SQL语法约束

SQL标识符(Identifiers)

SQL标识符需要满足以下四个条件：

- 首字母是大写或者小写的英文字母。
- 只能包含字母、数字和下划线(_)。
- 不能使用预留字。
- 标识符只能使用ASCII字符集。

举个非法SQL的例子，如下使用预留关键字的SQL是无法在数据库中执行的。

```
-- table 是关键字，不可作为表名出现在SQL中。
CREATE TABLE TABLE (column1 character);
-- column是列的关键字，不可作为列名出现在SQL中
CREATE TABLE X (COLUMN varchar(10));
```

UPDATE

- SQL92语法

  ```
  UPDATE table_name
  SET assignment [, assignment ] , ... [ WHERE search_condition ] ;
  ```

  对update语句中assignment的语法做一下详细的描述。

  在92标准中assignment语法是这样定义的

  ```
  column = { expr | NULL }
  |( column [, column ], ...) = ( expr [, expr ]) | ( column [, column ] ,
  ... ) = ( query_expression )
  ```

  - `column = { expr | NULL }` 是比较常规，也是大多数数据库支持的语法。
  - `( column [, column ], ...) = ( expr [, expr ])` 这种语法通过具体的SQL来看会更清晰，`update t_order set (name, gender) = ('whatever', 1) where order_id = 13134;` 。实测发现，主流的oracle和mysql目前都不支持这种写法。
  - 第三种update赋值写法暂且不描述了，DDAL暂不支持。

- DDAL语法

  目前只支持第一种assignment的写法。update语句存在全片更新的场景，如果未指定分片键，SQL会被路由到所有分片执行，这在ddal中是属于正常的流程。但绝大多数场景下还是推荐update语句携带分片字段。

DELETE

- SQL92语法

```
DELETE FROM [ owner_name.] { table_name | view_name } [ WHERE
search_condition ] ;
```

where条件和owner_name是可选项。如果where条件未指定或者未指定分片键，则删除所有分片上指定表的所有行。

- DDAL语法

  ddal完全支持SQL92的语法。

- 示例

```
DELETE FROM customer WHERE customer_name = 'RALPH' ;
```

INSERT

- SQL92语法

```
INSERT INTO [ owner_name.]{table_name|view_name }
[ ( column_name [, column_name ] , ... ) ]
{ VALUES ( value [, value ] , ... ) | query_expression } ;
```

- DDAL语法

```
INSERT INTO [ owner_name.]{table_name|view_name }
[ ( column_name [, column_name ] , ... ) ]
{ VALUES ( value [, value ] , ... )} ;
```

ddal不支持values后跟查询语句，并且单条语句有且只能有一组value。

- 示例

```
INSERT INTO customer (cust_no, name, street, city, state) VALUES
(1001, 'RALPH', '#10 Columbia Street', 'New York', 'NY') ;
```

注意：

  - 非全局表的insert语句必须指定分片键，可通过values或者hint指定分片键。
  - 全局表的insert语句如果未指定分片键则会路由至所有分片。

以下两种SQL是不支持的

```
-- 一条语句插入多行
INSERT INTO customer (cust_no, name, street, city, state) VALUES
(1001, 'RALPH', '#10 Columbia Street', 'New York', 'NY'), (1002,
'RALPH', '#10 Columbia Street', 'New York', 'NY');
-- values是查询语句
INSERT INTO neworders (order_no, product, qty)
SELECT order_no, product, qty FROM orders WHERE order_date = SYSDATE ;
```

SELECT

- SQL92语法

```
SELECT column_list FROM table_list
[ WHERE search_condition ]
[ GROUP BY grouping_condition ]
[ HAVING search_condition ]
[ORDER BY ordering_condition ]
[FOR UPDATE update_condition ] ;
```

- DDAL语法

```
SELECT column_list FROM table_list
[ WHERE search_condition ]
[ GROUP BY grouping_condition ]
[ORDER BY ordering_condition ]
```

select语句暂不支持having和for update。

关于column_list，table_list，grouping_condition以及ordering_condition的的详细说明，请看考对应小结。

COLUMN_LIST

- SQL92语法

```
[ ALL | DISTINCT ]
{ * | { table_name. | alias.}* [ , { table_name.| alias.} * ] ...
|expr [[AS ][' ]column_title [' ]]
[, expr [[AS ][' ]column_title [' ]]]...
| [ table_name.| alias.]column_name , ... ] }
```

  - [ALL | DISTINCT]

    结果集是否去重的标记。如果不指定这个关键字，默认是ALL，也就是不去重。

  - * | { table_name. | alias. } *

    查询from从句指定的表的所有列。

  - * expr [[AS][']column_title [']]

`expr` 支持数学表达式以及聚合函数。表达式计算的结果列可以指定别名 `column_titile` 。在表达式和别名中间可以使用 `as` 关键字，并且别名上可以加单引号 `'`

表达式的支持列表请参考表达式小节。

- [table_name. | alias.]column_name , ...]

  指定列作为结果集的输出。

- DDAL语法

  DDAL兼容SQL92的select语法。

- 示例

  如下两个语句将返回customer表的所有列数据

```
SELECT * FROM customer;
SELECT customer.* FROM customer;
```

  多表联查时，表名+列名和表名+星号可混用

```
SELECT Customer.CustNum, Customer.Name, Invoice.* FROM Customer, Invoice
;
```

  列别名的用法

```
select FirstName as 'First Name', LastName as 'Last Name', state as 'New
England State' from employee where state = 'NH' or state = 'ME' or state
= 'MA' or state = 'VT' or state = 'CT' or state = 'RI';
```

  当from子句中多张表存在相同字段时，select子句必须指定列名

```
select
Customer.custnum,
Customer.city as "Customer City",
Customer.State as 'Customer State',
Billto.City as "Bill City",
Billto.State as 'Bill State'
from Customer, Billto
where Customer.City = 'Clinton';
```

  customer和billto表都存在city和state字段，所以select语句的column_list需要指定列名。

ORDER BY

- SQL92语法

```
ORDER BY {expr |posn }[ASC |DESC ]
[, {expr |posn }[ASC |DESC ], ...]
```

expr表示column_list中的列名，posn是column_list中列队迎的下标，从1开始。

asc是升序，desc表示降序。

注意点：

- 如果select语句中使用order by，应放在所有从句的最后。
- order by具有排序传递特性。即如果当前排序字段的值相同，那么使用与其紧邻的后一个字段排序，以此类推。
- 如果查询语句使用union语句，order by只能使用position排序。

- DDAL语法

  ddal兼容SQL92定义的order by语法。但是ddal暂不支持union，所以示例中的第二条语句仅供参考。

  是否支持posn还待验证。

- 示例

```
-- Produce a list of customers sorted by last_name.
SELECT last_name, street, city, state, zip
     FROM customer
     ORDER BY last_name ;
-- Produce a merged list of customers and suppliers sorted by last_name.
SELECT last_name, street, state, zip
FROM customer
UNION
SELECT last_name, street, state, zip FROM supplier
ORDER BY 1 ;
```

GROUP BY

- SQL92语法

```
GROUP BY [ table_name.]column_name ...
```

group by指定的列名必须在select的列表中。相反，select的列表中的列必须在group by从句中或者是聚合函数的一部分。

举个例子：

```
select cname, tno from course group by tno;
```

上述语句在MySQL中可以执行，但是并不符合SQL92标准，所以在标准SQL92的数据库上无法执行。

聚合函数（Aggregation Function）

这一章节列举DDAL目前支持的函数，不在此列表中的函数表示暂不支持。

- AVG 平均值。

```
AVG ( { [ ALL ] expression } | { DISTINCT column_ref } )
```

从语法上可以看出avg还可以与distinct联合使用。

```
-- 保留重复行计算平均值
select avg(order_id) from t_order;

-- 剔除重复行再计算平均值
select avg(distinct(order_id)) from t_order;
```

- COUNT 计算结果集行数

```
COUNT ( { [ ALL ] expression } | { DISTINCT column_ref } | * )
```

如果count函数使用`*`，数据库会先将结果集分组再计算行数。如果count函数的入参不是`*`，而是某一列名，计算行数时数据库会忽略该列的null值。

```
-- 结果集分组后再Count
select count(*) from t_order;
-- 忽略create_time == null的记录，再count
select count(create_time) from t_order;
```

- SUM 求和

```
SUM ( { [ALL] expression } | { DISTINCT column_ref } )
```

SQL92标准中提到sum可用于任何类型的字段。但是实际使用过程中发现，数据库并没有支持这种说法，更多的还是只支持数字类型的字段。

- MIN MAX 求最大最小结果

```
-- min
MIN ( { [ ALL ] expression } | { DISTINCT column_ref } )

-- max
MAX ( { [ ALL ] expression } | { DISTINCT column_ref } )
```

示例语句

```
SELECT MIN (salary) FROM employee WHERE deptno = 20 ;

SELECT order_date, product, MAX (qty) FROM orders GROUP BY order_date,
product ;
```

分页

SQL92标准中并没有关于分页的描述，所以不同的数据库厂商都有各自的实现。比如oracle使用rownum，mysql使用limit。

作为数据库中间件，为了屏蔽数据库对分页的不同实现对应用的影响，DDAL制定了自己的分页写法。

分页线索提供两个参数：offset和count。

- offset 表示第一条数据相对表头的偏移量。offset=0时，返回结果从第一条数据开始查询。offset=1时从第二条数据开始查询。
- count 表示返回结果的条数。

对于分页场景，按照注释的写法对原SQL语句进行改写。

假设原语句是：

```
select id, name from user
```

如想查询原结果的前10条，SQL语句需改写为：

```
/*!raptor page(offset=0,count=10)*/select id, name from user
```

强烈建议使用者采用这种隔离数据库实现的方式进行分页