Mysql基础知识

TIPS

- SQL语句都应该以分号结尾";"。
- 日期通过Date进行存储。

常见关键字

使用数据库

- SHOW

```
查询数据库操作：
    SHOW DATABASES;
    SHOW TABLES;
    SHOW COLUMNS FROM TAB_NAME;
    SHOW CREATE DATABASE ;
    SHOW CREATE TABLE ;

查询数据库的状态：
    SHOW STATUS ;
    SHOW ERRORS ;
    SHOW WARNINGS ;
    SHOW GRANTS ;
```

查询

- SELECT
    - 通配符（*）：查询所有的列。
    - DISTINCT：对查询结果去重操作。使用必须放在所有列的前面，作用于所有列。
    - LIMIT OFFSET，SIZE：限制查询结果的容量，返回第(OFFSET)至(OFFSET+SIZE)条。
    - 完全限定名（TAB_NAME.COL_NAME）：使用联合查询时候，两张表可能存在一样的字段名。

```
查询字段信息：
    SELECT COL_NAME FROM TAB_NAME;
    SELECT * FROM TAB_NAME; // * 是通配符，返回数据包括所有字段
    SELECT DISTINCT COL_NAME1, COL_NAME2 FROM TAB_NAME; // DISTINCT是去重操
作(作用域必须是所有查询列)
    SELECT COL_NAME FROM TAB_NAME LIMIT OFFSET, SIZE; // 返回特定OFFSET区间内
的数据
    SELECT TAB_NAME.COL_NAME FROM TAB_NAME; //通过字段的完全限定名查询
```

子句

SQL 语句通常由多个字句构成。有些子句是必须的，有些不是必须的。常见的又：FROM 子句、WHERE 子句、ORDER BY 子句。

- ORDER BY：有两种选项，ASC|DESC（默认是ASC）。按照多个字段进行排序时，必须为每个字段都制定特定的排序顺序。

```
    SELECT * FROM TAB_NAME ORDER BY COL_NAME1; //查询结果按照COL_NAME1升序进行排列（缺省即默认情况，ASC排列）。
    SELECT * FROM TAB_NAME ORDER BY COL_NAME ASC; //查询结果按照COL_NAME1升序进行排列。
    SELECT * FROM TAB_NAME ORDER BY COL_NAME1 DESC; //查询结果按照COL_NAME1降序排列。
    SELECT * FROM TAB_NAME ORDER BY COL_NAME1 DESC,COL_NAME2 ASC; // 查询结果先按照COL_NAME1 进行降序排列，然后通过COL_NAME2 进行升序排列。
    SELECT * FROM TAB_NAME ORDER BY COL_NAME1 LIMIT OFFSET, SIZE; // 通过 LIMIT 对查询结果进行限制的时候，必需将 LIMIT 放在 ORDER BY 之后。
    注：
        ORDER BY 对查询结果大小写（A，a）的区分在与数据库的配置。mysql的默认排序方式：字典排序顺序，不能实现区分。

        LIMIT 必须放在 ORDER BY 之后。
```

- WHERE 子句：指定搜索条件，过滤数据（相较于应用层的筛选，引擎层的筛选可以减少传输带宽、减少应用层二次计算）。

操作符	含义
=	等于
<	小于
`>`	大于
<>	不等于
!=	不等于
<=	小于等于
>=	大于等于

```
    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 = 'BOB';
    // 符合 COL_NAME1 等于BOB条件的数据

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME2 < 18;
    // 符合 COL_NAME2 小于18 条件的数据

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME2 > 18;
    // 符合 COL_NAME2 大于18 条件的数据
```

```
    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME2 != 18;
    // 符合 COL_NAME2 不等于18 条件的数据

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME2 <> 18;
    // 同上
```

操作符	含义
IS NULL	判断空值
BETWEEN VALUE1 AND VALUE2	位于 VALUE1 和 VALUE2 之间

```
    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME2 BETWEEN 2 AND 18;
    // 符合 COL_NAME2 在[2, 18] 条件的数据

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME2 IS NULL;
    //符合 COL_NAME2 为 NULL 的条件

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 ="BOB" ORDER BY
TAB_NAME.COL_NAME1 DESC;
    //符合 COL_NAME1 等于 BOB 条件的数据，并且按照 COL_NAME1 降序排列

    注：
        ORDER 必须放在 WHERE 条件语句的之后。
        IS NULL： 必须是COL_NAME1 字段为NULL；查询结果为空并不代表 IS NULL。
```

操作符	含义
AND	联结两个条件，表示两个条件都成立
OR	联结两个条件，表示两个条件成立一个即可
IN	指定条件范围
NOT	否定后续所有的条件

```
    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 = 'BOB' AND
TAB_NAME.COL_NAME2 < 18;
    //符合 COL_NAME1 等于 BOB 、 COL_NAME2 小于 18 两个条件的数据

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 = 'BOB' AND
TAB_NAME.COL_NAME2 < 18;
    //符合 COL_NAME1 等于 BOB 或者 COL_NAME2 小于 18 其中之一的数据

    SELECT * FROM TAB_NAME WHERE (TAB_NAME.COL_NAME1 = 'BOB' OR
TAB_NAME.COL_NAME2 < 18) AND TAB_NAME.COL_NAME2 > 2;
```

```
    //符合 COL_NAME1 等于 BOB 且 COL_NAME2 小于 18 条件；或者 COL_NAME > 2 条件，满足之一即可。

    SELECT * FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 = 'BOB' OR TAB_NAME.COL_NAME2 < 18 AND TAB_NAME.COL_NAME2 > 2;
    //符合 COL_NAME1 等于 BOB 或者 COL_NAME2 在[2, 18]区间的数据。『AND 和 OR 两者有不同的优先级』
    //上下两句的语意并不相同

    SELECT * FROM TAB_NAME WHERE COL_NAME2 IN (2, 3, 18);
    //符合 COL_NAME2 在set(2, 3, 18) 中的数据。



    注：
        AND 和 OR 两者的优先级不一致，使用时候 AND > OR，避免系统发生错误解析，需要添加括号。
        IN 较 OR 有更高的执行效率、IN 可以包含 SELECT 子句、可读性更好、计算次序更好管理。
```

- LIKE操作符号
  - 通配符：用来匹配值的一部分的特殊符号
  - 搜索模式：由字面值、通配符或者两者组合构成的搜索条件

操作符	含义
%	代表零个或者多个字符，但是不能匹配NULL
_	代表一个字符

```
    SELECT TAB_NAME.COL_NAME1 FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 LIKE '% BOB ';
    SELECT TAB_NAME.COK_NAME1 FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 LIKE 'BOB_';

    注：
        使用通配符必须用LIKE关键字。
        不能过度使用通配符，会造成性能下降
        通配符放在最开始会造成性能下降最严重
```

- 正则表达式

操作符	含义
.	匹配一个字符
|	或者
[]	匹配其中一个字符

操作符	含义
-	匹配某个范围的字符

```
    SELECT TAB_NAME.COL_NAME1 FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 REGEXP
'1000';
    // 查询 COL_NAME1 字段包含 1000 的数据

    SELECT TAB_NAME.COL_NAME1 FROM TAB_NAME WHERE TAB_NAME.COL_NAME1 REGEXP
'^1000$'
    // 查询 COL_NAME1 字段为 1000 的数据

    SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP '.000';
    // 查询 COL_NAME1 字段以 000 结尾的四位字符

    SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP 'A|B';
    // 查询 COL_NAME1 字段 为 A 或者 B 的情况

    SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP '[1|2|3]'
    // 查询 COL_NAME1 为 1 或者2 或者3 的情况

    SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP '[A-Z]';
    // 查询 COL_NAME1 在范围 A至Z。

    注：
        正则表达式如果没有定位符，则匹配字段中的内容
        LIKE则是匹配整个字段。
```

转义符号：可以实现通配符等转义、元符号的转义

操作符	含义
\\	转义字符(-、+、[、])
\\f	换页符
\\n	换行
\\r	回车
\\t	制表
\\v	纵向制表

```
    SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP '\\+';
    // 查询COL_NAME1字段中包含+符号的数据。
```

匹配多个实例：

操作符	含义

操作符	含义
*	0个或者多个匹配
?	0个或者1个匹配
+	1个或者多个匹配
{n}	n个匹配
{n, }	至少n个匹配
{n, m}	n到m个匹配

```
SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP '\\([0-9] sticks?\\)'
```

定位符：

操作符	含义
`^	字符开始
$	字符结束

```
SELECT * FROM TAB_NAME WHERE COL_NAME1 REGEXP '^BOB$';
SELECT * FROM TAB_NAME WHERE COL_NAME1 LIKE 'BOB';
SELECT * FROM TAB_NAME WHERE COL_NAME1 = 'BOB';

注：
    正则表达式如果没有定位符，则匹配字段中的内容
    LIKE则是匹配整个字段。
```

计算字段

对查询的返回的数据进行格式化

- 拼接字段

Concat(file1, file2, ……, fileN)，fileN可以是常量。

- 去空格

Trim(COL_NAME)：删除字符串前后的空格 RTrim(COL_NAME)：删除字符串右边的空格 LTrim(COL_NAME)：删除字符串左边的空格

- 四项运算

操作符	含义
+	COL_NAME2 + COL_NAME3

操作符	含义
-	COL_NAME2 - COL_NAME3
*	COL_NAME2 * COL_NAME3
/	COL_NAME2 / COL_NAME3

- 别名（Alias） AS：为字段取别名（字段可以是列表的原始的列，也可以是拼接字段等）

```
    SELECT Concat('COL_NAME1:', RTrim(COL_NAME1), 'COL_NAME2', COL_NAME2)
FROM TAB_NAME;

    SELECT Concat(Trim(COL_NAME1), '(', COL_NAME2, ')') AS concat_col FROM
TAB_NAME ORDER BY COL_NAME2;

    SELECT COL_NAME2 * COL_NAME3 AS COL_NEW FROM TAB_NAME ORDER BY
COL_NAME2;
```

函数

函数的可移植性不佳，每个平台都自己的函数。以下内容是MYSQL的函数。

- 字符串的处理函数

函数名	含义
LTrim()	删除字符左边的空格，并返回处理结果
RTrim()	删除字符右边的空格，并返回处理结果
Trim()	删除字符两边的空格，并返回处理结果
Upper()	所有字符都转换为大写，并返回处理结果
Lower()	所有字符都转换成小写，并返回处理结果
Left()	返回字符串最左边的字符
Right()	返回字符串最右边的字符
Length()	返回字符串的长度
Locate()	返回串的一个子串
Soundex()	返回串的SOUNDEX的值
SubString()	返回子串的字符

```
    SELECT * FROM TAB_NAME WHERE Soundex(COL_NAME) = Y.Lee;

    注：
        SOUNDEX是将文本串转换成语音表示的字母数字模式的算法。如 Soundex(Y.Lie) 转化
```

```
后可以匹配 Y.Lee
```

- 数字的处理函数

函数名	含义
Pi()	返回圆周率
Rand()	返回一个随机数
Sin()	返回正弦值
Cos()	返回余弦值
Tan()	返回正切值
Abs()	返回一个绝对值
Exp()	返回指数值
Mod()	返回求模值
Sqrt()	返回平方根

- 日期的处理函数

在处理日期的数据时候，以日期的数据存储更能实现有效的索引和查询。

聚合

对数据进行聚合查询

函数	含义	说明
AVG()	返回平均值	不统计NULL
COUNT()	返回条目数量	COUNT()不统计NULL，COUNT(*)统计NULL
MAX()	返回最大值	不统计NULL
MIN()	返回最小值	不统计NULL
SUM()	返回和	不统计NULL

```
    SELECT COUNT(*) AS sample_num, MAX(COL_NAME1) AS col_max,
MIN(COL_NAME2) AS col_min FROM TAB_NAME;

    SELECT COUNT(DISTINCT(COL_NAME2)) AS col_count FROM TAB_NAME;

    注：
        DISTINCT 不能使用在COUNT(*) 中。
```

数据分组

GROUP BY ..... HAVING：对查询数据进行分组操作，然后通过HAVING对分组进行条件筛选。

GROUP BY 相关规定：

- 可以实现通过嵌套分组实现数据的精细化展示
- 嵌套分组在最后一个字段进行汇总；每个字段都会进行计算，展示在最后一个字段。
- NULL 是被单独分为一组
- GROUP BY 必须放在 WHERE 子句之后。
- SELECT 中出现的字段都必须在 GROUP BY 中进行展示，聚合字段除外。（！！！）

```
    SELECT * FROM TAB_NAME GROUP BY COL_NAME1;

    SELECT COL_NAME1, COUNT(COL_NAME2) FROM TAB_NAME GROUP BY COL_NAME1;

    SELECT COL_NAME1, COUNT(COL_NAME1) FROM TAB_NAME GROUP BY COL_NAME1
WITH ROLLUP;

    SELECT COL_NAME1, COUNT(COL_NAME1) FROM TAB_NAME GROUP BY COL_NAME1
HAVING COUNT(*) >= 10;

    SELECT COL_NAME1, SUM(COL_NAME2 * COL_NAME3) AS col_new FROM TAB_NAME
GROUP BY COL_NAME1 HAVING COL_NAME > 10 ORDER BY col_new;

    注：
        WHERE：对每一条数据进行筛选，处理对象是每条数据。
        HAVING：对分组内的数据进行筛选，处理对象是每个分组的数据。
        WITH ROLLUP：可以在分组数据上再次进行统计。
```

子查询

- 进行查询过滤
- 嵌套子查询过多会造成性能下降
- 子查询从内向外处理
- 通常和 in 进行嵌套使用

```
    SELECT * FROM TAB_NAME1 WHERE COL_NAME1 IN (
        SELECT COL_NAME5 FROM TAB_NAME2
    );

    SELECT COL_NAME1, COL_NAME2, (
        SELECT COUNT(TAB_NAME2.COL_NAME1) FROM TAB_NAME2 WHERE
TAB_NAME.COL_NAME1 == TAB_NAME2.COL_NAME1
    ) AS order FROM TAB_NAME ORDER BY TAB_NAME.COL_NAME1;
```