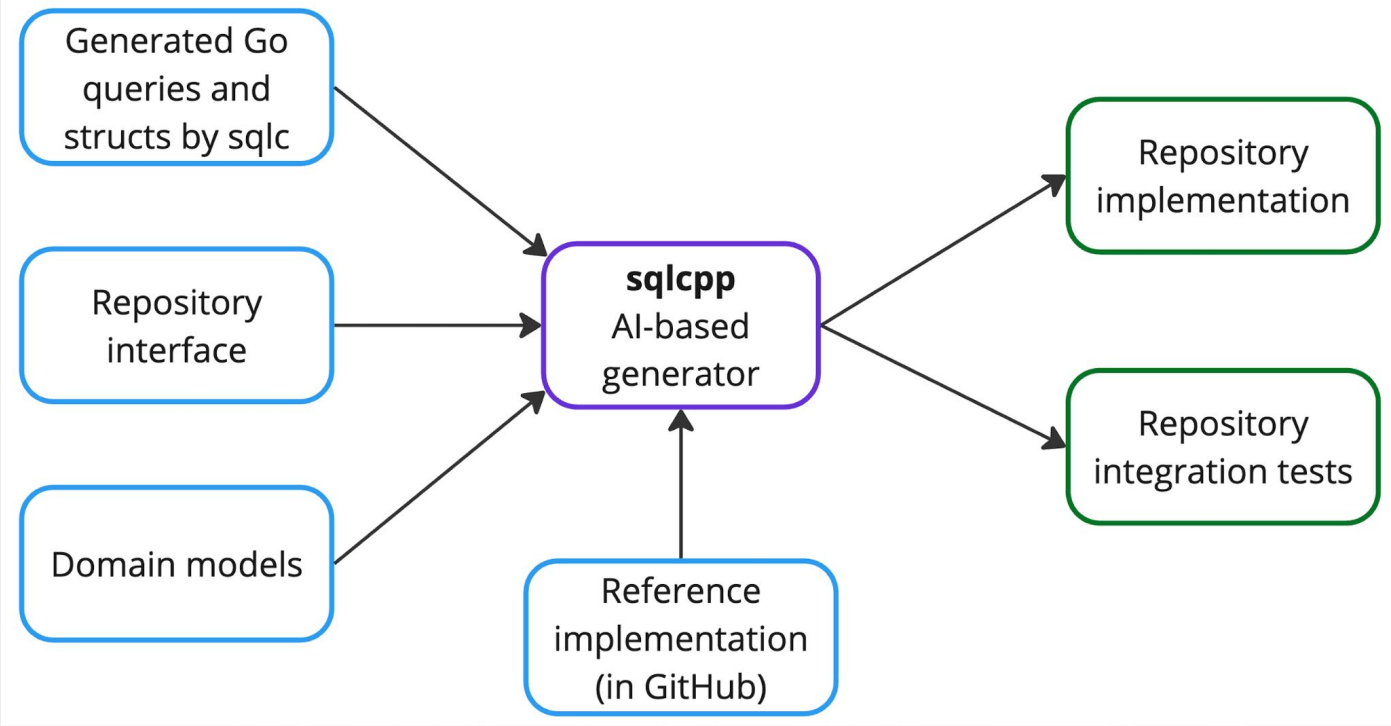
Repository

- Generated queries and structs are useful building blocks
- Repository should accept and return domain models
- Repository method can call multiple queries in Tx

Delegation and mapping

```
func (r *repo) GetCart(ctx context.Context, ownerID string) (domain.Cart, error) {

    // rows []GetCartRow - generated by sqlc
    // q *db.Queries - generated by sqlc
    rows, _ := r.q.GetCart(ctx, ownerID)

    // map sqlc rows -> domain items
    items := mapGetCartRowsToDomain(rows)


    return domain.Cart{OwnerID: ownerID, Items: items}, nil

}
```

sqlc++


Generated Go queries and structs by sqlc
Repository interface
Domain models
sqlcpp
AI-based generator
Reference implementation (in GitHub)
Repository implementation
Repository integration tests

GOLAB 2025

OCTOBER 5th → 7th

NIKOLAY
KUZNETSOV
@ZALANDO

TALK

Beyond sqlc: teaching AI to generate repositories and integration tests in Go

Takeaways

- Type-safe, code-generated boilerplate
- Still need repositories, domain models and mapping
- Try it out: sqlc.dev

Thank you! Q&A

Nikolay Kuznetsov

 nikolayk812

 nkuznetsov

 github.com/nikolayk812/sqlcpp