Erlang Solutions Ltd.

Erlang Term Storage

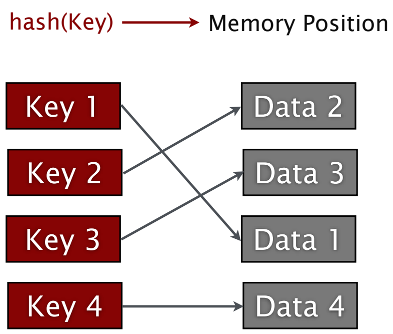
Overview: Erlang term storage

- ETS Tables
- Handling Elements
- Searching and Traversing
- Match Specifications and Select
- Other Issues
- Table Visualiser

ETS Tables

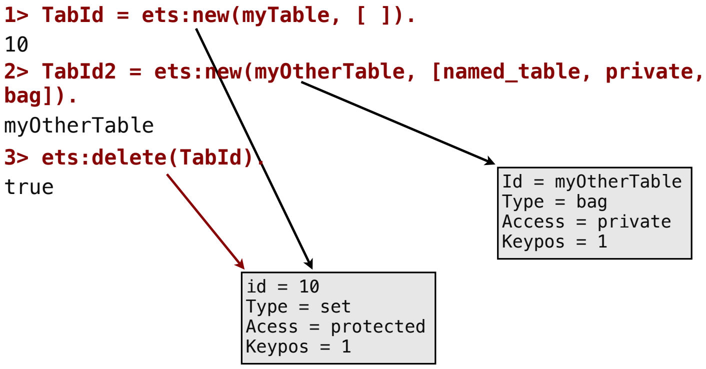
- Provides a mechanism to store large data quantities
  - Data is stored as tuples
- Data is stored in dynamic tables and accessed though keys as hash tables or binary trees
- Constant lookup time regardless of table size
- Has a low level search mechanism
- No transaction handling

ETS Tables



- Tables can be sets, bags or ordered sets
- Sets and bags are implemented as hash tables
  - A hash function maps the key to the element's memory position
- Ordered sets are arranged as binary trees

Creating and Deleting: example



```
1> TabId = ets:new(myTable, [ ]).
10
2> TabId2 = ets:new(myOtherTable, [named_table, private,
bag]).
myOtherTable
3> ets:delete(TabId).
true
```

```
Id = myOtherTable
Type = bag
Access = private
Keypos = 1
```

```
id = 10
Type = set
Acess = protected
Keypos = 1
```

ETS tables: creating & deleting

- Table Options can be:
  - **set**, where every key is unique
  - **ordered_set**, keys are unique, traversed linearly
  - **bag**, duplicate keys can exist, elements are unique
  - **duplicate_bag**, duplicate elements can coexist
- Access rights include:
  - **public**, every process can read and write
  - **protected**, everyone can read, owner can write
  - **private**, only owner can read and write
  - **{keypos,Pos}**, which tuple element is the key
  - **named_table** statistically registers the name

Handling Elements: example

```
1> ets:new(countries, [set, named_table]).
countries
2> ets:insert(countries, {luigi, italy}).
true
3> ets:lookup(countries, dieter).
[]
4> ets:lookup(countries, luigi).
[{luigi,italy}]
5> ets:insert(countries, {luigi, austria}).
true
6> ets:lookup(countries, luigi).
[{luigi,austria}]
7> ets:delete(countries, luigi).
true
```

```
Id = countries
Type = set
Access = protected
Keypos = 1
==================
{luigi, italy}
{luigi, austria}
```

Handling Elements

```
ets:insert(TabId | TableName, Tuple)
ets:delete(TabId | TableName, Key)
```

- **insert/2** inserts an element in the table
- The tuple must be of size greater than or equal to the key position
- In sets or ordered sets, inserting elements with the same key or identical elements will result in the old elements being deleted
- **delete/2** removes the element from the table

Handling Elements

```
ets:lookup(TabId | TabName, Key)
```

- Searches the table for elements with the key
- For sets, the return value is **[]** or **[Tuple]**
- For bags, the return value is **[]** or a list of tuples
- Constant lookup time for sets and bags
- Proportional lookup time to the log(Size) for ordered_sets

Handling Elements: example

```
1> TabId = ets:new(people, [bag]).
10
2> ets:insert(TabId, {luigi, france}).
true
3> ets:insert(TabId, {luigi, france}).
true
4> ets:insert(TabId, {luigi, italy}).
true
5> ets:lookup(TabId, luigi).
[{luigi,france},{luigi,italy}]
```

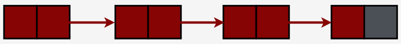
- In a bag, the same object can not occur more than once
- Time order of the object insertion is preserved
  - If {X,Y} is inserted after {X,Z}, a lookup will return [{X,Z}, {X,Y}]

Traversing Tables: example

```
1> ets:new(jobs, [named_table, ordered_set]).
jobs
2> ets:insert(jobs, [{cesarini, axd301}, {lelle, anx}]).
true
3> ets:insert(jobs, [{anders, gprs}, {ola, axd301}]).
true
4> K1 = ets:first(jobs).
anders
5> K2 = ets:next(jobs, K1).
cesarini
6> ets:next(jobs, lelle).
ola
7> ets:next(jobs, ola).
'$end_of_table'
8> ets:last(jobs).
ola
```

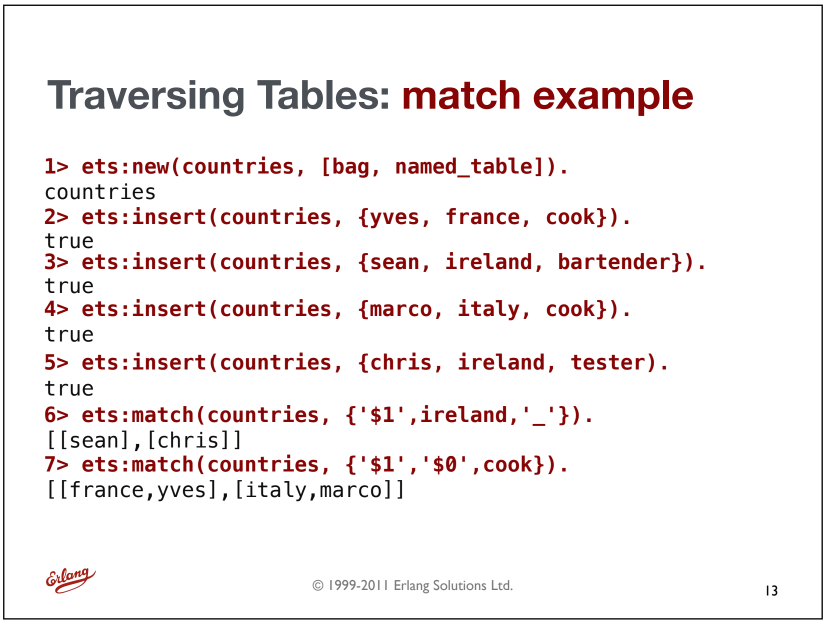
```
Id = jobs
Type = ordered_set
Access = protected
Keypos = 1
==================
{anders, gprs}
{cesarini, axd301}
{lelle, anx}
{ola, axd301}
```

Traversing Tables



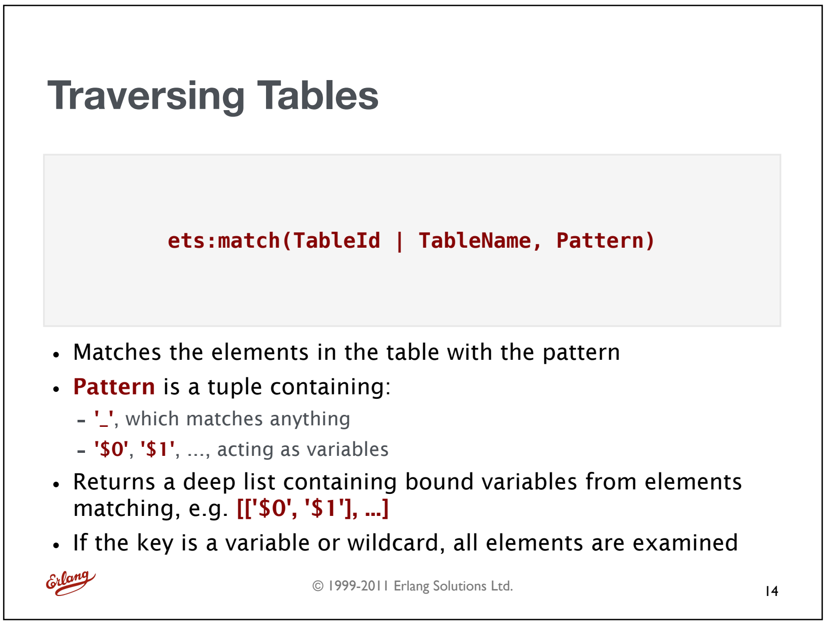
```
ets:first(TableId | TableName)
ets:next(TableId | TableName, Key)
```

- Returns the first/next key or **'$end_of_table'**
- In ordered sets, keys are returned in lexicographical order
- In bag sets, the hash order is returned
- **last/1** returns the last element in ordered_sets and the first element in bags and sets



Traversing Tables: match example

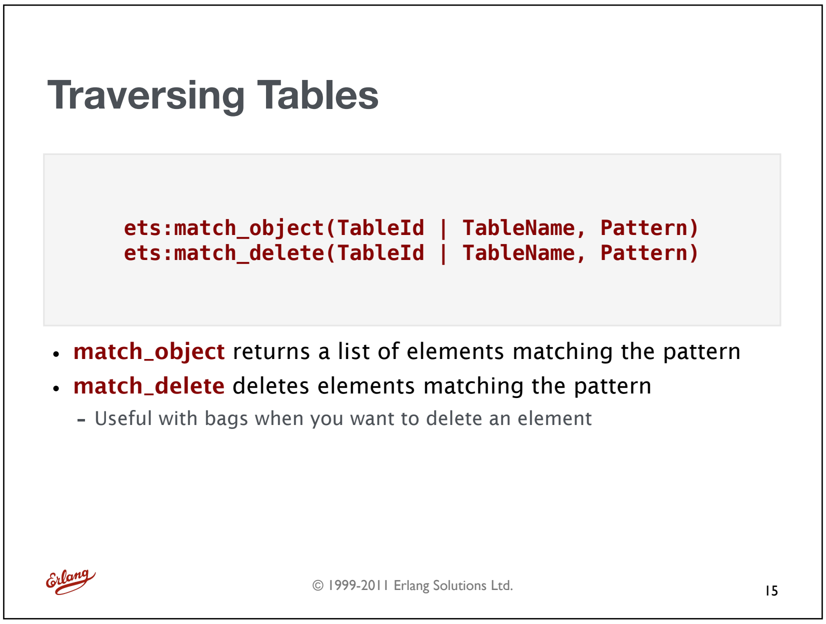
```
1> ets:new(countries, [bag, named_table]).
countries
2> ets:insert(countries, {yves, france, cook}).
true
3> ets:insert(countries, {sean, ireland, bartender}).
true
4> ets:insert(countries, {marco, italy, cook}).
true
5> ets:insert(countries, {chris, ireland, tester).
true
6> ets:match(countries, {'$1',ireland,'_'}).
[[sean],[chris]]
7> ets:match(countries, {'$1','$0',cook}).
[[france,yves],[italy,marco]]
```



Traversing Tables

```
ets:match(TableId | TableName, Pattern)
```

- Matches the elements in the table with the pattern
- **Pattern** is a tuple containing:
  - '_', which matches anything
  - '$0', '$1', ..., acting as variables
- Returns a deep list containing bound variables from elements matching, e.g. [['$0', '$1'], ...]
- If the key is a variable or wildcard, all elements are examined



Traversing Tables

```
ets:match_object(TableId | TableName, Pattern)
ets:match_delete(TableId | TableName, Pattern)
```

- **match_object** returns a list of elements matching the pattern
- **match_delete** deletes elements matching the pattern
  - Useful with bags when you want to delete an element

Traversing Tables

WARNING!!
Use match with extreme care!

- All match operations are implemented as BIFs
- BIFs disrupt the real time properties of the system
  - Match operations on big tables stop other processes from executing
- Use first/next to traverse big tables

Match Specifications: example

```
1> ets:new(countries, [bag, named_table]).
countries
2> ets:insert(countries, {yves, france, cook}).
true
3> ets:insert(countries, {sean, ireland, bartender}).
true
4> ets:insert(countries, {marco, italy, cook}).
true
5> ets:select(countries, [{{'$1', '$2', '$3'},
                           [{'==', '$3', cook}],
                           [['$2', '$1']]}]).
[[france,yves],[italy,marco]]
```

Match Specifications

```
[{{'$1','$2','$3'},
  [{'==','$3','cook'}],
  [['$2','$1']]}]
```

- A match specification consists of an Erlang Term
- Describes a "programme" that tries to match
- Compiled to something more efficient than a function
- Powerful, but complex to write, and often unreadable
- Match specifications can be generated from literal anonymous functions

Match Specifications: fun2ms

```
ets:fun2ms(LiteralFun)
```

- Translates a literal fun into a match specification
- The fun is transformed at compile time and can not be dynamic
  - It must be statically declared in the call to fun2ms in a module
  - ets:fun2ms/1 in the shell, with funs defined in the shell still works
- Fun can only take one argument a tuple of arguments
- A header file must be included:
  - `-include_lib("stdlib/include/ms_transform.hrl")`

Match Specifications: example

```
1> ets:new(countries, [bag, named_table]).
countries
2> ets:insert(countries, {yves, france, cook}).
true
3> ets:insert(countries, {sean, ireland, bartender}).
true
4> ets:insert(countries, {marco, italy, cook}).
true
5> MS = ets:fun2ms(fun({Name, Country, Job}) when Job ==
cook -> [Country, Name] end).
[{{'$1','$2','$3'},[{'==','$3',cook}],[['$2','$1']]}]
6> ets:select(countries, MS).
[[france,yves],[italy,marco]]
```

Select

```
ets:select(TableId | TableName, MatchSpec)
ets:select(TableId | TableName, MatchSpec, Limit)
ets:select(Continuation)
```

- select/2 is a more general version of match that uses a match specification
- select/3 takes a limit on how many answers are returned, and returns the matched list and a continuation
- select/1 takes a continuation from a limited select and returns the next Limit elements that match

Other Issues

```
ets:tab2file(TableId | TableName, FileName)
ets:file2tab(FileName)
ets:tab2list(TableId | TableName)
```

- **tab2file/2** dumps a table on file
  - returns **ok | {error, Reason}**
- **file2tab/1** reads it
  - returns **{ok, Tab}** or **{error, Reason}**
- **tab2list/1** returns a list with all the elements of the table

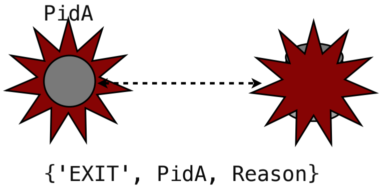
Other Issues

```
9> ets:info(countries).
{{memory,320},{owner,<0.48.0>}, {name,countries},{size,4},
 {node,nonode@nohost},{named_table,true},{type,bag},
 {keypos,1}, {protection,protected}}
10> ets:i().
 id        name       type size mem   owner
----------------------------------------------------------
 8         code       set   250  9854  code_server
 9         code_names set    37  3822  code_server
 ac_tab    ac_tab     set     6   842  application_contr...
 countries countries  bag     4   320  <0.48.0>
...
```

Other Issues

- Records can also be inserted in ETS tables
- Set the key position of the tuple representation when creating the table
  - Use the **#RecordType.KeyField** information directive
- If you want to insert records from the shell, you must use the tuple representation or load the definition in the shell
- If you want to match record tables, remember to set the fields to '_':
  - `#Record{name=Name, phone='$1', _ = '_'}`

Other Issues



- Tables are linked to the process which created them
- If the process terminates, the table is automatically deleted
- Be careful when creating and using ETS tables from the shell

Other Issues

- Tables are not garbage collected
- They must be deleted manually
- With over 20 elements, ETS tables are more efficient than lists
- ETS operations are implemented in BIFs
- ETS tables are stored in RAM
- Disk only ETS tables are implemented in the **dets** module

Table Visualizer

- A graphical tool used to examine ETS and Mnesia tables
  - It includes tables in connected nodes
- Allows creating and editing of tables
- Polls the tables for changes
  - Changes are visible through a colouring scheme
- Can view table information

Table Visualizer

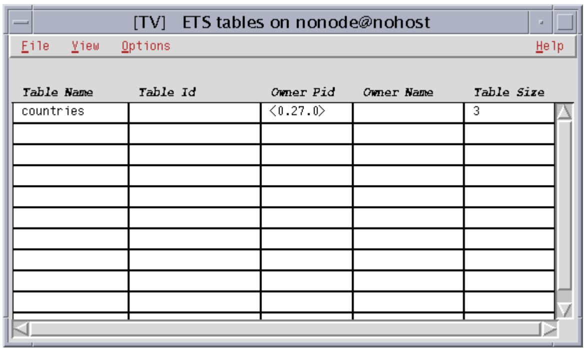
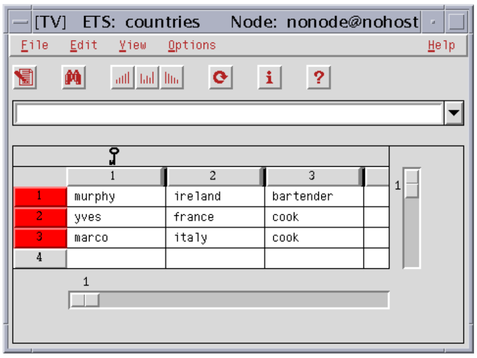


Table Name	Table Id	Owner Pid	Owner Name	Table Size
countries		<0.27.0>		3

- tv:start()

Table Visualizer



	1	2	3
1	murphy	ireland	bartender
2	yves	france	cook
3	marco	italy	cook
4			

- Click on the table to visualise it

Erlang Term Storage

- ETS Tables
- Handling Elements
- Searching and Traversing
- Match Specifications and Select
- Other Issues
- Table Visualizer