Leex and Yecc

a practical application to SQL, the query language to be beaten

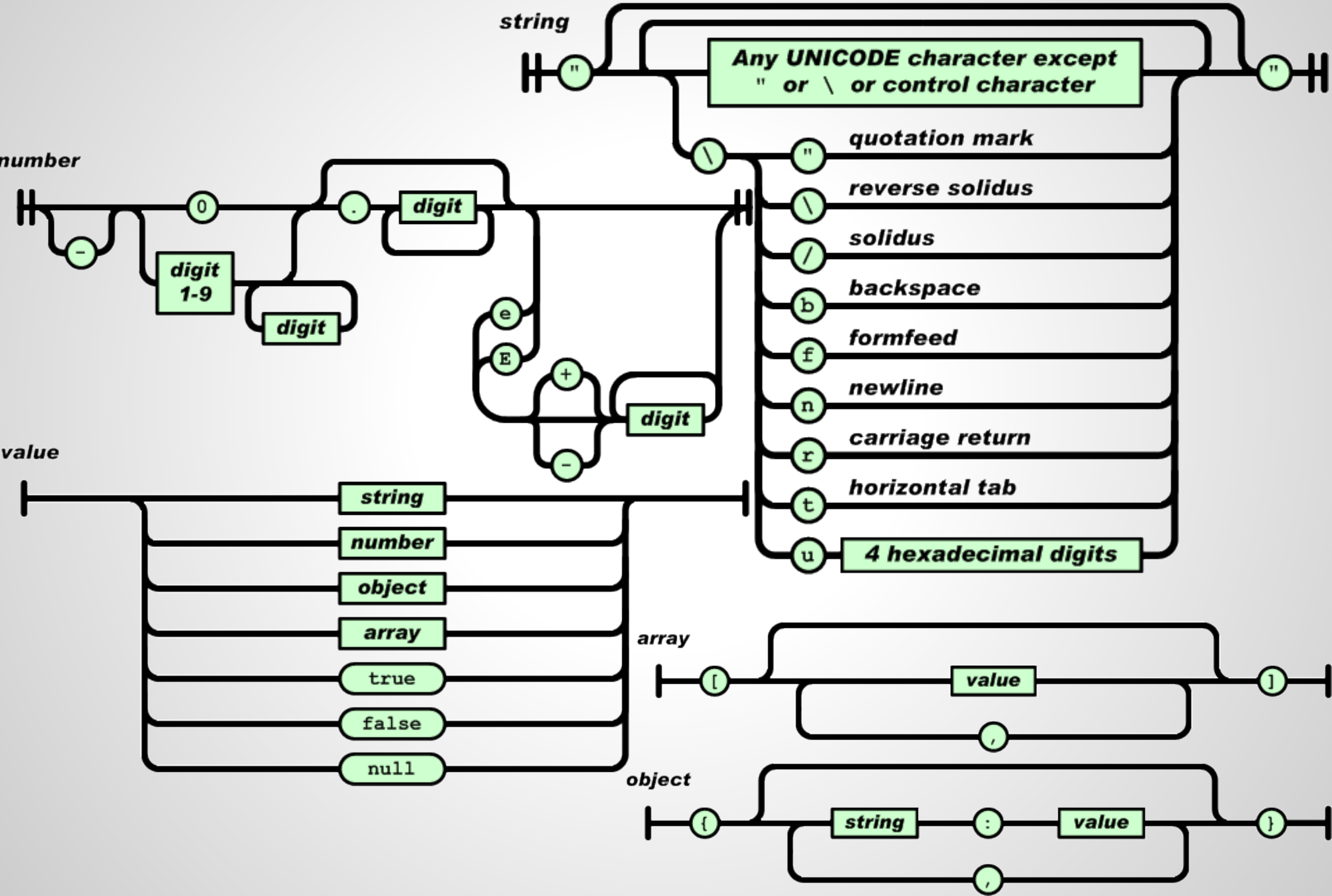
presenters (in order of appearance)
Bikram Chatterjee
Stefan Ochsenbein

Leex and Yecc are erlang implementations of Lex (Lexical Analyzer Generator or tokenizer) and Yacc (Yet Another Compiler Compiler or LALR grammar parser) respectively. The presentation will show by example how easy it is to write a tokenizer and a parser based on the LALR grammar. Its is assumed that the audience already has a basic understanding of terms like “grammar”, “LALR”, parsing etc. This presentation is structured in two parts:

The first part will explain how to write a lexer script (.xrl) and a matching parser grammar (.yrl). The generated erlang module sources will then be used to parse some sample data (JSON in this example).

The power of Yecc and Leex are demonstrated in the second part of the talk with a more practical and more complex use-case of SQL parsing. SQL is the most often and most successfully used query language on the planet. It may pay to look at it in depth before we can talk about extensions, adaptations or replacements for NoSql concepts.

- "How" and not "Why"
- Definitation of tokenizer and LALR (`.xrl` and `.yrl` files)
- Generating tokeniser and parser sources (`leex` and `yecc`)
- The rebar magic
- `yajc` example (<u>Y</u>et <u>A</u>nother <u>J</u>son <u>C</u>ompiler) https://github.com/c-bik/yajc
- A more complex and practical example - SQL
- *Stefan takes over from here*


string
"
Any UNICODE character except " or \ or control character
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits
"
number
-
0
digit 1-9
digit
.
digit
e
E
+
-
digit
value
string
number
object
array
true
false
null
array
[
value
,
]
object
{
string
:
value
,
}

```
% @file json_lex.xrl
% Copyleft
% @Author Bikram Chatterjee
% @Email razorpeak@gmail.com

Definitions.
D = [0-9]
S = (\+|\-)?
H = [a-zA-Z0-9]
Spl = (\\((u{H}{4})|([\"trf\bn\/])))

Rules.
([\s\t\r\n]+)                   : skip_token.
[\{\}\[\]\,\:]                  : {token, {list_to_atom(TokenChars),
TokenLine}}.
('true'|'false'|'null')         : {token, {list_to_atom(TokenChars),
TokenLine}}.
{S}{D}+                         : {token,{'NUMBER',TokenLine,list_to_integer
(TokenChars)}}.
{S}{D}+\.{D}+((E|e){S}{D}+)?    : {token,{'NUMBER',TokenLine,list_to_float
(TokenChars)}}.
"(([^\\\"])|{Spl})*"            : {token,{'STRING',TokenLine,strip_quotes
(TokenChars)}}.

Erlang code.
strip_quotes(StrChars) ->
    list_to_binary(string:substr(StrChars, 2, string:len(StrChars) - 2)).
```

```
Header "@file json_parse.yrl"
"%% Copyleft"
"%% @private"
"%% @Author Bikram Chatterjee"
"%% @Email razorpeak@gmail.com".

Nonterminals
 value object array value_list name_val_pair_list.

Terminals
 NUMBER STRING '{' '}' ',' ':' '[' ']' 'true' 'false'
'null'.

Rootsymbol value.

%Endsymbol '$end'. %(optional)

% operator precedence (optional)
% Right 100 '='.
% Nonassoc 200 '==' '=/='.
% Left 300 '+'.
% Left 400 '*'.
% Unary 500 '-'.
```

continued to next slide...

json_parser.yrl continued...

```
% grammer rules
value -> STRING                                                   : unwrap
('$1').
value -> NUMBER                                                   : unwrap
('$1').
value -> 'true'                                                   : 'true'.
value -> 'false'                                                  : 'false'.
value -> 'null'                                                   : 'null'.
value -> object                                                   : '$1'.
value -> array                                                    : '$1'.

array -> '[' value_list ']'                                       : '$2'.
object -> '{' name_val_pair_list '}'                              : '$2'.
name_val_pair_list -> STRING ':' value                            :
[{list_to_atom(unwrap_to_string('$1')), '$3'}].
name_val_pair_list -> STRING ':' value ',' name_val_pair_list     :
[{list_to_atom(unwrap_to_string('$1')), '$3'}|'$5'].

value_list -> '$empty'                                            : [].
value_list -> value                                               : ['$1'].
value_list -> value_list ',' value                                : '$1' ++
['$3'].

Erlang code.
unwrap({_,_,X}) -> X.
unwrap_to_string({_,_,X}) -> binary_to_list(X).
```

Compile Steps

1. Lexical Analyzer generation from .xrl
   - `leex:file(json_lex.xrl) %` *Generates json_lex.erl*
2. LALR-1 Parser generation from .yrl
   - `yecc:file(json_parse.yrl) -> {ok, "json_parse.erl"}`
3. Compile the generated lexer and parser modules (`json_lex.erl` and `json_parse.erl`)

Or

rebar magic

```
put .xrl and .yrl files in src folder and forget about it :)
```

Source distribution notes

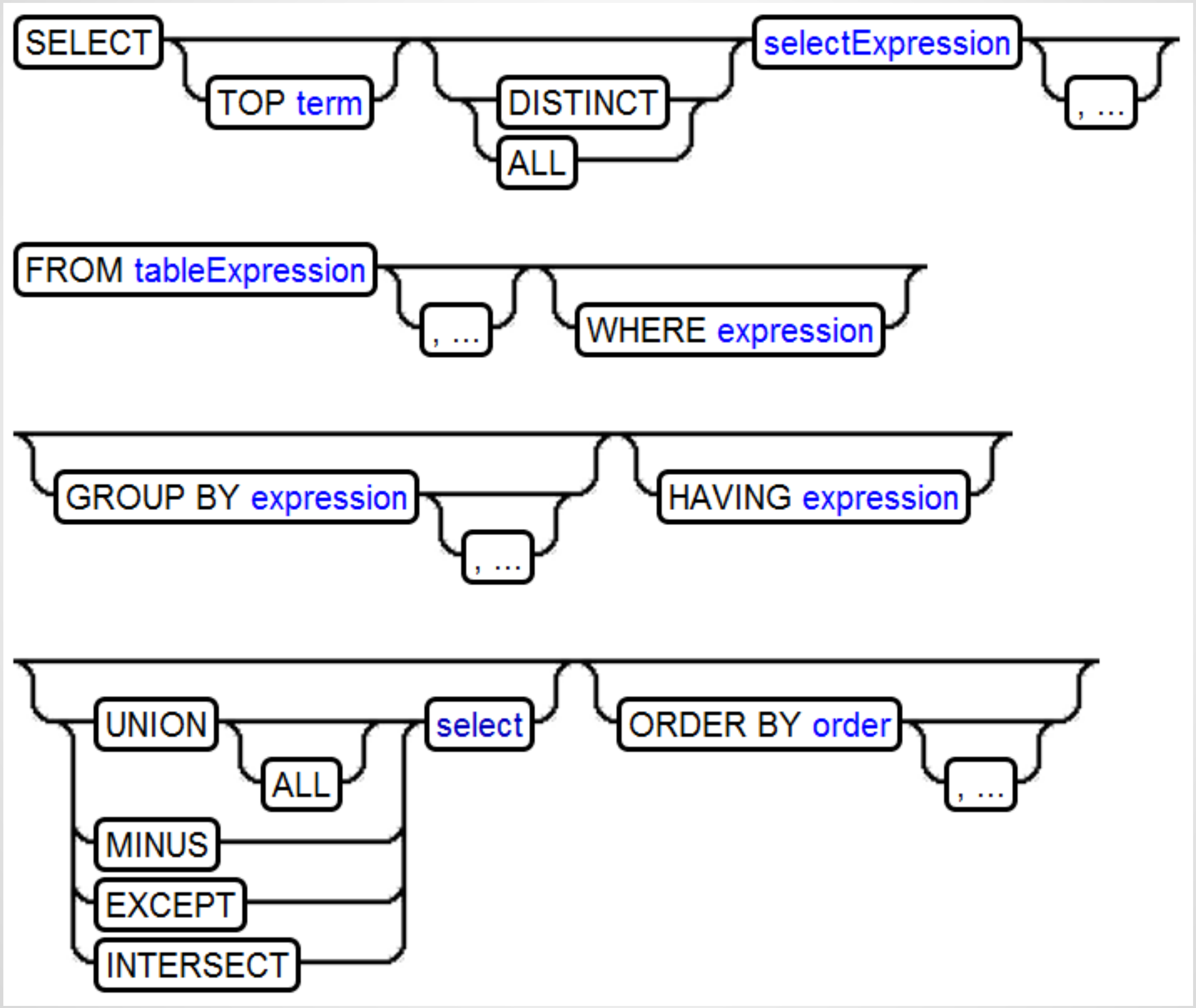
```
.gitignore lexer and parser module sourec filea or remove .xrl and .yrl
files
```

Using the lexer-parser

Input is JSON string of object

```
{
 a : "b",
 c : [10, -10]
}
```

```
5> {ok, Tokens, _} = json_lex:string( "{\"a\":\"b\", \"c\":[10,
-10]}").
{ok,[{'{',1},
     {'STRING',1,<<"a">>},
     {':',1},
     {'STRING',1,<<"b">>},
     {',',1},
     {'STRING',1,<<"c">>},
     {':',1},
     {'[',1},
     {'NUMBER',1,10},
     {',',1},
     {'NUMBER',1,-10},
     {']',1},
     {'}',1}],
    1}

6> json_parse:parse(Tokens).
{ok,[{a,<<"b">>},{c,[10,-10]}]}
```


SELECT
TOP term
DISTINCT
ALL
selectExpression
, ...
FROM tableExpression
, ...
WHERE expression
GROUP BY expression
, ...
HAVING expression
UNION
ALL
MINUS
EXCEPT
INTERSECT
select
ORDER BY order
, ...

K2 Informatics GmbH

.l file

```
%{
...
#define SV save_str(yytext)
#define TOK(name) { SV;return name; }
%}
%s SQL
%%

EXEC[ \t]+SQL  { BEGIN SQL; start_save(); }

<SQL>ALL       TOK(ALL)
<SQL>AND       TOK(AND)
...
<SQL>[A-Za-z][A-Za-z0-9_]*    TOK(NAME)
...


%%

void
yyerror(char *s)
{
printf("%d: %s at %s\n", lineno, s, yytext);
}
...
```

.xrl file

```
Definitions.

Rules.





...

(ALL|all) : {token, {'ALL', TokenLine}}.
(AND|and) : {token, {'AND', TokenLine}}.
...
[A-Za-z][A-Za-z0-9_]* : {token, {'NAME',
TokenLen, TokenChars}}.
...

Erlang code.
```

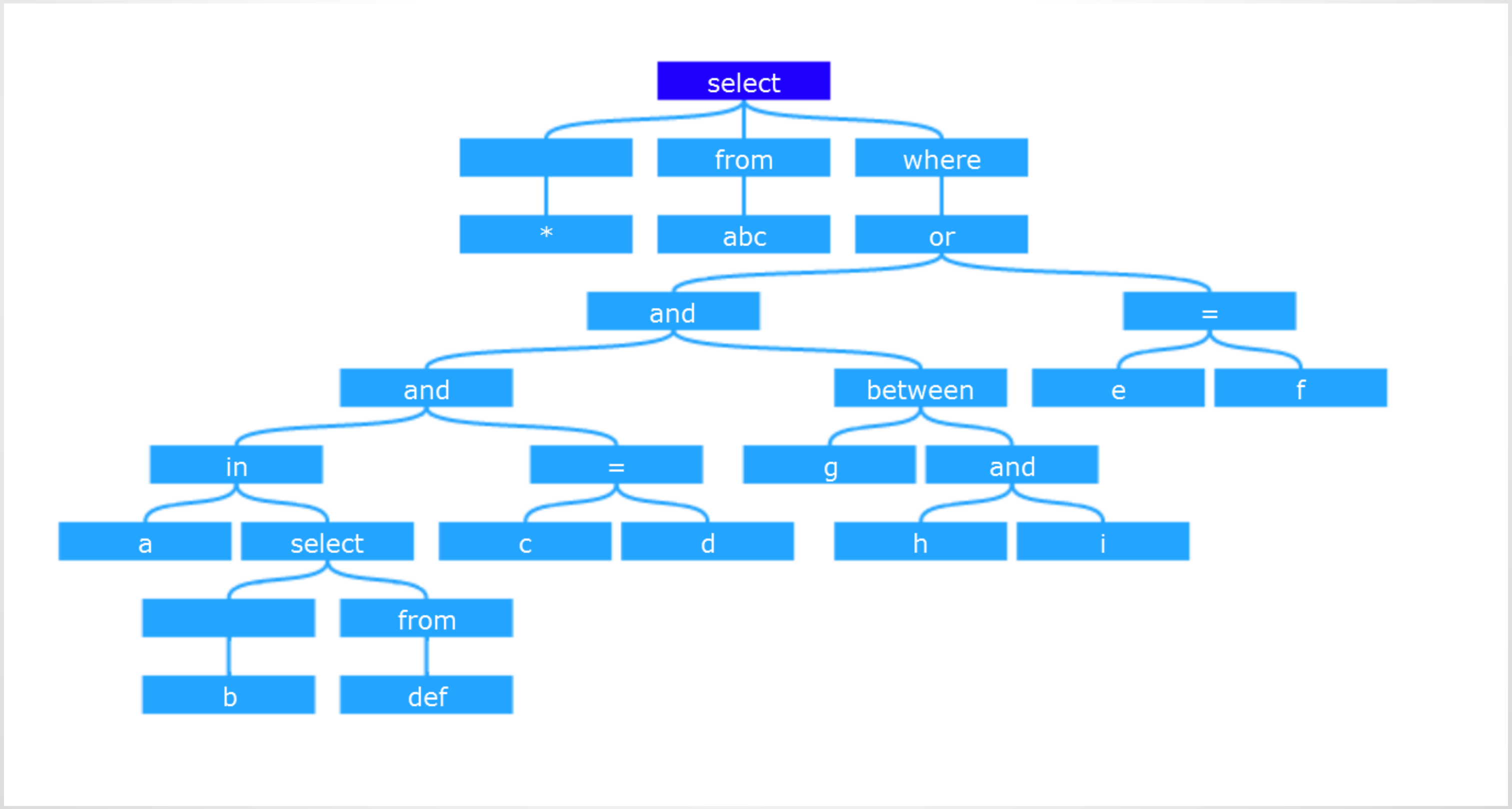
.y file

```
%union {
     int intval;
     ...
}
/* comments */


%token NAME

%left OR
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS

%token ALL AMMSC ANY AS ASC
%token CHARACTER CHECK CLOSE

%%

sql_list:
          sql ';'   { end_sql(); }
     |    sql_list sql ';' { end_sql(); }
     ;


%%
```

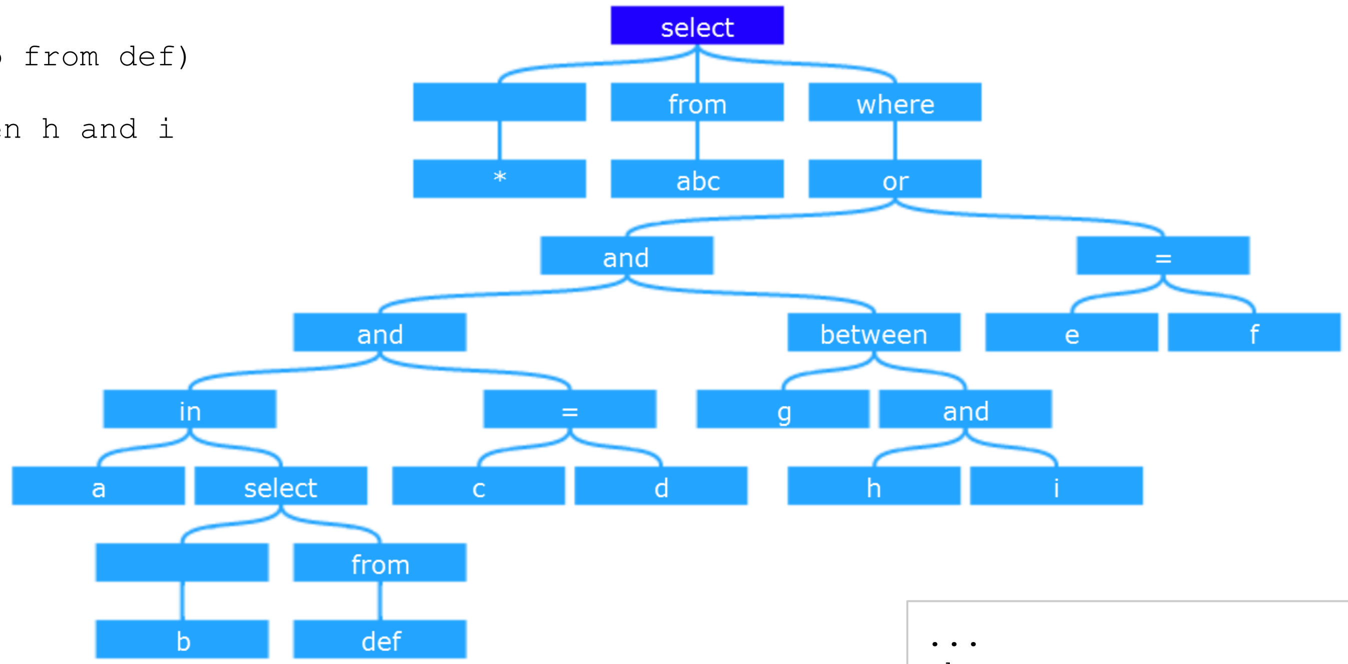
.yrl file

```
Header "%% Copyright (C) K2 Informatics GmbH"

% comments

Nonterminals sql_list
 sql.

Terminals NAME
 STRING.

Rootsymbol sql_list.

Left        100 'OR'.
Left        300 '+' '-'.
Left        400 '*' '/'.



sql_list -> sql ';'          : ['$1'].
sql_list -> sql_list sql ';' : '$1' ++
['$2'].



Erlang code.
unwrap({_,_,X}) -> X.
```

```
select * from  abc
where a in
   (select b from def)
and c=d
and g between h and i
or e=f
```



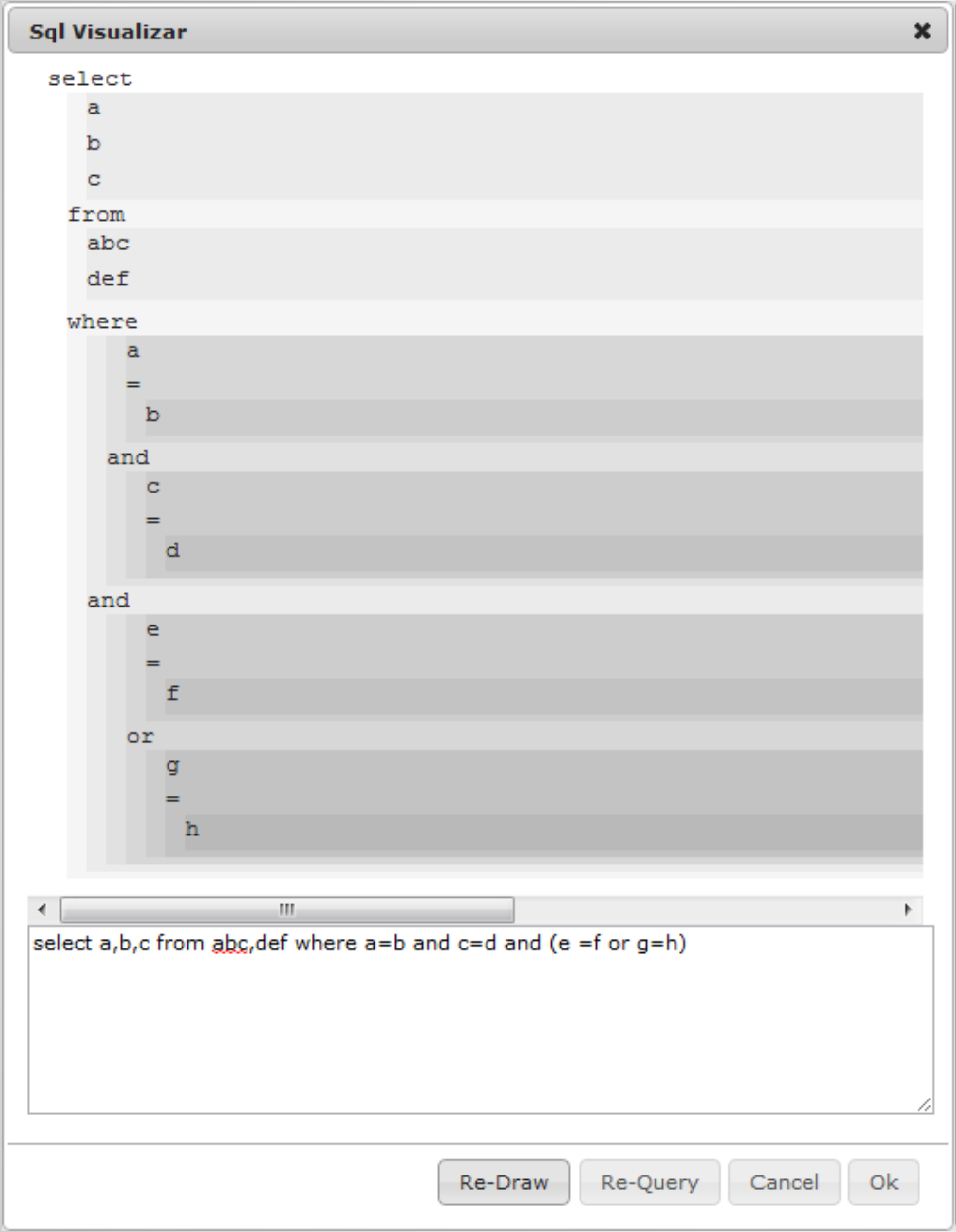
```
select * from  abc
where a in
    (select b from def)
and c=d
and g between h and i
or e=f
```



```
...
where
            a
            in(
                select
                    b
                from
                    def
            )
        and     c=d
        and
            g between h and i
    or     e=f
```

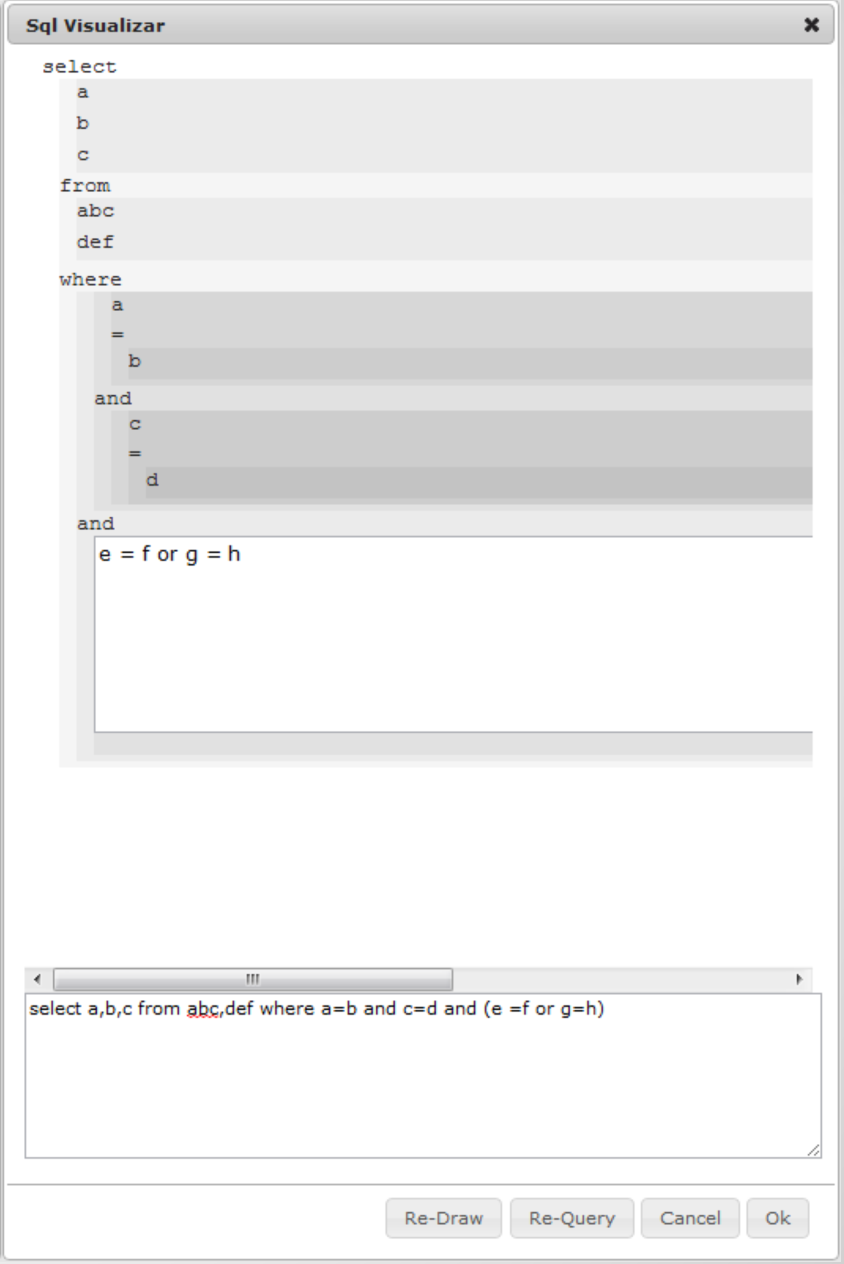
K² Informatics GmbH

Leex & Yecc 08.08.2012



```
select
  a
  b
  c
 from
  abc
  def
 where
    a
    =
     b
   and
     c
     =
      d
  and
     e
     =
      f
    or
      g
      =
       h
```

```
select a,b,c from abc,def where a=b and c=d and (e =f or g=h)
```

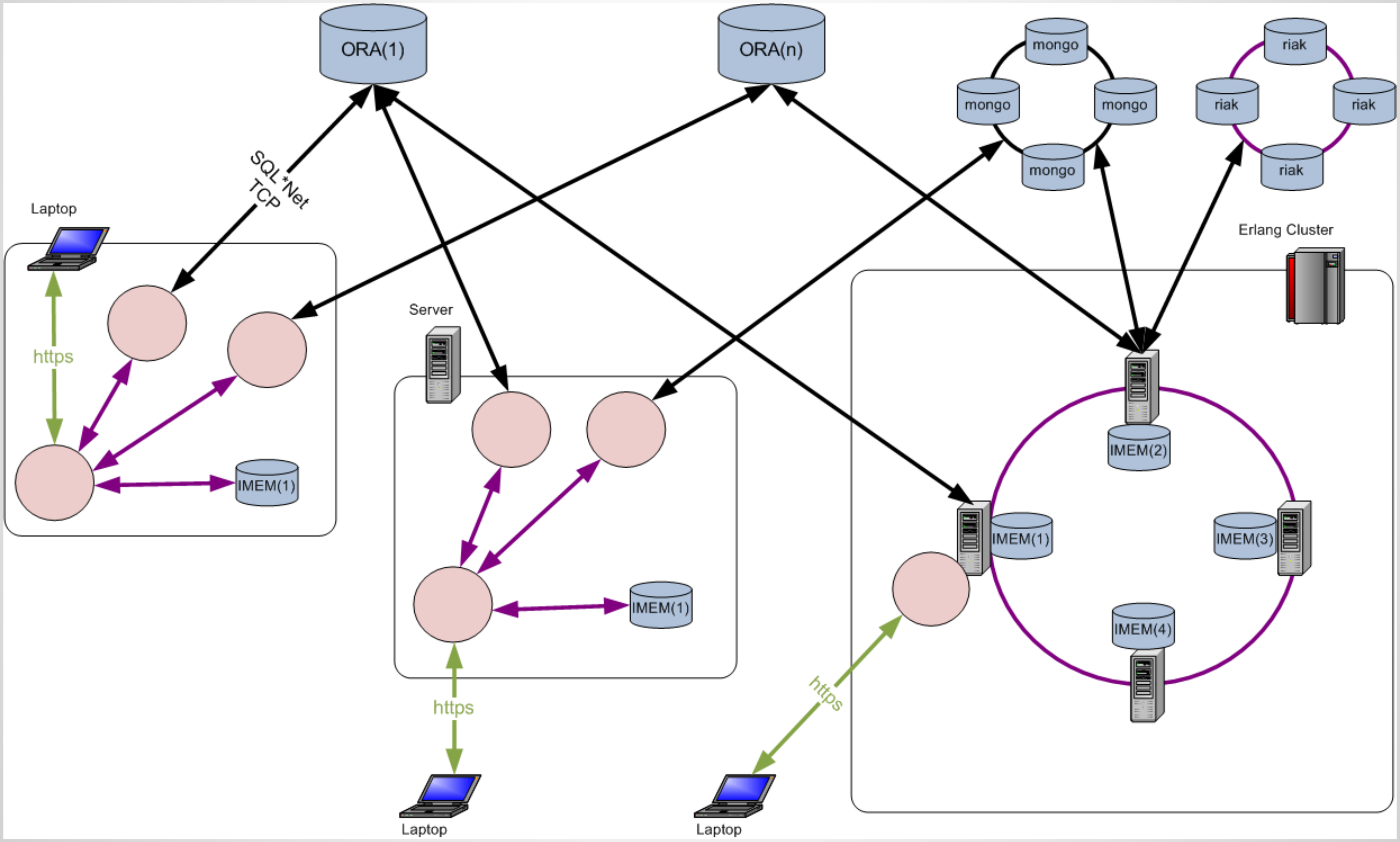


```
select
  a
  b
  c
 from
  abc
  def
 where
    a
    =
     b
   and
     c
     =
      d
  and
   e = f or g = h
```

```
select a,b,c from abc,def where a=b and c=d and (e =f or g=h)
```

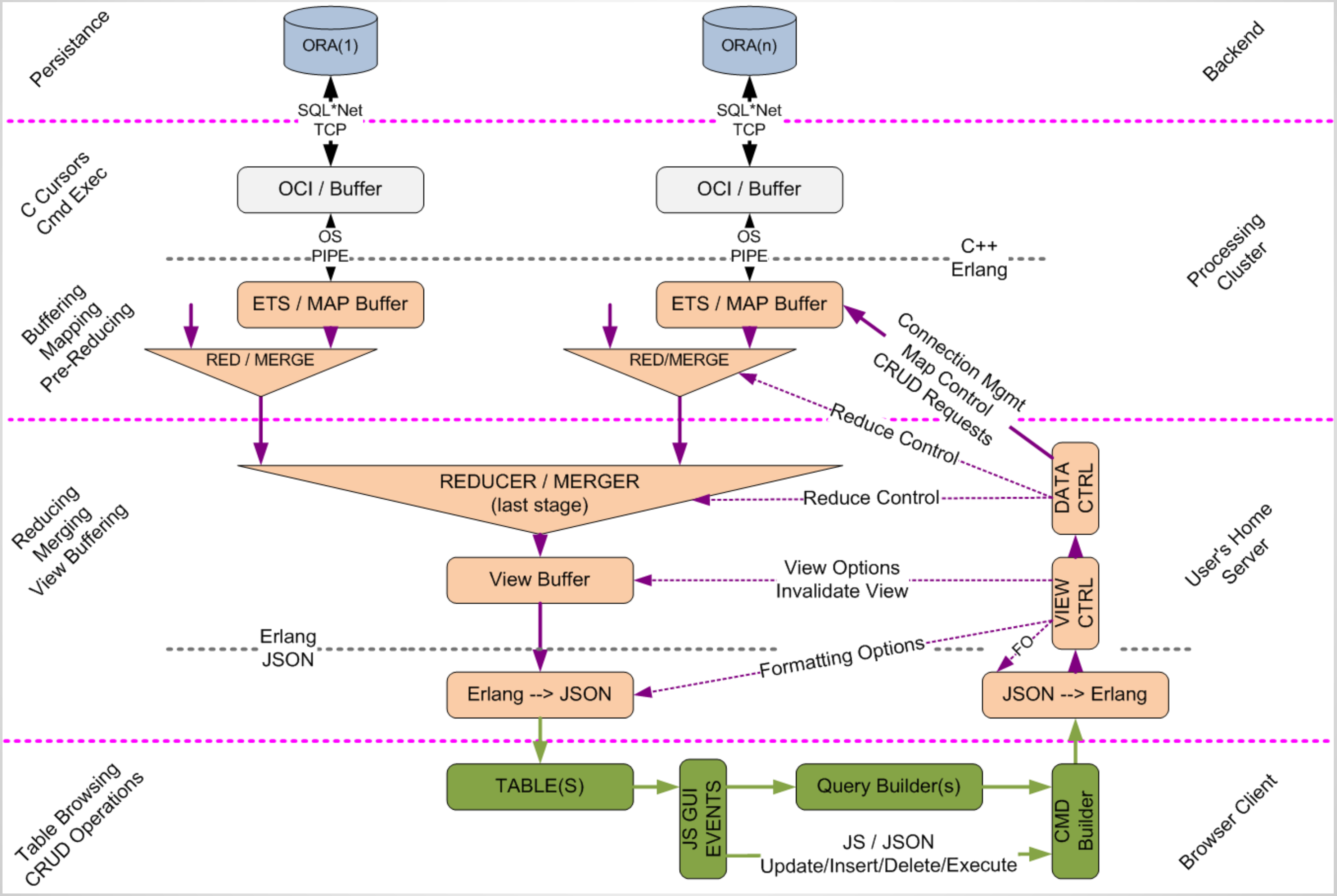
K2 Informatics GmbH

Leex & Yecc 08.08.2012



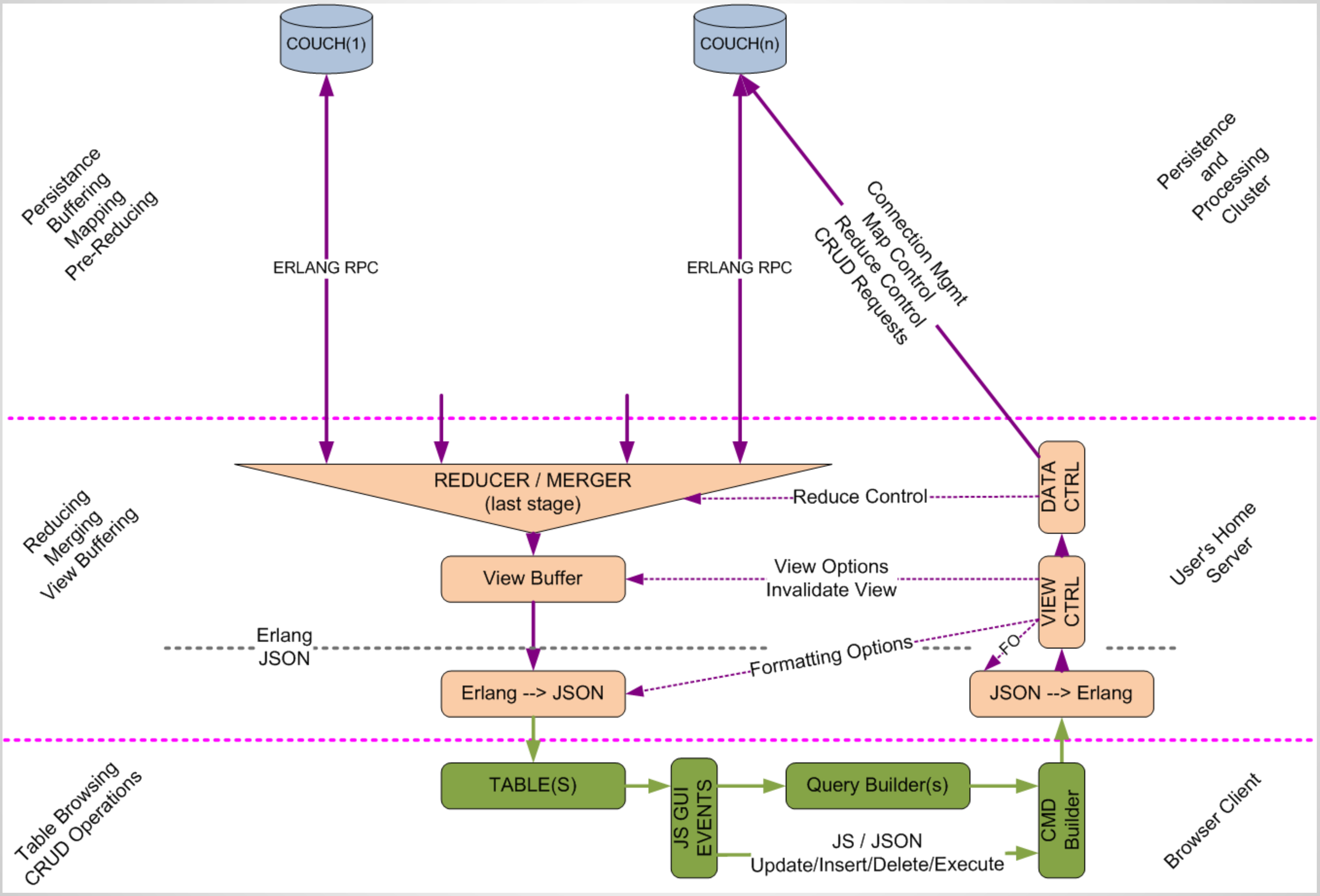
K² Informatics GmbH

Leex & Yecc 08.08.2012



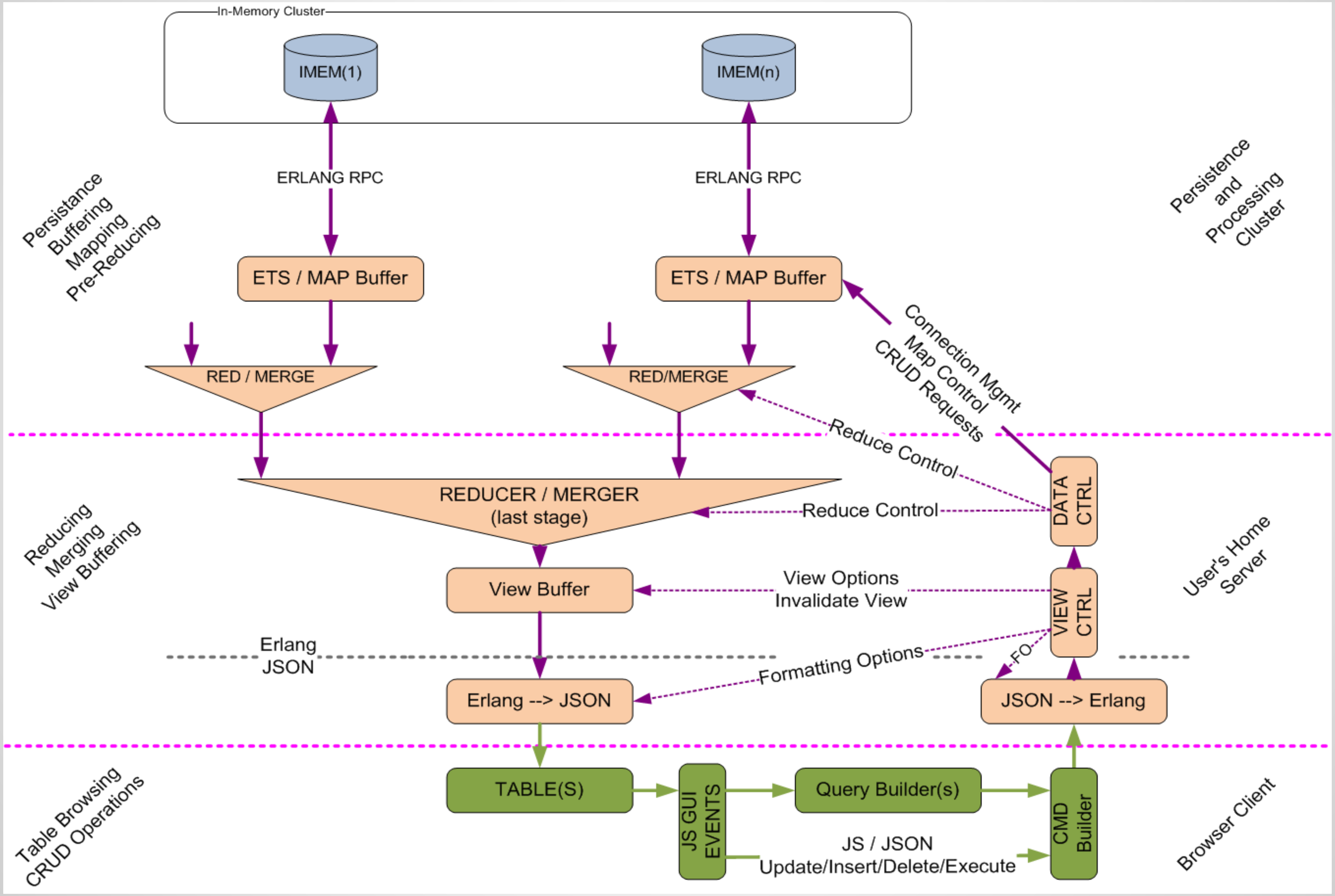
K² Informatics GmbH

Leex & Yecc 08.08.2012



K² Informatics GmbH

Leex & Yecc 08.08.2012



Thanks

References and related works

http://rustyklophaus.com/articles/20110208-LeexAndYecc.html
https://github.com/c-bik/yajc
http://www.json.org/fatfree.html
http://www.h2database.com/html/grammar.html
https://github.com/jchris/erlang-json-eep-parser