Christian Beikov

Blaze-Persistence GraphQL

High performance querying and Relay pagination

About us

blazebit @Blazebit
https://github.com/Blazebit

Blazebit Gesbr. is a working group doing commercial software and OSS under APL 2.0

About us

blazebit @Blazebit
https://github.com/Blazebit

Blazebit Gesbr. is a working group doing commercial software and OSS under APL 2.0

Christian Beikov
Curecomp
Blaze-Persistence lead

About us

blazebit @Blazebit
https://github.com/Blazebit

Blazebit Gesbr. is a working group doing commercial software and OSS under APL 2.0

Christian Beikov
Curecomp
Blaze-Persistence lead

Moritz Becker
Ordami
Blaze-Persistence committer

About us

blazebit @Blazebit
https://github.com/Blazebit

Blazebit Gesbr. is a working group doing commercial software and OSS under APL 2.0

Christian Beikov
Curecomp
Blaze-Persistence lead

Moritz Becker
Ordami
Blaze-Persistence committer

Bernd Artmüller
Zoomsquare/Findheim
Designer/Frontend

About us

blazebit @Blazebit
https://github.com/Blazebit

Blazebit Gesbr. is a working group doing commercial software and OSS under APL 2.0

Christian Beikov
Curecomp
Blaze-Persistence lead

Moritz Becker
Ordami
Blaze-Persistence committer

Bernd Artmüller
Zoomsquare/Findheim
Designer/Frontend

Jan-Willem Gmeilig Meyling
Pallas Athena Group
Community member

About us

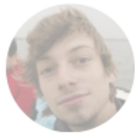
blazebit @Blazebit
https://github.com/Blazebit

Blazebit Gesbr. is a working group doing commercial software and OSS under APL 2.0

Christian Beikov
Curecomp
Blaze-Persistence lead

Moritz Becker
Ordami
Blaze-Persistence committer

Bernd Artmüller
Zoomsquare/Findheim
Designer/Frontend



Jan-Willem Gmeilig Meyling
Pallas Athena Group
Community member

Giovanni Lovato
Axians Italia, Vicenza
Community member

What’s GraphQL

> *GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.*
>
> **https://graphql.org/**

Originated at Facebook and moved to foundation part of Linux foundation

What’s GraphQL

- It’s a typed language, so one needs to define a schema
- Schema allows for tools like GraphiQL which should convince schema haters

- Goals
  - Eliminate over or under-fetching
  - Allow declarative type-checkable querying
  - Support for schema evolution
  - Increase productivity and performance

What's GraphQL

Schema

```
type Query {
    catById(id: ID!): CatWithOwnerView
}
type CatWithOwnerView {
    id: ID!
    name: String!
    owner: PersonSimpleView!
}
type PersonSimpleView {
    id: ID!
    name: String!
}
```

Query

```
query {
    catById(id: 1) {
        name
        owner {
            name
        }
    }
}
```

What's GraphQL

- Define elements you need in structured way based on schema
- Result payload is JSON according to schema structure

```
{
    catById: {
        name: "Cat 1"
        owner: {
            name: "Owner 1"
        }
    }
}
```

GraphQL type system

- Object types - keyword `type`
- Interface types - keyword `interface`
- Union types - keyword `union`
- Enum types - keyword `enum`
- Lists - denoted with `[]`

- Scalars
  - `String`
  - `Float`
  - `Int`
  - `Boolean`
  - `ID`
  - Custom e.g. `Date`

Other operations

Mutation

```
mutation {
    createCat(name: "New Cat") {
        id
        name
    }
}
```

Subscription

```
subscription {
    onCreate {
        id
        name
    }
}
```

- Mutation is like POST/PUT i.e. a modification operation
- Subscription is like a listener for changes

Other operations

Mutation

```
mutation {
    createCat(name: "New Cat") {
        id
        name
    }
}
```

Subscription

```
subscription {
    onCreate {
        id
        name
    }
}
```

- Mutation is like POST/PUT i.e. a modification operation
- Subscription is like a listener for changes
- But focus in this presentation is on queries

Traditional REST design

- Create endpoint for every entity type to list or access by id
- Mirrors the normalized database design and is easy to implement
- Providing relationship data is rarely done, often only id available
- Leads to inefficient lookups done by client, maybe even N + 1 issues
- Data aggregation/composition is done by client in waterfall manner
- Client might have bad reception, so subsequent requests might fail

- `/cats` returns list of cats, but only with owner ids
- `/cats/{id}` returns details of cat and also owner name
- `/person/{id}` returns details of person

GraphQL access design

- Single endpoint for querying data `/graphql`
- Schema based querying to get only needed data in structured form
- GraphQL can be used as API gateway to reduce client latency at first
- Querying all at once can improve query and overall performance a lot
- But you can still stick to traditional REST design behind the scenes
- Subscriptions as a way to show live data
- Transport agnostic, so it doesn’t have to be executed through HTTP, it’s just a string

GraphQL features

- Aliases allow to fetch multiple object graphs at once and assign a name

```
query {
    billy: catById(id: 2) {
        id
        name
    }
    franky: catById(id: 7) {
        id
        name
    }
}
```

```
{
    billy: {
        id: 2
        name: "Billy"
    }
    franky: {
        id: 7
        name: "Franky"
    }
}
```

GraphQL features

- Variables declared with `$` allow to bind values separately
- Fragments allow to reuse object structure definitions

```
query TwoCats($c1: Int, $c2 Int) {
    c1: catById(id: $c1) {
        ...defFields
    }
    c2: catById(id: $c2) {
        ...defFields
    }
}
fragment defFields on CatWithOwnerView {
    id
    name
}
```

```
TwoCats: {
    c1: {
        id: 2
        name: "Billy"
    }
    c2: {
        id: 7
        name: "Franky"
    }
}
```

GraphQL features

- Directives are like annotations in Java and allow defining metadata for client/server

```
query {
    billy: catById(id: 2) @cacheControl(maxAge: 10) {
        id
        name
    }
    franky: catById(id: 7) @cacheControl(maxAge: 10) {
        id
        name
    }
}
```

GraphQL features

- Recent idea that came up, allow passing ids and versions of locally cached values to server
- Server can skip returning the full state of objects and maybe skip fetching data conditionally

```
query {
    catById(id: 2) {
        id
        name
        owner @skip(cached = {
            ids:      [1, 2, 3],
            versions: [2, 4, 1]
        }) {
            name
        }
    }
}
```

```
SELECT
    c.id,
    c.name,
    o.name
FROM Cat c
LEFT JOIN c.owner o
   ON (o.id, o.version) IN (
    (1, 2),
    (2, 4),
    (3, 1)
   )
WHERE c.id = 2
```

Possible misconceptions about GraphQL

- GraphQL does **not** require a specific database like e.g. graph database
- GraphQL is **not** tied to specific frameworks/languages
- GraphQL is **not** going to allow executing arbitrary SQL
- GraphQL is a specification language for communication, not a library or storage system
- Using GraphQL through HTTP allows to reuse existing security integrations e.g. OAuth

Possible misconceptions about GraphQL

- GraphQL does **not** require a specific database like e.g. graph database
- GraphQL is **not** tied to specific frameworks/languages
- GraphQL is **not** going to allow executing arbitrary SQL
- GraphQL is a specification language for communication, not a library or storage system
- Using GraphQL through HTTP allows to reuse existing security integrations e.g. OAuth
- So dont' be shy and give it a try!

Pagination with GraphQL and Relay spec

- Relay is a JavaScript framework by Facebook
- The interesting part is, it has a specification for cursor based pagination

```
query {
  findAll(first: 10) {
    edges {
      node {
        id
        name
      }
    }
    pageInfo {
      startCursor
      endCursor
    }
  }
}
```

Pagination with GraphQL and Relay spec

- Fetches first 10 nodes and delivers cursors of the start and end element

```
{
  findAll: {
    edges: [{
      node: {
        id: 1
        name: "..."
      }, ...
    }]
    pageInfo: {
      startCursor: "..."
      endCursor: "..."
    }
  }
}
```

Specification: https://facebook.github.io/relay/graphql/connections.htm

Pagination with GraphQL and Relay spec

- Use `endCursor` from previous response as argument for `after`

```
query {
  findAll(first: 10, after: "...") {
    edges {
      node {
        id
        name
      }
    }
    pageInfo {
      startCursor
      endCursor
    }
  }
}
```

What does the Relay specification define?

- Relay spec defines the types `Node` and `PageInfo`
- User object types should implement the `Node` interface
- A type with name ending with `Connection` e.g. `CatWithOwnerConnection`
  - Must have `pageInfo: PageInfo!`
  - Must have `edges: [CatWithOwnerEdge]`
- A type with name ending with `Edge` e.g. `CatWithOwnerEdge`
  - Must have `node: CatWithOwnerNode`
  - Must have `cursor: String!`
- Field returning a `Connection` must accept `before` and `after` arguments
- Must also accept `first` and `last` for specifying amount of elements

Implementing Relay types for a small model by hand

```
type Query {
  findAll(first: Int, last: Int, before: String, after: String): CatWithOwnerViewConnection!
}
type PageInfo {
  startCursor: String
  endCursor: String
}
type CatWithOwnerViewConnection {
  edges: [CatWithOwnerViewEdge]
  pageInfo: PageInfo
}
type CatWithOwnerViewEdge {
  node: CatWithOwnerViewNode!
  cursor: String!
}
type CatWithOwnerViewNode {
  id: ID!
  ...
}
```

Implementing Relay types for a small model by hand

- That's a lot of schema one has to define for getting the standard right
- Writing the schema is one part, but keeping up-to-date and in-sync with model is also hard
- How to implement `before` and `after` with cursors?
  - Serialize offset, limit and id to Base64 and re-execute with `id > after`?
  - Serialize keyset to Base64 and implement proper keyset pagination?

Implementing Relay types for a small model by hand

- That's a lot of schema one has to define for getting the standard right
- Writing the schema is one part, but keeping up-to-date and in-sync with model is also hard
- How to implement `before` and `after` with cursors?
  - Serialize offset, limit and id to Base64 and re-execute with `id > after`?
  - Serialize keyset to Base64 and implement proper keyset pagination?
- Doing that manually requires quite some work!

Implementing Relay types for a small model by hand

- That's a lot of schema one has to define for getting the standard right
- Writing the schema is one part, but keeping up-to-date and in-sync with model is also hard
- How to implement `before` and `after` with cursors?
  - Serialize offset, limit and id to Base64 and re-execute with `id > after`?
  - Serialize keyset to Base64 and implement proper keyset pagination?
- Doing that manually requires quite some work!

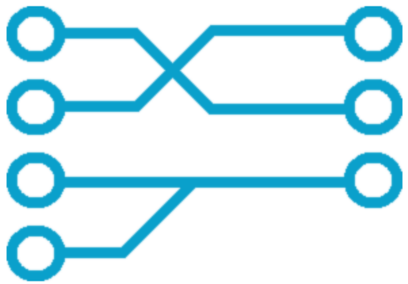
- Blaze-Persistence Entity View integration with GraphQL comes to the rescue!

What is Blaze-Persistence

- Library that builds on top of JPA with deep integration
- Fluent builder API that is easy to use
- Rich pagination support integrated
- Advanced SQL support on top of JPA model
  - SET operations (`UNION`, `INTERSECT`, etc.)
  - CTEs and recursive CTEs (`WITH` clause)
  - `RETURNING` clause support
  - `VALUES` clause and much more!

blaze persistence

What is Blaze-Persistence Entity Views



Entity Views are the ORM equivalent of database views

- Built on top of Blaze-Persistence Core
- Define how DTO maps to entity via JPQL.next expressions
- Querying only the parts you need reduces data transfer
- Flushing only the parts that may change helps as well

Some example use cases:

- Display data in UI
- Fetch data to be returned in REST resources
- Define updatable attributes per use case

Example View - Basic

Entity views are projections of entities

```java
@Entity
public class Cat {
    @Id
    Integer id;
    String name;
    Integer age;
    @ManyToOne(fetch = FetchType.LAZY)
    Cat mother;
    @ManyToOne(fetch = FetchType.LAZY)
    Cat father;
    @ManyToOne(fetch = FetchType.LAZY)
    Person owner;
    @OneToMany
    Set<Cat> kittens;
}
```

```java
@EntityView(Cat.class)
public interface CatView {

    @IdMapping
    Integer getId();

    String getName();

    @Mapping("LOWER(name)")
    String getLowerCaseName();

    @Mapping("SIZE(kittens)")
    Integer getNumKittens();

}
```

Entity Views and GraphQL

- GraphQL more or less requires us to define DTOs
- Blaze-Persistence Entity Views is a technology to implement the DTO pattern
- So let's reuse these DTO definitions to generate GraphQL types!

```
@EntityView(Person.class)
public interface PersonSimpleView {
    @IdMapping
    Long getId();
    String getName();
}
@EntityView(Cat.class)
public interface CatWithOwnerView {
    @IdMapping
    Long getId();
    String getName();
    PersonSimpleView getOwner();
}
```

Entity Views and GraphQL

- Integration with `graphql-java` project by using `GraphQLEntityViewSupport`
- In `DataFetcher` use `GraphQLEntityViewSupport` to create `EntityViewSetting`

```
GraphQLEntityViewSupport graphQLEntityViewSupport;

dataFetcher("findAll", new DataFetcher() {
    @Override
    public Object get(DataFetchingEnvironment dataFetchingEnvironment) {
        EntityViewSetting<CatWithOwnerView, ?> setting =
            graphQLEntityViewSupport.createPaginatedSetting(dataFetchingEnvironment);
        setting.addAttributeSorter("id", Sorters.ascending());
        if (setting.getMaxResults() == 0) {
            return new GraphQLRelayConnection<>(Collections.emptyList());
        }
        return new GraphQLRelayConnection<>(repository.findAll(setting));
    }
})
```

Entity Views and GraphQL

- The `GraphQLEntityViewSupport` is configurable to support custom names
- Relay support is not a must, you can also use an `offset` argument
- `GraphQLEntityViewSupport` also allows configuring custom names for pagination fields
- Use `GraphQLEntityViewSupport.createSetting` for non-paginated queries

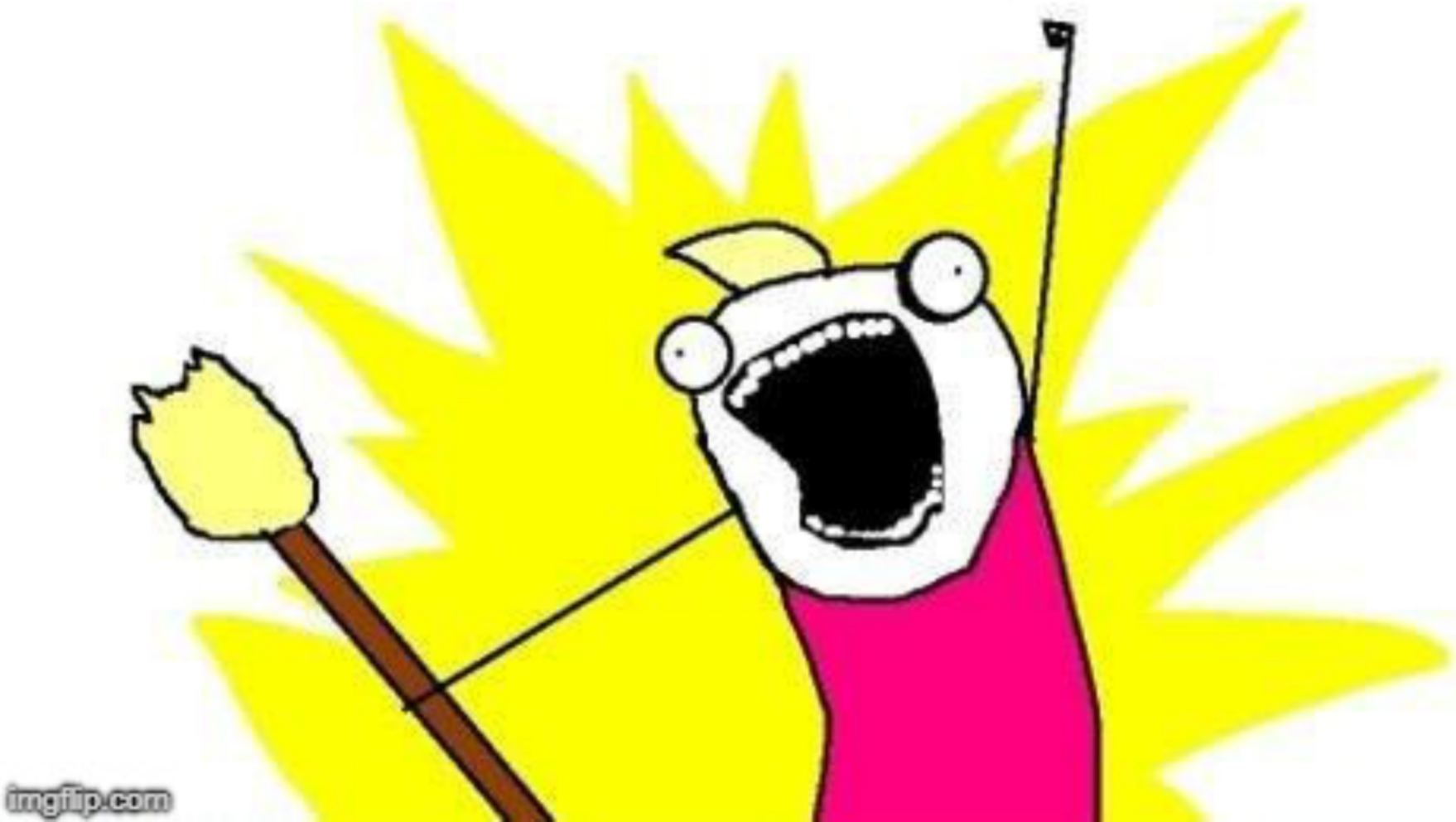
```
GraphQLEntityViewSupport graphQLEntityViewSupport;

dataFetcher("findById", new DataFetcher() {
    @Override
    public Object get(DataFetchingEnvironment dataFetchingEnvironment) {
        return repository.findById(
            graphQLEntityViewSupport.createSetting(dataFetchingEnvironment),
            Long.valueOf(dataFetchingEnvironment.getArgument("id"))
        );
    }
})
```

Entity Views and GraphQL

- The big advantage of entity views is separation of business and projection logic
- Extracting the projection logic from GraphQL queries allows to reuse your existing logic

```
@Component
public class CatViewRepository {
    @Autowired EntityManager em;
    @Autowired CriteriaBuilderFactory cbf;
    @Autowired EntityViewManager evm;
    public <T> List<T> findAll(EntityViewSetting<T, ?> setting) {
        BlazeCriteriaQuery<Cat> cq = BlazeCriteria.get(cbf, Cat.class);
        // Your JPA Criteria logic
        Root<Cat> c = cq.from(Cat.class);
        cq.where(c.get(Cat_.DELETED).in(false));
        // Execute query with entity views driving projection logic
        return evm.applySetting(setting, cq.createCriteriaBuilder(em)).getResultList();
    }
}
```

DEMO TIME



Why use Blaze-Persistence Entity Views for GraphQL?

- Works on top of JPA, so reuse your models and knowledge
- Increase productivity and performance with keyset pagination support of Relay spec implementation
- Improve security and productivity as Entity Views provide abstraction layer
  - Clearly define what properties are accessible
  - Define computed properties via ad-hoc associations called *correlations*
  - Type checked mappings during startup to avoid runtime surprises
- Fetch strategy on attribute definitions allows to fine tune performance
- Integration of selection list of GraphQL to only fetch and join data that was requested
- Allow to leave existing business logic untouched thanks to separation of projection logic

Future improvements based on user demands

- Integration is pretty new, so there might be some things that can be improved
- Entity View Attribute filters
  - Filter on attribute field name e.g. `name(where: "Ab%")`
  - Filter on query level e.g. `findAll(where : {name: "Ab%"})`
  - Maybe allow free form filters? `where: {name: {LIKE: "Ab%"}}`
- Entity View Attribute sorters
  - Sort on attribute field name e.g. `name(orderBy: ASC)`
  - What about multi-sort? Maybe generate enums for field names?
- No mutations integration yet, but looks we could map that to updatable entity vies
- Some users said cache directive integration we proposed would be awesome

I DON'T ALWAYS
IMPROVE PERFORMANCE
BUT WHEN I DO, I
USE ENTITY VIEWS AND
ALSO IMPROVE READBILITY
imgflip.com

Who uses Blaze-Persistence

axians

curecomp
the SRM company

ordami

Pallas Athena Group

Other Blaze projects

- Blaze-Job is a job scheduling toolkit
- Pluggable job model and job storage, API for triggering re-scanning
- Base implementations for in-memory or JPA based job storage
- Recurring jobs through job triggers
- Incremental processing of job instances
- Cluster and partitioning support in job scheduler
- Integration with Spring and Java EE technologies
- Great implementation if you need a job scheduler
- Or bring your own model and storage and use the scheduler in a cluster

Other Blaze projects

- Blaze-Expression is an expression/predicate DSL toolkit
- Make use of custom domain model in DSL via Blaze-Domain
- Serialize expressions to string form for storage
- Serialize expressions to expressions/predicates in Blaze-Persistence queries
- Interpret expressions on custom objects
- Full custom function support
- Like a lightweight subset of Drools that can compile to query predicates
- Brings most building blocks to implement something like JQL(Jira Query Language)

Other Blaze projects

- Blaze-Notify aims to provide a scalable notification engine
- Used at Ordami and Curecomp for transactional E-Mail sending
- Building blocks to be integrated into application for implementing notification campaigns
- Blaze-Expression based DSL for resolving target audience compiled to JPQL.next expression
- Channel implementations for SMTP, AWS SES/SNS, Webhook and bounce handling
- Clustering and partitioning support and integration with Spring and Java EE technologies
- Support for fully transactional at-least-once delivery
- Use advance DBMS features if possible to improve performance i.e. `SKIP LOCKED`
- Optional templating support with implementation for Freemarker

Blaze-Persistence Roadmap

- GraphQL Mutation integration as well as sorting and filtering for queries
- Table function support e.g. `generate_series()`
- `MERGE`/`UPSERT` statement
- Geospatial types and operations
- Range/Interval types and operations
- Lateral joins
- Custom static metamodel
- Debezium integration to update secondary index servers

Community & Support

Slack
We have notifications enabled, so just ask questions :)

GitHub issues
If you've found a bug or question, just create an issue

Stackoverflow
Don't hesitate to ask questions there too, we are listening

Commercial support & training
We help you to get the most out of Blaze-Persistence and Hibernate

Got questions?