AuDao Whitepaper

The idea behind AuDao is to speed up the very boring development of the **database layer** between a relational database (managed by SQL) and a business logic source code (in our case managed by Java). The developer / programmer creates **one configuration file** and everything **else is automatically generated**. The developer does not need to be aware of the database type, handle most of potential DB exceptions (SQLException), and can rely on maximum possible compile time syntax checks, such as column names, types, foreign keys, and similar. The tool does not introduce any new SQL hyper language; it just provides a smart link between SQL and Java.

AuDao also contains an importing tool called inspector, which allows **creating** the **configuration file from a database** instance. Once you set the access values, the import is automatic. The importing tool makes possible to create whole database layer without writing a single line of code.

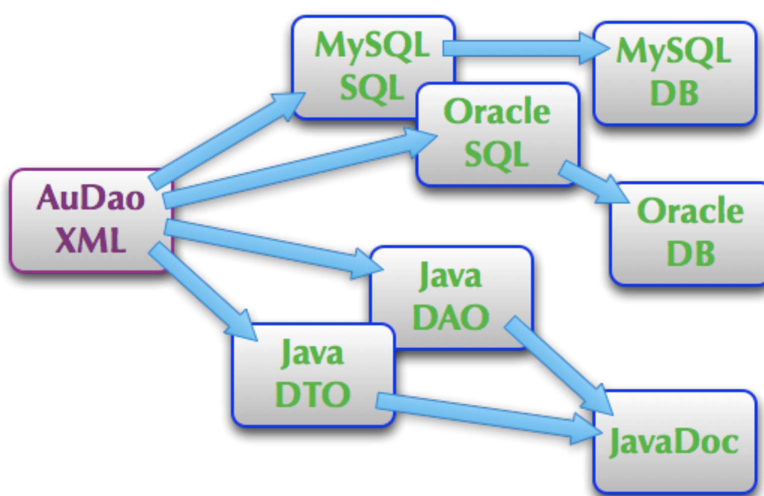


Figure 1 Blue arrows show automated code generation

AuDao automatically generates SQL Data Definition Language (DDL) and source code for Data Access Objects (DAO) / Data Transfer Objects (DTO) libraries. The input is a configuration file (XML) describing the database entities and relations. The output is a set of SQLs (CREATE/DROP/INSERT) and a Java JAR library plus Javadoc (optionally).

Currently the following databases are supported:

- MySQL - version 3, 4, 5
- Oracle - version 9i, 10g and 11g

and the following DAO / DTO programming language is supported:

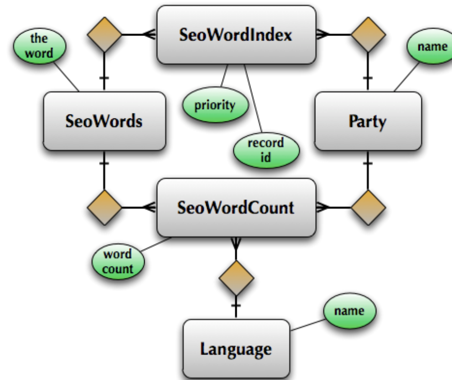
- Java - version 1.5, 1.6

© 2009 All rights reserved

Four Easy Steps

1. Design the relational database to fit your needs.

The first step is to create or update an Entity Relationship diagram on a paper or in any tool of your preference. It includes design of entities, relations, and indexes; structure optimization for large volume of data, and analysis of possible bottleneck. The step does not differ from the usual practice.




```
        <data>
            <row><c>100</c><c>ENGLISH</c></row>
            <row><c>110</c><c>FRENCH</c></row>
            <row><c>120</c><c>FINNISH</c></row>
            <row><c>190</c><c>CZECH</c></row>
        </data>
    </table>


    <table name="parties" java="Party">
        <columns>
            <column name="party_id">
                <type>int</type>
                <pk/>
            </column>
            <column name="party_name">
                <type max-length="32">String</type>
                <not-null/>
            </column>
        </columns>
        <indexes>
            <index name="inx_party_name">
                <unique/>
                <columns>
                    <column name="party_name"/>
                </columns>
            </index>
        </indexes>
    </table>

    <table name="seo_words">
        <columns>
            <column name="word_id">
```

2. Configure XML

Instead of writing SQL and Java code, the next step is to open your favorite XML editor. Proceed with definition of the database according to the XSD.

Another option is to create the XML by the importing tool from an already existing instance of the database.

This is the only coding needed and the amount of it is equivalent to hand written SQL.

3. Consult documentation

Documentation and plenty of examples are available for faster and easier implementation.

XSD:

```
<xs:complexType name="DeleteType">
  <xs:sequence>
    <xs:element name="comment" type="xs:string" minOccurs="0"/>
    <xs:element name="unique" minOccurs="0"/>
    <xs:choice>
      <xs:element name="all"/>
      <xs:element name="dynamic"/>
      <xs:element name="condition" type="ConditionType"/>
      <xs:element name="pk"/>
    <xs:choice>
  <xs:sequence>
  <xs:attribute name="name" type="NameType" use="required"/>
<xs:complexType>
```

Complex Type "MoveType"

Used by type TableType

Sequence: (comment?, (all | dynamic | condition))

Attributes:

Name	Required	Type	Default Value	Fixed Value	Description
name	yes	NameType			The name of the move - the java name is constructed as "move" + ucfirst(@name).
target	yes	TableNameType			The target table.

Elements:

Name	Card	Type	Description
comment	0 - 1	xs:string	A comment which is copied to the java code.
all	1	-	Moves all records.
dynamic	1	-	A generic move - allows to pass SQL condition + parameters.
condition	1	ConditionType	Explicit move - allows to pass parameters to predefined SQL.

XSD:

```
<xs:complexType name="MoveType">
  <xs:sequence>
    <xs:element name="comment" type="xs:string" minOccurs="0"/>
    <xs:choice>
      <xs:element name="all"/>
      <xs:element name="dynamic"/>
      <xs:element name="condition" type="ConditionType"/>
    <xs:choice>
```

© 2009 All rights reserved

4. Generate

The generator, typically as an ant task, generates the following:

- SQL scripts for your chosen data source
- SQL scripts execution
- Complete Java source code for your DAO / DTO objects
- Compiled and packed Java library
- Javadoc documentation for the Java source / library

The database layer is complete. Any changes should be done to the configuration XML file only.

Features

Build task

- The provided ant build file contains tasks to create / drop database
- DB location and access configuration stored separately from the DB definition, more data sources (e.g. test and production) supported

General

- For each table a DTO, DAO and DAO-implementation classes are generated.
- DaoFactory dynamically chooses which DAO-implementation is used (MySQL, Oracle).
- Inheritance of tables is supported.
- Abstract tables/classes are supported.
- Views are supported.
- Column references to the same or other tables are supported (column type is taken from the referenced table).
- Initial data can be also defined.

DTO

- Copy constructors allow easily copying and casting one object to another one - for table tree hierarchy.
- Support for enumerations.
- Overloaded "set" methods for types java.sql.Date and java.sql.Timestamp allows to easily assign also java.util.Date.
- Smart "toString()" method prints only non-null attributes.
- Inner enum "Column" allows to use the real database column names in safe way.
- Support for GWT - compilable to JavaScript.

© 2009 All rights reserved

DAO

- Finder methods are automatically created according to defined indexes and primary keys.
- Explicit finders can be defined:
  - "all" - returns all records
  - "query" - prepared SQL query + parameters
  - "dynamic" - dynamic SQL query + parameters
  - "ref" - N:M reference table is used to fetch the result
- The result set can be sorted according to predefined criteria
- Explicit "update" methods can be defined (similar possibilities as for finders)
- Explicit "delete" methods can be defined (similar possibilities as for finders)
- Explicit "truncate" method can be defined
- Explicit "move" methods which moves one or more items from one to another DB table can be defined (similar possibilities as for finders)
- "auto" columns for integer types allow to generate them automatically (mostly used for primary keys)
- "auto" columns for Date and Timestamp allows to set the current date and time automatically when inserting and/or updating
- DAO-implementation – MySQL:
  - "auto" integer columns are implemented using "autoincrement" feature
- DAO-implementation – Oracle:
  - "auto" integer columns are implemented using sequences
  - sorting: internationalization support using NLSSORT()

Contacts

If the AuDao looks interesting to you, please let us know. Currently, we use it internally, however we are ready to share it with a broader community. Let us know your interest on email:

audao@spolecne.cz

Společně s.r.o.

© 2009 All rights reserved