ERICSSON

USER’S GUIDE, DATA DICTIONARY FOR MNESIA

Purpose

This document describes how to use the data dictionary for the Mnesia database: rdbms, version 1.5.

This dictionary was first called sysMnesiaDict, and was part of the AXD 301 source code tree.

Author: Ulf Wiger <ulf.wiger@ericsson.com>

Contents

1 SUMMARY

The document describes how to use rdbms, a custom data dictionary for the Mnesia database.

2 READER'S GUIDELINES

Proper setup of the import tool assumes basic knowledge in Erlang and familiarity with the target database.

3 INTRODUCTION

Mnesia is a database engine for distributed real-time applications developed in Erlang. The tool described in this document was designed to provide additional type and integrity checking.

Features provided by rdbms that you otherwise will not get with mnesia include:

- Type checking, all the standard erlang data types, including records and custom validation functions.
- Referential integrity checks modeled after the SQL standard.
- Commit and abort triggers, which execute within the context of the transaction, also supporting nested transactions.
- Compound attributes that can be used e.g. for referential integrity checks.

4 TOOL OVERVIEW

rdbms stores metadata as user properties in the mnesia schema. Properties are stored in the form {Key, Value}, where Key is one of

- {attr, Name, Property}
- {tab, Name, Property}
- {record, Name, Property}

For example:

{{attr, age, type}, integer}

{{record, person, attrs}, [email, name, age, phone]}

{{tab, person, access}, Access_control_list}   % not implemented

4.1 CENTRAL FUNCTIONS

The following functions are central to rdbms.erl and will make sure that specified integrity constraints are met:

4.1.1 add_properties(Properties)

Inserts a list of properties into the dictionary. For details on valid properties, see chapter 4.2.

This function creates rdbms if necessary.

Returns {atomic, true} | {aborted, Reason}.

4.1.2 do_add_properties(Properties [, Table])

Similar to add_properties/1, but this function is meant to be called from within a schema transaction. Example:

```
add_properties(Properties) ->
    F = fun() ->
        do_add_properties(Properties)
    end,
    mnesia_schema:schema_transaction(F).
```

do_add_properties(Properties, Table) adds table properties for

4.1.3 activity(Fun)

Starts a mnesia transaction which executes the function Fun, using rdbms as a callback module for mnesia operations.

Note that this function must be used instead of mnesia:transaction() or mnesia:activity() in order to benefit from the metadata stored in the dictionary.

4.1.4 create_table(Name, Options)

Analogous to mnesia:create_table(Name, Options), but also recognizes the option {rdbms, Properties}.

4.1.5 do_create_table(Name, Options)

Like create_table/2, but intended to be called from within a schema transaction, example:

```
mnesia_schema:schema_transaction(
    fun() ->
            rdbms:do_create_table(company,
                                  [{attributes, [name,address,...]},
                                   {rdbms, [{attr,name,string}]}]),
            rdbms:do_create_table(
                emp,
                [{attributes,[name,company,...]},
                 {rdbms,
                  [{{attr,{emp,company},references},
                    [{company,name,[{match,full},
                                    {delete,ignore},
                                    {update,no_action}]
                     }]},
                   {{attr,{company,name},add_references
                    [{emp,company, [{match,full},
                                    {delete,cascade}}
                                    {update,ignore}]}
                  ]}
                ])
    end).
```

This allows for complex database schema manipulation with full rollback semantics.

4.1.6 delete_table(Name)

Analogous to mnesia:delete_table(Name). This function also makes sure that references to the deleted table are removed from all remaining tables.

4.1.7 do_delete_table(Name)

Like delete_table(Name), but designed to be called from within a schema transaction.

4.2 HELPER FUNCTIONS

The following functions are meant to provide access to metadata outside the dictionary, as well as offer some functionality which is not currently in mnesia.

4.2.1 select(Table, Attribute, Value, SelectAttrs)

Fetches objects from Table where Attribute == Value, and returns the information specified in SelectAttrs (a list or tuple of attribute names).

If SelectAttrs is a tuple, the function will return a tuple with the specified attribute values for each object; if SelectAttrs is a list, a list will be returned instead.

Example:

given a table with records #person{id,surname, lastname,company,tfn}

```
rdbms:select(person, surname, "Joe", {surname,lastname,company})
[{"Joe", "Armstrong", "Ericsson"}]
```

Both Attribute and SelectAttrs may specify compound attributes (see below).

4.2.2 fetch_objects(Table, Attribute, Value)

Fetches objects from Table where Attribute == Value.

Attribute may specify any attribute, even a compound attribute (see below). The most appropriate retrieval method is chosen automatically.

4.2.3 null_value()

Returns a special null value which is used by rdbms to signify missing information. Per definition, this is a value which is distinguishable from any legal value. While it is impossible for an Erlang application to generate such a value, rbms makes an honest attempt by returning ’#.[].#’.

4.2.4 attributes(Table)

Returns a list of attributes, in order, for the table Table.

4.2.5 all_attributes(Table)

Returns a tuple, {PhysicalAttributes, LogicalAttributes}, where PhysicalAttributes is the same as attributes(Table), and LogicalAttributes is a list of attributes which are not physically part of the record – e.g. compound attributes. LogicalAttributes is represented as a list of tuples, {AttrName, Info}, e.g. {name, {compound, [surname, lastname]}}

4.2.6 attribute(Pos, Table)

Returns the name of the attribute in position Pos of Table.

4.2.7 position(AttrName, Table)

Returns the position of attribute AttrName of Table.

4.2.8 type(attr, Attribute)

Returns the type of Attribute.

Attr can be either the attribute name or {Table, Attribute}.

4.2.9 default(attr, Attribute)

Returns the default value for Attribute.

Attr can be either the attribute name or {Table, Attribute}.

If no default value for Attribute has been specified in the dictionary, null() will be chosen, unless Attribute is of type record, in which case a default record (possibly containing only nulls) will be returned.

4.2.10 default(record, Table)

Returns a default record for Record, similar to #<Record>{}.

4.2.11 required(attr, Attr)

Returns true | false.

This property is not checked by verify(Record), but should be checked explicitly by user applications.

4.2.12 bounds(attr, Attr)

Returns {inclusive, {Min, Max}} | {exclusive, {Min, Max} | undefined.

{inclusive, {Min, Max}} means Min =< Value =< Max;

{exclusive, {Min, Max}} means Min < Value < Max.

4.3 THE #RDBMS_OBJ{} RECORD

A special record, #rdbms_obj{} has been defined to represent data objects in a more detailed fashion. The structure of the #rbms_obj{} is such:

name	name of the data object
attributes	[{AttrName, Type, Value}]

The purpose of this representation is to allow for generic code to operate on data objects directly, e.g. GUI functions.

4.3.1 make_object(Type [, Values])

Returns an #rdbms_obj{} record, either with default values, or with the values specified in Values = [{AttrName, Value}]

4.3.2 make_record(TabName | RdbmsObj)

Returns a record -- either a default record for a given table, or a record generated from a given #rdbms_obj{}.

4.4 VALID PROPERTIES

Properties are given as {Key, Value} tuples, where the key consists of a {Class : attr | rec | tab, Name, Property} tuple.

An alternative syntax is {{Class, Name}, [{Property, Value}]}.

4.4.1 Attribute Properties

Property	Values	Description
type	atom | list | tuple | integer | float | term |	Plain types
	string	byte list (verified using list_to_binary(Value))
	text	atom or string (byte list)
	number	integer or float
	{record, Rec : atom()}	The attributes for record Rec must be defined in the dictionary.
	{compound, [SubAttr]}	A set of values derived from attributes in a given record. See 4.4.1
	oid	Automatic type, required, defaults to a unique value
required	true | false	If true, null_value() is not allowed
default	true | false	default value. If omitted, null_value() is used.
bounds	{inclusive,Min,Max} | {exclusive, Min, Max}	Bounds checking - not restricted to type
references	[{Tab2, Attr2, Actions}]	see 4.5.2
add_references	[{Tab2,Attr2, Actions}]	see 4.5.2
drop_references	[{Tab2,Attr2}]	see 4.5.2

4.4.2 Record Properties

Property	Values	Description
verify	F : function/1	F(Record) is called in addition to the standard verification, and is expected to return true | false. See 4.4.3
attributes	[AttrName : atom()]	Used to define records which do not have their own table definition.
action_on_read	F : function/2	If specified, mnesia:read({Tab, Rec}) will result in a call to F(Tab, Rec). See 4.4.4
action_on_write	F : function/2	If specified, mnesia:write(Rec) will result in a call to F(Rec). See 4.4.4
action_on_delete	F : function/2	If specified, mnesia:delete(Tab, Key) will result in a call to F(key, Key), and mnesia:delete_object(Obj) will result in a call to F(obj, Obj). See 4.4.4

4.4.3 Table Properties

Property	Values	Description
action_on_read	F : function/2	If specified, mnesia:read({Tab, Rec}) will result in a call to F(Tab, Rec)

Property	Values	Description
action_on_write	F : function/2	If specified, mnesia:write(Rec) will result in a call to F(Rec)
action_on_delete	F : function/2	If specified, mnesia:delete(Tab, Key) will result in a call to F(key, Key), and mnesia:delete_object(Obj) will result in a call to F(obj, Obj)

4.5 ADVANCED PROPERTIES

4.5.1 Compound Attributes

Compound attributes should not be listed in the ’attributes’ list, whether it be the mnesia ’attributes’ table property (for normal tables), or the rdbms ’attributes’ property (for custom record types).

Compound attributes can be used in the select() and fetch_objects() functions, as well as in definitions of referential integrity.

4.5.2 Referential Integrity

Referential integrity rules are stored as metadata in the following way:

```
{attr, {Tab, Attr}, [{Tab2, Attr2, RefActions}]},
where

Tab    : the referencing table
Attr   : the referencing attribute
Tab2   : the referenced table
Attr2  : the referenced attribute(s) -
         atom() | {atom()} | function(Object,Value)

RefActions : {Match, DeleteAction : Action,
              UpdateAction : Action}
Match  : handling of null values - partial | full
Action : referential action - no_action | cascade |
set_default | set_null
```

An alternative syntax for RefActions is a {Key,Value} list:

```
RefActions ::= [{match, partial | full} | {delete, Action} | {update,
Action}]
```

If any of the ’delete’ or ’update’ options are omitted, no_action is used as default.

The ’match’ option only matters when the reference is made using a compound attribute (where nulls are allowed):

- If omitted, a match is made if (a) any component of the referencing attribute is null, or (b) the referencing attribute equals the referenced attribute.
- If ’partial’ is specified, a match is made if (a) every component of the referencing attribute is null, or (b) each non-null component of the referencing attribute is equal to its counterpart in the referenced attribute.
- If ’full’ is specified, a match is made if (a) every component of the referencing attribute is null, or (b) the referencing attribute equals the referenced attribute.

References can also be added to existing tables using the option {{attr,{Tab,Attr},add_references},[{Tab2,Attr2,Actions}]}. This is useful especially when tables are defined e.g. as callback functions from different modules, and one does not wish to pre-declare references (rdbms will fail if there are references pointing to non-existent tables.)

Dropping references can be done in a similar fashion using the option {{attr,{Tab,Attr},drop_references},[{Tab2,Attr2}]}.

4.5.2.1 Example of Referential Integrity

A common pattern is illustrated for edification: Given a ’company’ table and an ’employee’ table, we want to make sure that employees cannot be added to a company that does not exist, and we want to make sure that when a record is deleted from the ’company’ table, all related employee records are also deleted.

Table attributes:

company:	[name, address, org_code]
employee:	[name, company, position, age, address]

We want to link company.name and employee.company two ways, with different semantics depending on whether we are operating on the ’company’ or on the ’employee’ table. In both directions, the ’match’ option is irrelevant, as we are not dealing with compound attributes. The default value for ’match’ is ’full’.

- When creating a new ’company’ record, we do not want to enforce any check against the ’employee’ table (**update: ignore**).
- When deleting a ’company’ record, we want to delete all related ’employee’ records (**delete: cascade**).
- When creating a new ’employee’ record, we want to check whether there is a corresponding ’company’ record (**update: no_action**).
- When deleting an ’employee’ record, we do not want to enforce any checks agains the ’company’ table (**delete: ignore**).

In other words, we arrive at the following referential attributes:

company:	{{attr,{company,name},references}, [{employee, company, {full,cascade,ignore}}]}
employee:	{{attr,{employee,company},references}, [{company, name, {full,ignore,no_action}}]}

4.5.3 Verification Functions

Verification functions are not allowed to modify the object being verified, and can only return ’true’ or ’false’. They should avoid operations which have side-effects, since a mnesia transaction can be restarted.

4.5.4 Custom Access Functions

The following access functions can be specified for a table:

- {action_on_read, function/1}
- {action_on_write, function/2}
- {action_on_delete, function/2}

Since these functions are expected to have side-effects, it may be necessary to register a hook for committing or undoing the effect of the functions. This can be done with the functions:

- rdbms:register_commit_action(F)
- rdbms:register_rollback_action(F)

More than one action of each type can be registered, and they will be forgotten after the completion of a transaction. The actions are called in LIFO (last in, first out) order **after** commit/rollback. No attempt is made to catch exits.

4.5.5 Example

DataInit methods for tables dpLmTable and eqm_dp_addr_table:

```
init_tables()->
    Nodes = mnesia:table_info(schema, disc_copies),
    catch begin
        create_table(dpLmTable,
                     [{disc_copies, Nodes},
                      {snmp, [{key, {integer, integer, integer}}]},
                      {attributes, record_info(attrs, dpLmTable)}]),
        create_table(eqm_dp_addr_table,
            [{disc_copies, Nodes},
            {attributes, record_info(attrs, eqm_dp_addr_table)}])
    end.

create_table(Name, Attribs) ->
    case mnesia:create_table(Name, Attribs) of
        {atomic, _} ->
            ok;
        {aborted, Reason} ->
            throw({error, {Name, Reason}})
    end.

init_data()->
    Props = [
        { {attr, {dpLmTable, dpLmKey}, type}, tuple },
        { {attr, {dpLmTable, dpLm}, type}, string },
        { {attr, {dpLmTable, dpLmRowStatus}, type}, integer },

        { {attr, {eqm_dp_addr_table, dp_id}, type}, {record, dp_id} },
        { {attr, {eqm_dp_addr_table, dp_address}, type}, tuple },

        % definition of the record dp_id
        { {record, dp_id, attrs}, record_info(attrs, dp_id)},
        { {attr, subrack_no, type}, integer },
        { {attr, em_slot_no, type}, integer },
        { {attr, piu_slot_ no, type}, integer },
        { {attr, local_dp_no, type}, integer }],
    rdbms:add_properties(Props).
```

Note that attrs does not need to be specified for records which correspond to a Mnesia table, since this information is already stored in the Mnesia schema. Only use this property when defining records which do not correspond to a Mnesia table.

The type and the default value have to be specified since Mnesia don’t have that information.

4.6 REQUIRED FILES

- rdbms.beam

5 TERMINOLOGY

-