What’s new in Hibernate 6.0?

by Christian Beikov

Who am I?

Christian Beikov

Long time Hibernate community contributor

Full time Hibernate developer at Red Hat since 2020

Founder of Blazebit and creator of Blaze-Persistence

Living in Vienna/AT and Bonn/DE

Like to play tennis and go running

Motivation for Hibernate 6.0

Performance improvements

- Read by name is slow in JDBC `ResultSet`
- JPA Criteria implementation required string building and parsing

Design improvements

- Query AST was Antlr2 based and hard to maintain/extend
- Dialect specific SQL translation was hard
- Runtime model was centered around `Type`
- Cleanup of deprecated APIs and SPIs
- More type safety and generics improvements

Read by name

Changing to position based `ResultSet` reading has big impact

- Throughput testing showed it is faster
- Many APIs and SPIs needed to adapt e.g. `Type`
- Allows omitting select aliases and produce smaller SQL

Took the opportunity to implement select item deduplication

- Occasionally requested enhancement
- Reduce amount of fetched data

Read by name

Hibernate 5.x

```
select
    entity0_.id as id1_1_0_,
    entity0_.name as name2_1_0_,
    entity0_.id as onetoone3_1_0_
from
    Entity entity0_
where
    entity0_.id=?
```

Hibernate 6.0

```
select
    e1_0.id,
    e1_0.name
from
    Entity e1_0
where
    e1_0.id=?
```

JPA Criteria

5.x handles JPA Criteria on top of HQL

- `CriteriaQuery` is translated to HQL and then parsed
- But uses a special `CriteriaLoader` with custom fetch handling

6.0 introduces the semantic query model (SQM) as AST model

- SQM AST implements JPA Criteria API
- HQL is also parsed to SQM AST
- Further boost with `hibernate.criteria.copy_tree` disabled
- Smart handling of plain values parameter vs. literal

Query improvements

Semantic query model (SQM) as unified AST for JPA Criteria and HQL

Easier maintenance/extensibility thanks to update to ANTLR4

Functions received lots of new features

- `FunctionReturnTypeResolver` with full AST access e.g. `extract(second)`
- `FunctionArgumentTypeResolver` inference through context e.g. `coalesce(..)`
- `ArgumentsValidator` for early type validation
- Can generate special SQM- and SQL-AST for e.g. emulations

Query improvements

Set operation support (from ANSI SQL-92)

```
select e from Entity e where e.type = 1

union

select e from Entity e where e.type = 2

intersect

select e from Entity e where e.type = 3

except

select e from Entity e where e.deleted
```

Query improvements

Set operation support in JPA Criteria

```
HibernateCriteriaBuilder cb = session.getCriteriaBuilder();

var q1 = cb.createQuery( Entity.class );

var q2 = cb.createQuery( Entity.class );

// Apply restrictions to q1 and q2 ...


TypedQuery<Entity> q = session.createQuery( cb.union( q1, q2 ) );
```

Query improvements

`LIMIT`, `OFFSET` and `FETCH` clause support (ANSI SQL 2003)

```
select p.name, p.score

from Player p

order by p.score desc

fetch first 3 rows with ties
```

name	score
Thor	10
Hulk	10
Tony	7
Vision	7

Query improvements

`LIMIT`, `OFFSET` and `FETCH` clause support

```
HibernateCriteriaBuilder cb = session.getCriteriaBuilder();

JpaCriteriaQuery<Entity> q1 = cb.createQuery( Entity.class );

JpaRoot<Entity> root = q1.from( Entity.class );

q1.fetch( 3, FetchClauseType.ROWS_WITH_TIES );


TypedQuery<Entity> q = session.createQuery( q1 );
```

Query improvements

Support for `OVER` clause a.k.a. window functions (ANSI SQL 2003)

```
select
  format(c.ts as 'yyyy''Q''Q'),
  sum(c.amount),
  sum(c.amount) - lag(sum(c.amount))
    over (order by format(c.ts as 'yyyy''Q''Q')) as delta
from Costs c
group by format(c.ts as 'yyyy''Q''Q')
```

period	amount	delta
2021Q4	100	NULL
2022Q1	120	20

Query improvements

Ordered set-aggregate functions i.e. `LISTAGG` (ANSI SQL 2016)

```
select pr.id, listagg(p.name, ', ') within group (order by p.name)
from Project pr join pr.participants p
group by pr.id
order by pr.id
```

id	participants
1	Bruce Banner, Natasha Romanoff, Tony Stark
..	..

Query improvements

Summarization support i.e. `ROLLUP` (ANSI SQL-99)

```
select
  s.state,
  s.city,
  sum(s.price * s.quantity)
from Sales s
group by rollup (s.state, s.city)
order by s.state, s.city nulls last
```

state	city	amount
CA	SF	100
CA	SJ	120
CA	NULL	220

Query improvements

`FILTER` clause support (ANSI SQL 2016)

```
select
  count(*) filter (where s.price * s.quantity < 10) as small_sales,
  count(*) filter (where s.price * s.quantity >= 10) as big_sales
from Sales s
```

small_sales	big_sales
10	20

Query improvements

`ILIKE` predicate support

```
select e

from Entity e

where e.name ilike '%tony%'
```

Fallback to `lower(e.name) like lower('%tony')`

Query improvements

Tuple syntax support with great emulation

Align expressions/predicates i.e. `... where my_function(value > 1) and ...`

Temporal arithmetic support i.e. `ts1 - ts2` returns `Duration`

Duration extraction support with `BY` operator i.e. `(ts1 - ts2) by hour`

Duration literal support i.e. `ts1 + 1 day`

Support for `DISTINCT FROM`predicate

etc.

Query improvements

`Session#createSelectionQuery` for select only queries

- No `executeUpdate` method

`Session#createMutationQuery` for mutation only queries

- No `getResultList`/`getSingleResult` etc. methods

Early validations and communication of intent

Dialect specific SQL

SQL-AST as intermediate representation of (modern) SQL as tree

5.x has a single HQL to SQL string translator with lots of if-branches per dialect-support

Dialects in 6.0 provide custom translators for handling emulations

SQL-AST enables sophisticated transformations/emulations efficiently

- Introduction of dummy `FROM` elements (Sybase)
- Emulate aggregate functions through window functions (MySQL, SQL Server, …)

Runtime model

Model in 5.x was centered around `org.hibernate.type.Type`

Created OO runtime model since switch to read by position required changes anyway

Moved logic to runtime model defined through capabilities i.e. `Fetchable`

Removed multi-column basic mappings in favor of custom embeddable mappings

Split logic from `BasicType` into `JdbcType`/`JavaType` and removed most implementations

Introduce `BasicTypeReference` in `StandardBasicTypes` for late resolving

Mapping improvements

Support for new SQL types through Dialect contribution

- `JSON` type for storing any type as `JSONB`/`JSON`/`CLOB`/`VARCHAR`
- `UUID` type for `java.util.UUID`
- `INTERVAL_SECOND` type for `java.time.Duration`
- `INET` type for `java.net.InetAddress`

Replace `@TypeDef` with type safe variants i.e. `@JavaTypeRegistration`

Replace `@Type` with type safe variants i.e. `@JavaType`

Mapping improvements

`CompositeUserType` bridges gap between custom types and embeddables

Embeddable mapper class for defining mapping structure

Full control on construction and deconstruction of custom type

- Support for library types i.e. `MonetaryAmount`
- Repurpose existing types i.e. `OffsetDateTime`

Out of the box support for Java records in discussion

Demo time

https://github.com/beikov/presentation-hibernate-6

Other changes

Switch to Jakarta Persistence

Remove deprecated stuff i.e. legacy Criteria API

Add type variables to APIs/SPIs where possible

`Dialect` was majorly updated

Renaming of `JavaTypeDescriptor` and `SqlTypeDescriptor`

See https://github.com/hibernate/hibernate-orm/blob/main/migration-guide.adoc

What’s next?

Subqueries in the from clause

Common table expressions

Lateral joins

Insert-or-Update a.k.a. Upsert

Table functions

SQL structs

SQL arrays

Q & A