JOOQ-JAVA OBJECT ORIENTED QUERYING

Anuj Sharma[1], Paras Nath Barwal[2]

[1]*Project Associate, E-governance, CDAC, Noida*
[2]*Joint Director, E-governance, CDAC, Noida*

Abstract

JOOQ is a query language as derived from SQL or JOOQ is a query language in which SQL uses the objects of OOQ. As there are many issues in integration of java application with various databases via JDBC. This paper proposes a new java library known as JOOQ. The primary aim of JOOQ focuses on integration issues in java programming language. With JOOQ, programmers can write SQL queries in a simpler and faster way.

JOOQ thus decreases the programming effort by 10-20% which leads to higher quality of code with low error rate. Thus the application is also significantly light and less prone to errors.

Keywords: *Java, JDBC, SQL, Database, Error, JOOQ.*

----------***----------

1. INTRODUCTION

Java Object Oriented Querying, usually known as JOOQ, is a light database-mapping programming library in Java that actualizes the dynamic record design. The main intention of this paper is to present both social and research work situated by giving a space particular language to build queries from classes produced from a database composition.

One of the feature JOOQ offers that we like most is the capacity to create code specifically from existing database tables (like how JAXB functions with XSD patterns). The ensuing Java code authorizes sort wellbeing in SQL questions alongside abstracting the underlying information store execution. This conduct permits a developer to utilize the same code with Mysql or Postgres in creation, and H2 or comparable being developed.

JOOQ guarantees that SQL ought to start things out in any database reconciliation. Consequently, it doesn't present another literary question language, yet rather takes into consideration building plain SQL from JOOQ protests and code produced from a database diagram. JOOQ utilizes JDBC to call the underlying SQL inquiries. While it gives deliberation on top of JDBC, JOOQ does not have to the extent that and many-sided quality as standard research work social mapping libraries, for example, Hibernate and JPA.

JOOQ's closeness to SQL has favorable circumstances over run of the mill object-social mapping libraries. SQL has numerous peculiarities that can't be utilized as a part of an item arranged programming standard; this set of contrasts is alluded to as the research work social impedance befuddle. By being near SQL, JOOQ serves to anticipate language structure lapses and sort mapping issues. Additionally, variable tying is dealt with. It is additionally conceivable in JOOQ to make exceptionally perplexing inquiries, that include associating, unions, settled choose and complex joins. JOOQ additionally upholds database-particular peculiarities, for example, UDTs, enum sorts, put away techniques and local capacities.

2. BACKGROUND

In spite of the fact that protest turned programming languages, for example, Java, Smalltalk, and C++ and C # have ruled the server-side business application world for a long time, shockingly question arranged database did not take off because of various reasons. Social database still is the larger part of persistence mechanism.

The standard programming interface to database is SQL, which was initially intended for human cooperation purposes. As a set arranged language, with extremely constrained control stream and detached language structure, it gradually turns into a hindrance for an immaculate turned framework plan. Inside the information access layer, over and over again we discover engineers discard the lovely protest model, however begin connecting SQL strings (either specifically or by implication) and make such a variety of chaotic codes. Far more detestable, on account of the limitable of SQL, numerous engineers use information access layer, or the information model itself as the beginning stage when outlining a framework. This ground-up methodology has a tendency to make the last framework more administration arranged or more procedural.

JOOQ wraps the fundamental SQL language into bland item arranged APIs, and conceals the underline complexities, for example, porting to contrast databases. It goes about as a mapping apparatus between items to lines in database tables, additionally has the ability of controlling information with control stream. Since the yield of JOOQ compiler is Java classes, designers can without much of a stretch coordinate JOOQ into their applications.

There are times where a bloated ORM is the right decision for an undertaking. Different times, it simply includes an execution punishment and impedes conversing with the underlying information store. There are other comparative schemas (Mybatis, Slick for Scala) however we have discovered JOOQ to be decently finish peculiarity savvy and exceptionally receptive to bugs.

3. DESIGN GOALS

The research work is about JOOQ as a basic language and simple to learn and utilization for Java designers. The punctuation and language structure is near Java rather than SQL. Really all the rationale of creating SQL strings are totally escaped JOOQ developer.

3.1 Object-Oriented

Everything is an objects in JOOQ; there is no primitive sort. All operations are characterized at class level, for example, Create, Retrieve, and Update and Delete (standard CRUD in SQL). Those exceptionally entangled question semantics are pleasantly wrapped in Criteria, Projection and Join interfaces. From engineers' point of view, they are just managing an uncommon set of research works, which happened to be put away in a social database transparently to their configuration model.

3.2 Database Generic

JOOQ will be aggregated into Java classes, which utilize ANSI standard SQL and can run against any social databases without adjustment. This makes application porting simple, yet it additionally suggests a restriction of JOOQ, we can't help any database particular gimmicks or changes. In future discharges, we will add database flavor choice to permit streamlining for particular databases.

3.3 Less Overhead

The code produced by JOOQ compiler ought not to be much slower than transcribed local SQL squares. Since compiler produces all SQL explanations (in view of advancement, for example, utilizing arranged articulations, or table join requesting) as opposed to linking them at runtime, JOOQ can really give better execution sometimes.

3.4 Simple

JOOQ has comparative straightforward grammar and language structure like Java. It has less decisive word, and does not help primitive information sort. Since it fills a solitary need of showing SQL inquiry, the API is straightforward and clean.

3.5 Easy to Integrate

Since JOOQ compiler produces Java classes (or other OO language yield in future discharges), it is not difficult to incorporate it into existing frameworks as information access layer.

3.6 Portable

Java is a convenient language runs on most working frameworks. With its database nonexclusive peculiarity, JOOQ is very versatile to numerous platforms.

4. IMPLEMENTATION EXAMPLE

a) A nested query selecting from an aliased table:

```
-- Select authors with books that are sold out
SELECT * FROM AUTHOR x
    WHERE EXISTS (SELECT 1
            FROM BOOK
           WHERE BOOK.STATUS = 'SOLD'
             AND BOOK.AUTHOR_ID = x.ID);
```

b) And its equivalent in JOOQ DSL:

```
// Use the aliased table in the select statement
create.selectFrom(table("AUTHOR").as("x"))
      .where(exists(selectOne()
                .from(table("BOOK"))
where(field("BOOK.STATUS").equal(field("BOOK_STAT
US.SOLD")))

.and(field("BOOK.AUTHOR_ID").equal(field("AUTHOR.I
D")))));
```

c) More simply, using code generation from the database metadata to generate constants:

```
// Use the aliased table in the select statement
final Author x = AUTHOR.as("x");

create.selectFrom(x)
      .where(exists(selectOne()
                .from(BOOK)

.where(BOOK.STATUS.equal(BOOK_STATUS.SOLD))
                .and(BOOK.AUTHOR_ID.equal(x.ID))));
```

5. PARTITIONING WITH SQL

JOOQ is an extraordinary structure when it needs to work with SQL in Java without having an excessive amount of ORM in its way. In the meantime, it might be incorporated into numerous situations as it is putting forth to help for some database-particular peculiarities. One such database-particular gimmick is dividing in PostgreSQL. Apportioning in PostgreSQL is for the most part utilized for execution reasons in light of the fact that it can enhance question execution in specific circumstances.

5.1 Partitioning in PostgreSQL

With the dividing gimmick of PostgreSQL that have the likelihood of part information that would structure an immense table into various separate tables. Each of the parts is a typical table which inherits its sections and stipulations from a guardian table. This supposed table legacy could be utilized for "reach parceling" where, for instance, the information from one extent does not cover the information from an alternate extends as far as identifiers, dates or other criteria.

Like in the accompanying illustration, it can have parceling for a table "creator" that has the same remote key of a table "authorgroup" in all its columns.

```
CREATE TABLE author (
        authorgroup_id int,
        LastName varchar(300)
);

CREATE TABLE author_1 (
        CONSTRAINT authorgroup_id_check_1
        CHECK ((authorgroup_id = 1))
        INHERITS (author);
 );

CREATE TABLE author_2 (
        CONSTRAINT authorgroup_id_check_2
        CHECK ((authorgroup_id = 2))
        INHERITS (author);
 );
```

As should be obvious, it set up legacy and – with a specific end goal to have a basic example – it simply put one imperative watching that the segments have the same "authorgroup_id". Essentially, these results in the "creator" table have just table and section definitions, however no information. On the other hand, when questioning the "creator" table, PostgreSQL will truly question all the inheriting "author_n" tables giving back a joined together come about.

5.2 A Trivial Approach to using JOOQ with Partitioning

To work with the apportioning depicted above, JOOQ offers a few alternatives. Programmer can utilize the default way which is to let JOOQ create one class for every table. So as to embed information into numerous tables, user would need to utilize diverse classes. This methodology is utilized as a part of the accompanying scrap:

```
InsertQuery query1 = dsl.insertQuery(AUTHOR_1);
query1.addValue(AUTHOR_1.ID, 1);
query1.addValue(AUTHOR_1.LAST_NAME, "Data");
query1.execute();

InsertQuery query2 = dsl.insertQuery(AUTHOR_2);
query2.addValue(AUTHOR_2.ID, 1);
query2.addValue(AUTHOR_2.LAST_NAME, "Data");
query2.execute();

// select
Assert.assertTrue(dsl
.selectFrom(AUTHOR_1)
.where(AUTHOR_1.LAST_NAME.eq("Data"))
fetch().size() == 1);

Assert.assertTrue(dsl
.selectFrom(AUTHOR_2)
.where(AUTHOR_2.LAST_NAME.eq("Data"))
.fetch().size() == 1);
```

Here the code shows that various classes created by JOOQ need to be utilized, so relying upon what number of allotments to be done, produced classes can infect codebase. Additionally, envision that inevitably need to repeat over parts, which would be unwieldy to do with this methodology. An alternate methodology could be that utilizes JOOQ to construct fields and tables utilizing string control yet that is mistake inclined again and anticipates help for nonexclusive sort security. Additionally, consider the situation where it needs genuine information detachment as far as multi-tenure.

It can be seen that there are a few contemplations to do when working with dividing. Luckily JOOQ offers different methods for working with divided tables, and in the accompanying we'll analyze approaches, so that can pick the one most suitable for database queries.

5.3 Using JOOQ with Partitioning and Multi-Tenancy

JOOQ's runtime-composition mapping is regularly used to acknowledge database situations, such that for instance amid advancement, one database is questioned yet when sent to creation, the questions are going to an alternate database. Multi-occupancy is an alternate suggested use case for runtime mapping as it takes into consideration strict apportioning and for arranging user application to just utilize databases or tables being arranged in the runtime-composition mapping. So running the same code would bring about meeting expectations with diverse databases or tables that relying upon the setup which took into consideration genuine detachment of information regarding multi-tenure.

The accompanying setup taken from the JOOQ documentation is executed when making the Dslcontext so it might be viewed as a framework wide setting:

```
Settings settings = new Settings()
.withRenderMapping(new RenderMapping()
.withSchemata(
.new MappedSchema().withInput("RAM")
.withOutput("MY_WORLD")
.withTables(

new MappedTable().withInput("AUTHOR")
.withOutput("AUTHOR_1"))));

// Add the settings to the Configuration
DSLContext create = DSL.using(
connection, SQLLanguage.ORACLE, settings);
 // Run queries with the "mapped" configuration
create.selectFrom(AUTHOR).fetch();

// results in SQL:
// “SELECT * FROM MY _WORLD.AUTHOR_1”
```

Utilizing this methodology user can delineate table to one segment forever e.g. "Creator" to "Author_1" nature "RAM". In an alternate environment user could decide to guide "Creator" table to "Author_2".

Runtime-outline mapping just permits user to guide to precisely one table on a for every question premise, so user couldn't deal with the utilization situation where user would need to control more than one table parcel. In the event that user might want to have more adaptability user may need to consider the following methodology.

5.4 Using JOOQ with Partitioning and without Multi-Tenancy

In the event that user have to handle numerous table parcels without having multi-tenure, user require a more adaptable method for getting to segments. The accompanying sample demonstrates how user can destroy it an element and sort safe way, staying away from lapses and being usable in the same rich way user are utilized to by JOOQ:

```
// add
for(int i=1; i<=2; i++)
{
Builder part = forPartition(i);
        InsertQuery query =
        dsl.insertQuery(part.table(AUTHOR));
query.addValue(part.field(AUTHOR.ID), 1);
query.addValue(part.field(AUTHOR.LAST_NAME),
"Data");
query.execute();
}

// select
 for(int i=1; i<=2; i++)
{
Builder part = forPartition(i);
Assert.assertTrue(dsl
.selectFrom(part.table(AUTHOR))
.where(part.field(AUTHOR.LAST_NAME).eq("Well"))
.fetch()
.size() == 1);
}
```

What user can see above is that the segment numbers are inattentive away so user can utilize "Creator" table rather than "Author_1". Accordingly, user code won't be dirtied with numerous produced classes. Something else is that the practitioner research work is introduced powerfully so user can utilize it for instance within a circle like above. Likewise it takes after the Builder design so user can work on it like user are utilized to by JOOQ.

The code above is doing precisely the same as the first inconsequential scrap, yet there are numerous profits like sort sheltered and reusable access to parceled tables.

5.5 Integration of JOOQ Partitioning without Multi-Tenancy into a Maven Build Process (Optional)

In the event that user are utilizing Continuous-Integration user can incorporate the result above so JOOQ is not creating tables for the divided tables. This might be accomplished utilizing a customary interpretation that prohibits certain table names when producing Java classes. At the point when utilizing Maven, user incorporation may look something like this:

```
<generator>
<name>org.jooq.util.DefaultGenerator</name>
<database>
        <name>org.jooq.util.postgres.PostgresDatabase</n
        ame>
        <includes>.*</includes>
        <excludes>.*_[0-9]+</excludes>
        <inputSchema>${db.schema}</inputSchema>
</database>
<target>
<packageName>com.user.company.jooq</packageName>
<directory>target/generated-sources/jooq</directory>
</target>
</generator>
```

At that point its simply calling mvn introduce and JOOQ expert plugin will be creating the database outline in aggregation time.

5.6 Integrating JOOQ with PostgreSQL: Partitioning

This research work portrayed how JOOQ in fusion with the parceling peculiarity of PostgreSQL might be utilized to actualize multi-tenure and enhance database execution. PostgreSQL's documentation expresses that for dividing "the profits will ordinarily be advantageous just when a table would generally be huge. The accurate time when a table will profit from dividing relies on upon the application; in spite of the fact that a dependable guideline is that the extent of the table ought to surpass the physical memory of the database server."

Accomplishing backing for dividing with JOOQ is as simple as including design or a little utility class, JOOQ is then ready to help parceling with or without multi-occupancy and without giving up sort wellbeing. Separated from Java-level incorporation, the depicted result likewise easily coordinates into user manufacture and test methodology.

6. CONCLUSIONS

JOOQ gives an influential option to the Object-Relation access layer. It is unadulterated Object-Oriented, easy to learn, database and stage nonexclusive. What's more, since JOOQ absorbed a lot of people great plans from existing OR ventures, this makes it a decent application for any new Java venture which requires social database as constancy layer.

Ultimately, while JOOQ does not help transaction semantics it is reasonably simple to wrap it within Spring Transactions. Obviously one could combine Spring Transactions with Spring JDBC, however would lose the expressiveness and force of the JOOQ API.

REFERENCES

[1] Zhongling Li. (February 5, 2006). "Object-Oriented Query Language," IEEE A Technical White Paper.

[2] Lucas Eder. (April, 2014). "Integrating jOOQ with PostgreSQL: Partitioning," In DZone research work magazine, "SQLZone".

[3] Rohit Pai. (May 2013). "Brief introduction to jOOQ", LiftHoff Communications Magazine, 05.

[4] S. Ozdemir. (July, 2014). "Nested query approach using jOOQ", Journal of Information Science and Engineering 25.

[5] Berts. (July, 2014). "ORM frameworks in jOOQ", IEEE transaction.

[6] Codegen Maven. (2011). "jOOQ effective repository used to manage database connectivity and library".

[7] Petri Kainulainen. (July 2014). "Typesafe dabase and library of jOOQ". Hawaii International Conference on Database System Sciences (HICSS '00).

[8] Majid Azimi. (January, 2014). "JOOQ must become de facto standard for database querying", IEEE transaction.

[9] Alessio Harri ,. (May 2014). "OpenJPA is the workhorse and JavaOOQ is the artist", ACM/Baltzer Journal of Database management, Vol. 1, No. 3, pp. 255-265.

[10] Grambery Gmbh. (2014). "Object-relational mapping using jOOQ", Proceedings of the SBT/IEEE International Database Management Symposium.

[11] F. Zhan, C. Noon. (1996). "jOOQ's reason for being - compared to JPA", Database Science.

[12] L-V. Israel. (2006). "Implementation of jOOQ on SQL queries". PhD thesis, University of Siegen.

[13] G. Chen C. Li, M. Ye and J. Wu. (2005). "jOOQ's reason of being - compared to LINQ". In 2nd IEEE International Conference on Database Query Languages, ICMASS, pages 8–15.

[14] G. Chen M. Ye, C. Li and J. Wu. ,. (2005). "jOOQ as an innovative solution for a better integration of Java applications". In Performance, Computing Conference, PCCC, pages 535–540.

[15] J. M. Ng C. P. low, C. Fang and Y. H. Ang. (2008). "Integration on stack overflows problem using jOOQ". Computer Science, 31:750–759.