ORM Tools

1 Definition and Importance

ORM (Object-Relational Mapping) tools are tools that convert relational database to object-oriented approach and the developer can manipulate the database without writing raw SQL queries and instead use the functions that the ORM provide, also some of those tools can generate code from the database schema to interact with it.

Hence the developer will use those functions, it will simplify database operation (CRUD) and it will enhance the readability and writability and make the development easier and faster.

2 Some of the popular ORM Tools

We will see different ORM tools and also tools that are not exactly ORM tools but it serves the same goal.

2.1 Exposed

It's a lightweight ORM which means it has the basic features of ORM but without the overhead of full ORM that may reduce the performance, and it offers two approaches which are SQL DSL (Domain Specific Language) and DAO (Data Access Object).

The DSL keeps SQL concepts while adding type safety and the DAO provides an object-oriented approach similar to ORM.

Advantages:

1- No optimization overhead because it's lightweight.
2- Easier integration
3- Provides two approaches.

Disadvantages:

1- Low flexibility

2- Relatively new and still evolving.

Example:

Define a table function:

```
import org.jetbrains.exposed.sql.Table

const val MAX_VARCHAR_LENGTH = 128

object Tasks : Table("tasks") {
    val id = integer("id").autoIncrement()
    val title = varchar("name", MAX_VARCHAR_LENGTH)
    val description = varchar("description", MAX_VARCHAR_LENGTH)
    val isCompleted = bool("completed").default(false)
}
```

Query the table:

```
fun main() {
    Database.connect("jdbc:h2:mem:test", driver = "org.h2.Driver")

    transaction {
        SchemaUtils.create(Tasks)

        val taskId = Tasks.insert {
            it[title] = "Learn Exposed"
            it[description] = "Go through the Get started with Exposed tutorial"
        } get Tasks.id

        val secondTaskId = Tasks.insert {
            it[title] = "Read The Hobbit"
            it[description] = "Read the first two chapters of The Hobbit"
            it[isCompleted] = true
        } get Tasks.id

        println("Created new tasks with ids $taskId and $secondTaskId.")

        Tasks.select(Tasks.id.count(), Tasks.isCompleted).groupBy(Tasks.isCompleted).forEach {
            println("${it[Tasks.isCompleted]}: ${it[Tasks.id.count()]} ")
        }
    }
}
```

2.2 Hibernate

It’s an ORM that implements JPA (Java Persistence API) and has more features on top of it, also it has HQL (Hibernate Query Language) that is similar to SQL but it’s object-oriented language.

Advantages:

1- High flexibility
2- Object-oriented approach
3- Large community

Disadvantages:

1- Complexity
2- Performance overhead
3- Steep learning curve

Example:

Define a table:

```java
@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
}
```

Query the table:

```
public class App {
    public static void main(String[] args) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction transaction = session.beginTransaction();

        Employee employee = new Employee("John Doe", 50000);
        session.save(employee);

        transaction.commit();

        Employee fetchedEmployee = session.get(Employee.class, employee.getId());
        System.out.println("Fetched Employee: " + fetchedEmployee);

        session.close();
        HibernateUtil.getSessionFactory().close();
    }
}
```

2.3 jOOQ

It’s an internal DSL and source code generator which offers the ability to write SQL queries using Java code and make it type-safe.

Advantages:

1- Compile time type safety.
2- Code generation.
3- Easy integration.

Disadvantages:

1- No Built-in Relationship Management
2- Require a good knowledge of SQL.
3- Bigger amount of code.

Example:

Query the table

```
public class App {
    public static void main(String[] args) {
        try (DSLContext context = JooqUtil.getDSLContext()) {
            context.insertInto(EMPLOYEES)
                   .columns(EMPLOYEES.NAME, EMPLOYEES.SALARY)
                   .values("Alice Johnson", 60000.00)
                   .execute();

            Result<Record> result = context.select().from(EMPLOYEES).fetch();

            for (Record record : result) {
                int id = record.get(EMPLOYEES.ID);
                String name = record.get(EMPLOYEES.NAME);
                double salary = record.get(EMPLOYEES.SALARY);
                System.out.println("Employee: " + id + ", " + name + ", " + salary);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.4 Ktorm

A lightweight ORM designed for Kotlin and provides a SQL DSL.

Advantages:

1- Designed for Kotlin so it supports its features such as extension functions and lambda expressions.
2- Type-safe because it uses DSL.
3- No optimization overhead

Disadvantages:

1- Low flexibility
2- Smaller community

3- No Built-in Relationship Management

Example:

Define the table:

```
object Employees : Table<Nothing>("employees") {
    val id = int("id").primaryKey()
    val name = varchar("name")
    val salary = decimal("salary")
}
```

Query the table:

```
fun main() {
    db.insert(Employees) {
        set(Employees.name, "Alice Johnson")
        set(Employees.salary, 60000.00.toBigDecimal())
    }

    val employees = db.from(Employees).select()

    for (row in employees) {
        val id = row[Employees.id]
        val name = row[Employees.name]
        val salary = row[Employees.salary]
        println("Employee: $id, $name, $salary")
    }
}
```

2.5 SQLDelight

It is a type-safe SQL tool that generates APIs from SQL statements that must be written in a certain file.

Advantages:

1- High performance because of direct database access
2- Type safety

Disadvantages:

1- Requires good knowledge of SQL because it requires writing raw SQL.
2- No Built-in Relationship Management

Example:

Create SQLDelight schema:

```
CREATE TABLE employees (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    salary REAL NOT NULL
);

selectAllEmployees:
SELECT * FROM employees;

insertEmployee:
INSERT INTO employees (name, salary) VALUES (?, ?);
```

Execute the queries:

```
fun main() {
    val db = Database.database

    db.employeeQueries.insertEmployee("Alice Johnson", 60000.00)

    val employees = db.employeeQueries.selectAllEmployees().executeAsList()

    employees.forEach {
        println("Employee: ${it.id}, ${it.name}, ${it.salary}")
    }
}
```

3 Comparison

Feature	Exposed	Hibernate	jOOQ	Ktorm	SQLDelight
Type	Lightweight ORM	Full ORM	SQL DSL	Lightweight ORM	SQL based
ORM features	Limited	Full features	No ORM features	Limited	No ORM features
Relationship management	Yes	Yes	No	No	No
Language	Kotlin	Java with Kotlin support	Java with Kotlin support	Kotlin	Kotlin
Code generation	No	Yes	Yes	No	Yes
Learning curve	Low	High	Medium	Low	Medium