

1-SQL

分页查询优化

offset操作

offset在表数据量比较大的时候扫描的表数据量比较大，即开销比较大

Offset 操作的开销

```
tidb> SELECT id, name, mass FROM planets WHERE category_id = 1 ORDER BY mass DESC limit 20, 10;
+----+-----+-----+
| id | name           | mass   |
+----+-----+-----+
| 37 | .Dbaxc?y)o2U6oJQNO | 82861 |
...
| 191 | `GM{f$&YuH{-|uuU3) | 76997 |
+----+-----+-----+
10 rows in set (0.01 sec)

tidb> explain SELECT id, name, mass FROM planets WHERE category_id = 1 ORDER BY mass DESC limit 20, 10;
+-----+-----+-----+-----+
| id          | estRows | task      | access object |
+-----+-----+-----+-----+
| Projection_7 | 10.00   | root      |              |
| ↳Limit_12    | 10.00   | root      |              |
|   ↳IndexLookUp_26 | 30.00   | root      |              |
|     ↳IndexFullScan_17(Build) | 95.68   | cop[tikv] | table:planets, index:mass(mass, id) |
|       ↳Selection_19(Probe) | 30.00   | cop[tikv] |              |
|         ↳TableRowIDScan_18 | 95.68   | cop[tikv] | table:planets |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Keyset Seeker操作

Keyset Seeker 操作的开销

```
tidb> SELECT id, name, mass
    -> FROM planets WHERE category_id = 1 AND mass <= 83163 AND NOT (mass = 83163 and id >= 113)
    -> ORDER BY mass DESC, id DESC LIMIT 10;
+-----+-----+
| id   | name           | mass  |
+-----+-----+
| 37   | .Dbaxc?y}o2U6oJQNO | 82861 |
...
| 191  | `GM{f$&YuH{-|uuU3) | 76997 |
+-----+-----+
10 rows in set (0.01 sec)

tidb> explain SELECT id, name, mass
    -> FROM planets WHERE category_id = 1 AND mass <= 83163 AND NOT (mass = 83163 and id >= 113)
    -> ORDER BY mass DESC, id DESC LIMIT 10;
+-----+-----+-----+
| id       | estRows | task      | access object          |
+-----+-----+-----+
| Projection_7 | 10.00   | root     |                      |
| Limit_12   | 10.00   | root     |                      |
|   IndexLookUp_43 | 10.00   | root     |                      |
|     Selection_33(Build) | 31.89   | cop[tikv] | table:planets, index:mass(mass, id) |
|     IndexRangeScan_31 | 31.95   | cop[tikv] | table:planets, index:mass(mass, id) |
|     Selection_34(Probe) | 10.00   | cop[tikv] | table:planets          |
|     TableRowIDScan_32 | 31.89   | cop[tikv] | table:planets          |
+-----+-----+-----+
7 rows in set (0.00 sec)
```

Keyset Seeker分页的方法相对比offset分页方法，一般开销更小，特别是大表；

注意：这里的mass,id设置了索引；

Keyset Seeker分页方法

Keyset Seeker分页方法：把上一页的最后一行数据mass值作为下一页的起始值，并且通过NOT操作避免上一页最后一行被分到下一页中；

因为降序排列，如下一页的mass值肯定要小于等于90148，并且不能是mass值等于90148时，id还大于等于上一页最后一行的id值（即34334234234）的记录，即避免上一页的记录也被分到下一页中；

以每页 10 行分页查询质量 Top-30 的类地行星

- 上一页最后 1 行: | 34234234234 | xxxx | 90148 |
- 第 2 页的 10 行:

```
SELECT id, name, mass
  FROM planets
 WHERE category_id = 1
   AND mass <= 90148
   AND NOT (mass = 90148 and id >= 34234234234)
 ORDER BY mass DESC, id DESC
LIMIT 10;
+----+-----+-----+
| id | name           | mass |
+----+-----+-----+
| 233 | _^tUw]QWZ;x-'uiF'C | 90148 |
...
| 113 | &EYcK>f(e0S=KeK3qQ | 83163 |
+----+-----+-----+
10 rows in set (0.00 sec)
```

数据类型和表达式

数值数据类型

数值数据类型

- 数值数据类型:
 - 整数: 十进制数字
 - 定点数: 精确值的小数
 - 浮点数: 近似值的小数
 - Bit: 二进制比特值
- 数值数据类型要考虑的因素:
 - 要存储的值的范围
 - 所需的存储空间量
 - 精度和小数位数

整数

整数数据类型

- 数据类型:
 - TINYINT
 - SMALLINT
 - MEDIUMINT
 - INT, INTEGER
 - BIGINT
- 属性:
 - UNSIGNED: 不允许使用负数

Type	Storage	Signed Range	Unsigned Range
TINYINT	1 byte	-128 to 127	0 to 255
SMALLINT	2 bytes	-32,768 to 32,767	0 to 65,535
MEDIUMINT	3 bytes	-8,388,608 to 8,388,607	0 to 16,777,215
INT	4 bytes	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295
BIGINT	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615

定点数

定点数据类型

- 用于精确数值:整数、小数或两者
- 数据类型:
 - **DECIMAL:**
 - 固定十进制
 - 保持精度
- 示例:
 - 表示货币值: **cost DECIMAL (10,2)**
 - 示例值输出: **385.72**

decimal(22,2) 表示 小数和整数的总位数为22位，小数为2位；

如果是插入的数据小数位数多于2位，则会被截断为2位；

如果是插入的总位数多于22位，则会报错；

浮点数

精度会丢失

浮点数数据类型

- 用于近似值:整数、小数或两者
- 数据类型:
 - **FLOAT:**使用四个字节存储单精度值
 - **DOUBLE:**使用八个字节存储双精度值

浮点数据类型比较

Type	Storage	Signed Range	Unsigned Range
Float	4 bytes	-3.402823466E+38 to -1.175494351E-38, 0 1.175494351E-38 to 3.402823466E+38	1.175494351E-38 to 3.402823466E+38
Double	8 bytes	-1.7976931348623157E+308 to -2.2250738585072014E-308, 0 2.2250738585072014E-308 to 1.7976931348623157E+308	2.2250738585072014E-308 to 1.7976931348623157E+308

```
19 ##### 浮点数数据类型#####
20 * use test;
21 * drop table T2;
22 * create table T2(cost1 decimal(8,2), cost2 float(8,2));
23 * select * from T2;
24 * insert into T2 values(131072.32, 131072.32);
25
26 ##### 数值字面值#####
27 * SELECT 1.1 + 2.2 = 3.3, 1.1E0 + 2.2E0 = 3.3E0, 1.1E0 + 2.2E0;
28
```

数值字面值

数值字面值

- 数值可以是精确值或近似值的字面值。
 - 精确值字面值:
 - 在 SQL 语句中显示的即为其具体数值
 - 写为整数或十进制值,没有指数
 - 不受舍入误差产生的近似值的影响
 - 近似值字面值:
 - 并不总是按照 SQL 语句中指定的那样使用
 - 用科学记数法写成浮点数,带指数
 - 受舍入误差的影响
 - 例如:
 - `SELECT 1.1 + 2.2 = 3.3, 1.1E0 + 2.2E0 = 3.3E0, 1.1E0 + 2.2E0;`

```
mysql> SELECT 1.1 + 2.2 = 3.3, 1.1E0 + 2.2E0 = 3.3E0, 1.1E0 + 2.2E0;
+-----+-----+-----+
| 1.1 + 2.2 = 3.3 | 1.1E0 + 2.2E0 = 3.3E0 | 1.1E0 + 2.2E0 |
+-----+-----+-----+
|           1   |             0 | 3.3000000000000003 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

1.1+2.2时会自动转换为定点数，而带E的数会自动转换为浮点数，所以会出现精度丢失；

BIT数据类型

BIT 数据类型

- 数据类型:
 - **BIT**
- 比特位值:
 - 列宽 (M):每个值的比特位数
 - 1 到 64 位
- 示例:
 - 存储 4 位: **status_column BIT (4)**
- 字面值可表达为
 - **b'1101'**
 - **0b1101**

布尔类型

布尔表达式

- TiDB 考虑:
 - 常量名称:以任何字母大小写表述 **TRUE** 和 **FALSE**
 - **TRUE** 的结果为 **1**
 - **FALSE** 的结果为 **0**
- **BOOLEAN** 数据类型实际上被定义为 **TINYINT (1)**
- 例如:
 - `select 1=true, 1=TRUE, true;`

```
tidb> SELECT 1=true, 1=TRUE, true;
+-----+-----+-----+
| 1=true | 1=TRUE | TRUE |
+-----+-----+-----+
|      1 |      1 |      1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

注意: tinyint是1byte, 有符号数值范围为-128~127, 无符号范围为0~256;

tinyint(1)的1并不是整数只有1位, 123仍然可以插入到tinyint(1)类型中;

时间类型

时间数据类型

- **DATE**:
 - '2018-01-04': YYYY-MM-DD
- **TIME**:
 - '12:59:02': HH:MM:SS
- **DATETIME** 和 **TIMESTAMP**:
 - '2018-01-04 12:59:02': YYYY-MM-DD HH:MM:SS
- **YEAR**:
 - 1548: YYYY
- **TIME**, **DATETIME** 和 **TIMESTAMP** 接受带有最多 6 位数字的微秒, 格式是 **HH:MM:SS[.fraction]**
- **STRICT_TRANS_TABLE** 和 **STRICT_ALL_TABLES** 这两个 **SQL_MODE**, 会影响 TiDB 对日期零值的接受度
 - e.g: `INSERT INTO test.dropme (day) VALUES ('0000 00-00-00');`

timestamp与datetime的区别:

- timestamp: 受时区影响; 存储范围: 1970-01-01 00:00:01.000000 ~ 2038-01-19 03:04:17.999999 ;
- datetime: 不受时区的改变, 存的是什么就是什么; 存储范围: 0000-00-00 00:00:00 ~ 9999-12-31 23:59:59 ;

时间数据类型比较

Type	Storage	Scope
DATE	3 bytes	0000-01-01 to 9999-12-31
TIME	3 bytes	-838:59:59 to 838:59:59 Or -838:59:59.000000 to 838:59:59.000000
DATETIME	8 bytes	0000-01-01 00:00:00 to 9999-12-31 23:59:59 Or 0000-01-01 00:00:00.000000 to 9999-12-31 23:59:59.999999
TIMESTAMP	4 bytes	1970-01-01 00:00:01 to 2038-01-19 03:14:07 Or 1970-01-01 00:00:01.000000 to 2038-01-19 03:14:07.999999
YEAR	1 byte	1901 to 2155

时间戳和时区

- 当前时区: `SELECT @@time_zone;`
- 在被存储时, `TIMESTAMP` 值从当前时区转换为 UTC
- 在被查询时, 数据从 UTC 转换为当前时区

```
tidb> set time_zone='+09:00';
tidb> create table test.dropme (d1 timestamp, d2 datetime);
tidb> insert into test.dropme values (now(), now());
tidb> select * from test.dropme;
+-----+-----+
| d1      | d2      |
+-----+-----+
| 2022-02-06 01:12:23 | 2022-02-06 01:12:23 |
+-----+-----+
tidb> set time_zone='+08:00';
tidb> select * from test.dropme;
+-----+-----+
| d1      | d2      |
+-----+-----+
| 2022-02-06 00:12:23 | 2022-02-06 01:12:23 |
+-----+-----+
```

时间间隔关键字

- **INTERVAL** 是用于指定持续时间的关键字
- **INTERVAL** 不是一种数据类型
- e.g:
 - `SELECT '2021-01-01' + INTERVAL 10 DAY, '2021-01-01' + INTERVAL '5 2:3:4' DAY_SECOND;`

```
tidb> SELECT '2021-01-01' + INTERVAL 10 day, '2021-01-01' + INTERVAL '5 2:3:4' DAY_SECOND;
+-----+-----+
| '2021-01-01' + INTERVAL 10 day | '2021-01-01' + INTERVAL '5 2:3:4' DAY_SECOND |
+-----+-----+
| 2021-01-11 | 2021-01-06 02:03:04 |
+-----+-----+
1 row in set (0.00 sec)
```

字符串

- char 和 varchar
 - char: 固定长度；存储在定义时就已经确定，相对更浪费空间；
 - 查询速度快一点
 - varchar: 可变长度字符串；存储随实际存储的字符串变化；
 - 不能超过65535 byte
 - TiDB 默认的编码格式为UTFMB4，一个字符占4个字节（byte），所以字符个数最长为65535/4 个；
- binary 和 varbinary
- blob
- text
 -
- enum
- set

文本数据类型

文本数据类型

- **TINYTEXT**: 最大列长度为 255
- **TEXT**: 最大列长度为 65535
- **MEDIUMTEXT** and **LONGTEXT**:
 - 以上两个数据类型的最大列长度受下列系统参数影响：
 - TiDB server `txn-entry-size-limit`
 - TiKV server `raft-entry-max-size`

二进制和变量二进制

二进制和变量二进制

- 数据类型:
 - BINARY: 和 CHAR 类似, 固定长度 (静态) 二进制字节字符串
 - VARBINARY: 和 VARCHAR 类似, 可变长度 (动态) 二进制字节字符串
- BINARY 示例:
 - **bin_val BINARY (6)**: 存储 6 字节的二进制数据, 长度不足就使用 0 填充
- VARBINARY 示例:
 - **varbin_val VARBINARY (6)**: 存储最多 6 字节的二进制数据, 无填充

二进制大对象数据类型

二进制大对象数据类型

- BLOB:
 - 二进制大型对象
 - 可变长度二进制字节字符串
 - 包括 TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB 长度分别与 TEXT 类型对应

```
tidb> CREATE TABLE test.bbb (id blob);
tidb> INSERT INTO test.bbb values (cast('SD' as binary));
tidb> INSERT INTO test.bbb values ('SD');
tidb> SELECT id, cast(id as char) FROM test.bbb;
+-----+-----+
| id      | cast(id as char) |
+-----+-----+
| 0x5344  | SD           |
| 0x5344  | SD           |
+-----+-----+
```

blob: 0 ~ 65535 (2^16)

mediumblob: 0 ~ 2^32

longblob: 0 ~ 2^64

枚举和集合类型

枚举和集合类型

- 数据类型:
 - **ENUM**: 从枚举字符串值列表中选择单个字符串值
 - **SET**: 从字符串值列表中选择零个或多个字符串值
- **ENUM** 示例:
 - `ENUM('Terrestrial', 'Jovian', 'Dwarf')`
- **SET** 示例:
 - `SET('nose', 'head', 'eyes')`

```
tidb> DESC test.sss;
+-----+-----+-----+-----+-----+
| Field | Type            | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | set('A','B','C') | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
tidb> INSERT INTO test.sss VALUES ('AAA');
ERROR 1265 (01000): Data truncated for column 'id' at row 1

tidb> INSERT INTO test.sss VALUES ('C,B');
Query OK, 1 row affected (0.03 sec)
```

字符串数据类型比较

字符串数据类型比较

Type	Maximum Length
CHAR(M)	255 characters
BINARY(M)	255 bytes
VARCHAR(M),VARBINARY(M)	65,535 bytes
TINYBLOB,TINYTEXT	255 bytes
BLOB, TEXT	65,535 bytes
MEDIUMBLOB,MEDIUMTEXT	16,777,215 bytes
LONGBLOB, LONGTEXT	4,294,967,295 bytes
ENUM	65,535 values
SET	64 members

varchar, varbinary,blob,text 都是最大65535 字节，那这四个又有什么区别呢？

字符串表达式

- 使用字符串表达式时
 - 必须使用引号
 - 使用 ' 或 " 引号
 - **ANSI_QUOTES** SQL模式会将双引号内的字符解释为标识符
 - 例如: **SELECT "string_literal";**
 - 为了便于移植,应首选单引号

```
65
66 ##### 字符串表达式#####
67 • select 'AAA';
68 • select "aaa";
69
70 • set sql_mode=ansi_quotes; ↗
71
72 • select 'AAA'; ✓
73 • select "aaa"; ✗ 报错
74
75 ##### 字符集和排序规则#####
76 • show character set;
77
78 • show collation;
79
80 • select 'A'='a' collate utf8mb4_bin, 'A'='a' collate utf8mb4_general_ci, 'A'='a';
81
82
```

100% 20:70 |

Action Output

Time	Action	Response	Duration / Fetch Time
100 13:18:52	set sql_mode=ansi_quotes	0 row(s) affected	0.0097 sec
101 13:18:57	select 'AAA' LIMIT 0,1000	1 row(s) returned	0.065 sec / 0.000015...
102 13:19:02	select "aaa" LIMIT 0,1000	Error Code: 1054. Unknown column 'aaa' in 'field list'	0.022 sec

set sql_mode=ansi_quotes后双引号字符就被当作关键字了，如select、from和where等；

字符集和排序规则

- **Character Set** 是一组符号和编码。
 - 所有字符串都属于一个特定的字符集
 - TiDB 中的默认字符集是 utf8mb4
- **Collation** 是一组用于比较字符集中字符和字符排序顺序的规则
 - 影响字符和字符串的比较
 - TiDB 中的默认的规则是 utf8mb4_bin
- **SHOW CHARACTER SET**: 显示所有支持的字符集
- **SHOW COLLATION**: 显示所有支持的排序规则

```
75 ##### 字符集和排序规则#####
76 • show character set;
77
78 • show collation;
79
80 • create table T8(id varchar(32)) charset = utf8;
81
82 • select 'A'='a' collate utf8mb4_bin, 'A'='a' collate utf8mb4_general_ci, 'A'='a';
83
84 ##### CAST 函数#####
85 • select cast(' ' as binary), binary(' ');
86
87 ##### 其他 #####
100% 71:82 |
```

Result Grid Filter Rows: Search Export:

'A'='a' collate utf8mb4_bin	'A'='a' collate utf8mb4_general_ci	'A'='a'
0	1	0

Result Grid

表达式

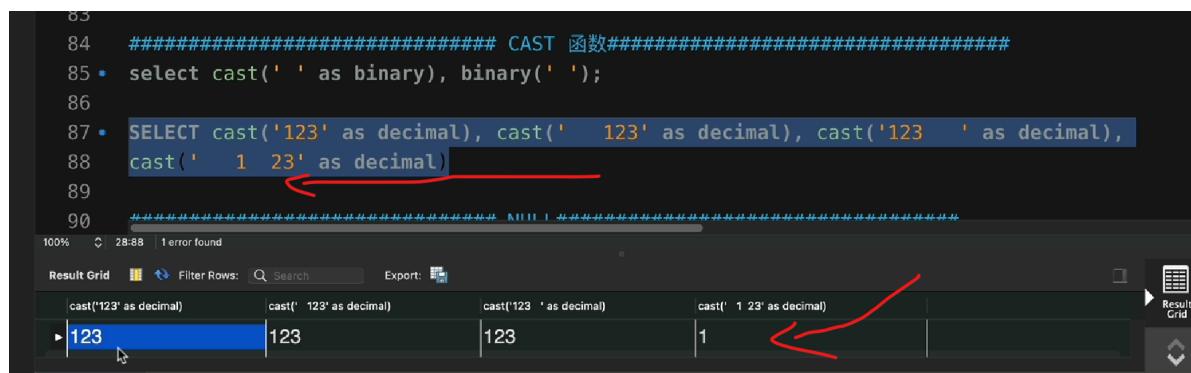
CAST函数

CAST 函数

- 语法:
 - `CAST(expression AS TYPE)`
- CAST 函数将任何类型的值转换为具有指定类型的值。目标类型可以是以下类型之一: `BINARY, CHAR, DATE, DATETIME, TIME, DECIMAL, SIGNED, UNSIGNED`
- 比如:
 - `SELECT CAST(' ' as binary), binary(' ');`

```
tidb> select cast(' ' as binary), binary(' ');
+-----+-----+
| cast(' ' as binary) | binary(' ') |
+-----+-----+
| 0x20                | 0x20          |
+-----+-----+
1 row in set (0.00 sec)
```

```
83
84  ##### CAST 函数#####
85 • select cast(' ' as binary), binary(' ');
86
87 • SELECT cast('123' as decimal), cast(' 123' as decimal), cast('123  ' as decimal),
88  cast(' 1 23' as decimal)
89
90
```



cast('123' as decimal)	cast(' 123' as decimal)	cast('123 ' as decimal)	cast(' 1 23' as decimal)
123	123	123	1

选择数据类型

选择数据类型

- **ABC 原则:**
 - **Appropriate:**合适
 - **Brief:**消耗最少的资源
 - **Complete:**数据不可丢失
- 主键的设计和考虑

Appropriate: 是时间就用时间类型，是数值就用数值类型；

NULL

NULL

- **NULL** 是一个 SQL 关键字,用于定义允许缺失值的数据类型
 - 未知: 存在值,但目前尚不知道精确值
 - 不适用: 如果指定了某个值,则该值将无法准确地具有代表性
 - 默认情况下允许空值
 - 如果列不允许空值,请使用 **NOT NULL** 声明确保数据完整性
 - **NULL** 是 **ORDER BY** 中的最小值
 - 例如:

```
tidb> SELECT NULL = NULL, NULL != NULL, NULL <=> NULL, NULL <=> 'x';
+-----+-----+-----+-----+
| NULL = NULL | NULL != NULL | NULL <=> NULL | NULL <=> 'x' |
+-----+-----+-----+-----+
|          NULL |           NULL |            1 |            0 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

NULL 安全判断符: `<=>`, 即使是NULL也能进行判断;

函数与表达式

使用函数

使用函数

- 函数可以在任何接受表达式的地方使用
- 大多数函数都需要参数才能执行
- 列可以用作参数
- 一个函数的输出可以用作另一个函数的输入
- 数学函数在出错时返回 NULL 值
 - 例如,除以零
 - 在 `ERROR_FOR_DIVISION_BY_ZERO` SQL 模式下,试图使用其返回值进行 `INSERT/DELETE` 时会报错
 - `ERROR 1365 (22012): Division by 0`

字符串比较

比较字符串

- `STRCMP()`: 比较 str1 和 str2
- 返回值:
 - 0 表示 str1 和 str2 是相同的
 - -1 表示 str1 小于 str2
 - 1 表示 str1 大于 str2
- `=`: 测试字符串是否相等
- 例如:
 - `SELECT strcmp('ABC', 'ABC'), strcmp('ABC', 'ABC'), strcmp('ABC', 'ABC');`
 - `SELECT 'ABC' = 'ABC';`

```
tidb> select strcmp('ABC','ABC'), strcmp('ABC','ABC'), strcmp('ABC','ABC');
+-----+-----+-----+
| strcmp('ABC','ABC') | strcmp('ABC','ABC') | strcmp('ABC','ABC') |
+-----+-----+-----+
| -1           | 0            | 1            |
+-----+-----+-----+
1 row in set (0.00 sec)
```

在字符串中查找字符串

在字符串中查找字符串

- **INSTR(), LOCATE(), POSITION()**: 返回指定字符串在另一个字符串中的位置
 - 例如: `select instr('TiDB','DB'), instr('TiDB','D'), locate('DB', 'TiDB'), position('DB' in 'TiDB');`
- **LOCATE(searchstr、str、pos)**: 其中 pos 指定搜索的起始位置:
 - 例如: `select locate('DB', 'TiDB is a NewSQL style DB', 7);`

```
tidb> SELECT instr('TiDB','DB'), instr('TiDB','D'), locate('DB', 'TiDB'), position('DB' in 'TiDB');
+-----+-----+-----+-----+
| instr('TiDB','DB') | instr('TiDB','D') | locate('DB', 'TiDB') | position('DB' in 'TiDB') |
+-----+-----+-----+-----+
|          3 |          3 |          3 |          3 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
tidb> SELECT locate('DB', 'TiDB is a NewSQL style DB', 7);
+-----+
| locate('DB', 'TiDB is a NewSQL style DB', 7) |
+-----+
|          24 |
+-----+
1 row in set (0.00 sec)
```

逆向、连接

- ### 逆向、连接
- **CONCAT(), CONCAT_WS()**: 将给定的参数连接成一个字符串
 - **REVERSE()**: 返回字符串,字符的顺序相反
 - 例如:
 - `SELECT reverse('placeholder'), concat('hello ','world'), concat_ws(':', 'pingcap', 'cmg', 'pe');`

```
tidb> SELECT reverse('placeholder'), concat('hello ','world'),
concat_ws(':', 'pingcap', 'cmg', 'pe');
+-----+-----+
| reverse('placeholder') | concat('hello ','world') |
+-----+-----+
| redlohecalp | hello world | pingcap:cmg:pe |
+-----+-----+
1 row in set (0.00 sec)
```

处理NULL

处理 NULL

- 传递 NULL 到 CONCAT 函数, 导致它返回 NULL
- CONCAT_WS() 忽略 NULL 参数
- 例如:

```
    •      select concat('Welcome ', 'to ', NULL), concat_ws(' ','Welcome','to', NULL);
```

```
tidb> SELECT concat('Welcome ', 'to ', NULL), concat_ws(' ','Welcome','to',NULL);
+-----+-----+
| concat('Welcome ', 'to ', NULL) | concat_ws(' ','Welcome','to',NULL) |
+-----+-----+
| NULL                                | Welcome to                         |
+-----+-----+
1 row in set (0.00 sec)
```

使用管道运算符连接字符串

使用管道运算符连接字符串

- 例如:
 - `SET sql_mode=PIPES_AS_CONCAT;`
 - `SELECT 'Welcome'||' to '||'TiDB!';`

```
tidb> set sql_mode=PIPES_AS_CONCAT;
Query OK, 0 rows affected (0.00 sec)
```

```
tidb> select 'Welcome'||' to '||'TiDB!';
+-----+
| 'Welcome'||' to '||'TiDB!' |
+-----+
| Welcome to TiDB!               |
+-----+
1 row in set (0.00 sec)
```

设置为`sql_mode=PIPES_AS_CONCAT`, '||' 才是连接符;

LEFT, RIGHT, LPAD和RPAD

LEFT, RIGHT, LPAD, RPAD

- **LEFT(str, numchar), RIGHT(str, numchar)**: 返回字符串中最左边或最右边的 `numchar` 字符
- **LPAD(str, len, padstr), RPAD(str, len, padstr)**: 返回一个在左侧或右侧填充有 `padstr` 的字符串, 长度不超过 `len` 个字符
- 例如:
 - `SELECT left('TSO',1), right('TSO',2), lpad('TSO',4,'#'), rpad('TSO',4,'#');`

```
tidb> SELECT left('TSO',1), right('TSO',2), lpad('TSO',4,'#'),  
rpad('TSO',4,'#');  
+-----+-----+-----+  
| LEFT('TSO',1) | RIGHT('TSO',2) | LPAD('TSO',4,'#') | RPAD('TSO',4,'#') |  
+-----+-----+-----+  
| T          | SO          | #TSO        | TSO#       |  
+-----+-----+-----+
```

SUBSTRING 减少字符串的一部分

SUBSTRING 检索字符串的一部分

- **SUBSTRING(string, startpos [, length])**: 返回一个子字符串, 其长度不超过字符的起始于 `startpos`
 - 如果省略 `length`, 则子字符串将包括结尾的所有字符
 - 如果 `startpos` 为负数, 则子字符串返回从末尾开始

```
tidb> SELECT substring('TiDB SQL', 2), substring('TiDB SQL', 1, 2), substring('TiDB SQL',  
-2, 2);  
+-----+-----+-----+  
| substring('TiDB SQL', 2) | substring('TiDB SQL', 1, 2) | substring('TiDB SQL', -2, 2) |  
+-----+-----+-----+  
| iDB SQL           | Ti                | QL               |  
+-----+-----+-----+
```

SUBSTRING_INDEX

SUBSTRING_INDEX

- **SUBSTRING_INDEX(str, delim, count)**: 返回分隔符出现次数 `count` 之前的字符串中的子字符串
 - 如果 `count` 为负数, 则方向反转
- 例如:
 - `SELECT substring_index('853:888:777:598',':',1),
substring_index('853:888:777:598',':',2),
substring_index('853:888:777:598',':',-2)\G`

```
tidb> SELECT substring_index('853:888:777:598',':',1),  
-> substring_index('853:888:777:598',':',2),  
-> substring_index('853:888:777:598',':',-2)\G  
***** 1. row *****  
substring_index('853:888:777:598',':',1): 853  
substring_index('853:888:777:598',':',2): 853:888  
substring_index('853:888:777:598',':',-2): 777:598  
1 row in set (0.00 sec)
```

TRIM, LTRIM, RTRIM: 修剪字符串

TRIM, LTRIM, RTRIM: 修剪字符串

- **TRIM([BOTH|LEADING|.TRAILING [substring] FROM] string)**
 - 返回全部 `substring` 移除后的字符串
 - 如果未指定 `BOTH`, `LEADING`, `TRAILING` 选项, 默认为 `BOTH`
 - 如果 `substring` 省略, 空格将被删除
- **LTRIM(), RTRIM(), TRIM()**: 可以删除出现在左侧、右侧或两侧的空格字符
- 例如:
 - `SELECT trim(leading 'Ti' from 'TiDB'), ltrim(' TiDB'), '#'||trim(' TiKV ')||'#';`

```
tidb> SET @@sql_mode=ORACLE;
Query OK, 0 rows affected (0.00 sec)
tidb> SELECT trim(leading 'Ti' from 'TiDB'), ltrim(' TiDB'), '#'||trim(' TiKV ')||'#';
+-----+-----+-----+
| trim(leading 'Ti' from 'TiDB') | ltrim(' TiDB') | '#'||trim(' TiKV ')||'#' |
+-----+-----+-----+
| DB           | TiDB        | #TiKV#      |
+-----+-----+-----+
1 row in set (0.00 sec)
```

插入和替换字符串的某些部分

插入和替换字符串的某些部分

- **REPLACE(str,substr,newstring)**: 将 `str` 中的 `substr` 替换为新字符串
- **INSERT(str,pos,len,newstring)**: 在 `str` 中的 `pos` 处放置新字符串, 覆盖 `len` 长度字符串
- 例如:
 - `SELECT replace('Use K8S to setup','K8S','tiup'), insert('Use OGG to migrate', 5, 3, 'Dumpling and Lightning')\G`

```
tidb> SELECT
-> replace(
->   'Use K8S to setup','K8S','tiup'
-> ),
-> insert(
->   'Use OGG to migrate', 5, 3, 'Dumpling and Lightning'
-> );
+-----+-----+
| replace('Use K8S to setup','K8S','tiup') | insert('Use OGG to migrate', 5, 3, 'Dumpling and Lightning') |
+-----+-----+
| Use tiup to setup                      | Use Dumpling and Lightning to migrate                |
+-----+-----+
1 row in set (0.00 sec)
```

确定字符长度

确定字符串长度

- **LENGTH()**: 以字节为单位返回字符串长度
- **CHAR_LENGTH()**: 返回字符串长度（以字符为单位）
- 例如:
 - `select length('TiDB 部署中'), char_length('TiDB 部署中');`

```
tidb> SELECT length('TiDB 部署中'), char_length('TiDB 部署中');
+-----+-----+
| length('TiDB 部署中') | char_length('TiDB 部署中') |
+-----+-----+
|          14 |                  8 |
+-----+-----+
1 row in set (0.01 sec)
```

匹配字符串模式: LIKE

匹配字符串模式: LIKE

- 使用 **LIKE** 运算符匹配字符串
 - `expression LIKE 'pattern'`
- 在模式中使用通配符
 - `%`: 匹配任何零个或多个字符的序列
 - `_`: 匹配任何单个字符
- **NOT LIKE**: 用于反向匹配

匹配字符串模式: RLIKE

匹配字符串模式: RLIKE

- 正则表达式是为复杂搜索指定模式的强大方法
- `RLIKE, REGEXP` 运算符: `expression RLIKE|REGEXP 'regexp-pattern'`
- \ 作为转义字符
- () 对多个正则表达式元素进行分组
- 例如:
 - `^`: 字符串的开头
 - `$`: 字符串的结尾
 - `.`: 任何单个字符
 - `x *`: 零个或更多 `x`
 - `x+`: 一个或多个 `x`
 - `x?`: 零个或一个 `x`

日期和时间函数

日期和时间函数

- `NOW()`: 服务器上的当前日期时间, 返回 DATETIME
- `CURDATE()`: 服务器上的当前日期, 返回 DATE
- `CURTIME()`: 当前服务器上的时间, 返回 TIME
- `YEAR()`: 年份 YEAR 格式
- `MONTHNAME()`: 一年中的月份
- `DAYNAME()`: 一周中的某天
- `DAYOFYEAR()`: 一年中的某天
- `HOUR(), MINUTE(), SECOND(), MICROSECOND()`

```
tidb> SELECT now(), curdate(), curtime(), year(now()), dayname(now());  
+-----+-----+-----+-----+  
| now() | curdate() | curtime() | year(now()) | dayname(now()) |  
+-----+-----+-----+-----+  
| 2022-01-06 10:09:10 | 2022-01-06 | 10:09:10 | 2022 | Thursday |  
+-----+-----+-----+-----+
```

日期和时间算术

日期和时间算术

- **DATE_ADD, DATE_SUB**
- **+/- INTERVAL**

```
tidb> SELECT now() + interval '1 12:12:12.123456' DAY_MICROSECOND,
      -> now() + interval '12.123456' SECOND_MICROSECOND,
      -> now() - interval '1-3' YEAR_MONTH,
      -> date_add(now(), interval 1 WEEK),
      -> date_sub(now(), interval 1 QUARTER)
      -> \G
*****
1. row *****
now() + interval '1 12:12:12.123456' DAY_MICROSECOND: 2022-01-07 22:29:02.123456
now() + interval '12.123456' SECOND_MICROSECOND: 2022-01-06 10:17:02.123456
now() - interval '1-3' YEAR_MONTH: 2020-10-06 10:16:50
date_add(now(), interval 1 WEEK): 2022-01-13 10:16:50
date_sub(now(), interval 1 QUARTER): 2021-10-06 10:16:50
```

将日期格式化为字符串

将日期格式化为字符串

- **DATE_FORMAT()**
 - http://dev.mysql.com/doc/mysql/en/date-and-time-functions.html#function_date-format
- **GET_FORMAT()**
 - 例如: `GET_FORMAT({DATE|TIME|DATETIME}, {'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL'})`

```
tidb> SELECT
      -> date_format(now(), '%a %Y-%m-%d %T'), date_format(now(), '%Y-%m-%d %r'),
      -> date_format(now(), '%W, %M %e, %Y'), date_format(now(), '%a, %b %e %l:%i %p'),
      -> date_format(now(), '%d-%b-%Y'), date_format('2013-11-23 22:23:00', '%a, %b %e, %Y'),
      -> date_format(now(), '%M %e, %Y %H:%i'), date_format(now(), '%W, the %D of %M')\G
*****
1. row *****
date_format(now(), '%a %Y-%m-%d %T'): Thu 2022-01-06 10:28:24
date_format(now(), '%Y-%m-%d %r'): 2022-01-06 10:28:24 AM
date_format(now(), '%W, %M %e, %Y'): Thursday, January 6, 2022
date_format(now(), '%a, %b %e %l:%i %p'): Thu, Jan 6 10:28 AM
date_format(now(), '%d-%b-%Y'): 06-Jan-2022
date_format('2013-11-23 22:23:00', '%a, %b %e, %Y'): Sat, Nov 23, 2013
date_format(now(), '%M %e, %Y %H:%i'): January 6, 2022 10:28
date_format(now(), '%W, the %D of %M'): Thursday, the 6th of January
```

创建日期

创建日期

- **MAKEDATE(year, dayofyear)**: 创建 DATE
 - **MAKETIME(hour,min,sec)**: 创建 TIME
 - **STR_TO_DATE(str, fmt)**: 创建 DATETIME
 - 日期字面值:
 - 例如:
 - `select GET_FORMAT(DATE, 'INTERNAL'), GET_FORMAT(TIME, 'INTERNAL'), GET_FORMAT(DATETIME, 'INTERNAL'), MAKEDATE(2020,89), MAKETIME(11,44,57)\G`
- ```
tidb> SELECT
 -> GET_FORMAT(DATE, 'INTERNAL'), GET_FORMAT(TIME, 'INTERNAL'), GET_FORMAT(DATETIME, 'INTERNAL'),
 -> MAKEDATE(2020,89), MAKETIME(11,44,57)\G

1. row *****
GET_FORMAT(DATE, 'INTERNAL'): %Y%m%d
GET_FORMAT(TIME, 'INTERNAL'): %H%i%s
GET_FORMAT(DATETIME, 'INTERNAL'): %Y%m%d%H%i%s
MAKEDATE(2020,89): 2020-03-29
MAKETIME(11,44,57): 11:44:57
1 row in set (0.00 sec)
```

# 比较日期

## 比较日期

- **DATEDIFF(expr1, expr2)**: 返回两个 DATETIME 值之间的天数
  - 例如:
    - `select DATEDIFF(now(), '2022-02-22'), DATEDIFF('2022-02-22', NOW());`

```
tidb> SELECT DATEDIFF(now(), '2022-02-22'), DATEDIFF('2022-02-22',NOW());
+-----+-----+
| DATEDIFF(now(), '2022-02-22') | DATEDIFF('2022-02-22',NOW()) |
+-----+-----+
| -44 | 44 |
+-----+-----+
1 row in set (0.00 sec)
```

# 基本算术函数

## 基本算数函数

- 示例:

```
tidb> SELECT
 -> abs(-234.5),sign(-6),
 -> floor(343.4543),ceiling(2343.34),
 -> truncate(343.4543,3),truncate(343.4543,-1),
 -> round(343.4543),round(343.4543,2),round(343.4543,-2)\G

1. row *****
abs(-234.5): 234.5
sign(-6): -1
floor(343.4543): 343
ceiling(2343.34): 2344
truncate(343.4543,3): 343.454
truncate(343.4543,-1): 340
round(343.4543): 343
round(343.4543,2): 343.45
round(343.4543,-2): 300
```

# 其他常用函数

## 其他常用函数

- 几何函数:
  - PI(), DEGREES(), RADIANS()
- 三角函数:
  - SIN(), COS(), TAN(), COT(), ASIN(), ACOS(), ATAN(), ATAN2()
- 其他功能:
  - EXP(), LN(), LOG(), LOG2(), LOG10(), POWER(), SQRT(), MOD(), RAND()

## 窗口函数

### 窗口函数

- 窗口函数为结果集中的每个输入的记录计算输出
- 除了示例中的 RANK(), TiDB 支持的窗口函数还有:
  - CUME\_DIST(), DENSE\_RANK(), FIRST\_VALUE(), LAG(), LAST\_VALUE(), LEAD(), NTH\_VALUE(), NTITLE(), PERCENT\_RANK(), ROW\_NUMBER()

```
tidb> SELECT
 -> name, mass, gravity,
 -> rank() over (partition by category_id order by mass desc) as mass_rank_within_category
 -> FROM universe.planets;
+-----+-----+-----+-----+
| name | mass | gravity | mass_rank_within_category |
+-----+-----+-----+-----+
Pluto	0.013	0.7	1
Jupiter	1898	23.1	1
Saturn	568	9.0	2
Neptune	102	11.0	3
Uranus	86.8	8.7	4
Proxima Centauri b	7.5819	11.3	1
Earth	5.97	9.8	2
Venus	4.87	8.9	3
Mars	0.642	3.7	4
Mercury	0.33	3.7	5
+-----+-----+-----+-----+
10 rows in set (0.01 sec)
```

### 获取上一行的值

```
160 ✘ select name, mass, LAG(name, 1, '---') over (order by mass)
161 from universe.planets
162 order by mass;
163
```

Result Grid    Filter Rows:    Search:    Export:

| name               | mass   | LAG(name, 1, '---') over (order by mass) |
|--------------------|--------|------------------------------------------|
| Pluto              | 0.013  | ---                                      |
| Mercury            | 0.33   | Pluto                                    |
| Mars               | 0.642  | Mercury                                  |
| Venus              | 4.87   | Mars                                     |
| Earth              | 5.97   | Venus                                    |
| Proxima Centauri b | 7.5819 | Earth                                    |
| Uranus             | 86.8   | Proxima Centauri b                       |
| Neptune            | 102    | Uranus                                   |
| Saturn             | 568    | Neptune                                  |
| Jupiter            | 1898   | Saturn                                   |

## 获取下一行的值

```
160 * select name, mass, LEAD(name, 1, '---') over (order by mass)
161 from universe.planets
162 order by mass;
163
```

Result Grid    Filter Rows    Search    Export:

| name               | mass   | LEAD(name, 1, '---') over ( order by mass) |
|--------------------|--------|--------------------------------------------|
| Pluto              | 0.013  | Mercury                                    |
| Mercury            | 0.33   | Mars                                       |
| ► Mars             | 0.642  | Venus                                      |
| Venus              | 4.87   | Earth                                      |
| Earth              | 5.97   | Proxima Centauri b                         |
| Proxima Centauri b | 7.5819 | Uranus                                     |
| Uranus             | 86.8   | Neptune                                    |
| Neptune            | 102    | Saturn                                     |
| Saturn             | 568    | Jupiter                                    |
| Jupiter            | 1898   | ---                                        |

## Flow Control函数

- case 表达式
- nullif() 函数
- ifnull() 函数
- if() 函数

## NNULLIF 和 IFNULL

- **NULLIF(expr1, expr2)**: 测试 **expr1** 和 **expr2** 是否相等
- **IFNULL(expr1, expr2)**: 返回第一个非空 **expr**
- 例如:
  - **select NULLIF('A', 'A'), NULLIF('A', 'B');**
  - **select IFNULL('A', 'B'), IFNULL(17/0, 'A');**

```
tidb> SELECT
 -> NULLIF('A', 'A'),
 -> NULLIF('A', 'B');
+-----+
| NULLIF('A', 'A') | NULLIF('A', 'B') |
+-----+
| NULL | A |
+-----+
1 row in set (0.00 sec)

tidb> SELECT
 -> IFNULL('A', 'B'),
 -> IFNULL(17/0, 'A');
+-----+
| IFNULL('A', 'B') | IFNULL(17/0, 'A') |
+-----+
| A | A |
+-----+
1 row in set (0.00 sec)
```

## IF 函数

- **IF(cond, then\_return, else\_return)**
  - 例如: `select IF('ABC' > 'aBC', '+++', '---');`

```
tidb> SELECT IF('ABC' > 'aBC', '+++', '---');
+-----+
| IF('ABC' > 'aBC', '+++', '---') |
+-----+
| --- |
+-----+
1 row in set (0.00 sec)
```

## 表达式下推

### 表达式下推

- 向 **TiKV** 下推表达式
  - 当 TiDB 从 TiKV 中读取数据的时候, TiDB 会尽量下推一些表达式运算到 TiKV 中, 从而减少数据传输量以及 TiDB 单一节点的计算压力
  - 逻辑运算: `AND (&&), OR (||), NOT (!)`
  - 比较运算: `<, <=, =, != (<>), >, >=, <=>, IN(), IS NULL, LIKE, IS TRUE, IS FALSE, COALESCE()`
  - 数值运算: `+, -, *, /, ABS(), CEIL(), CEILING(), FLOOR()`
  - 控制流运算: `CASE, IF(), IFNULL()`
  - 日期运算: `DATE_FORMAT()`
- 通过观察执行计划中任务字段的 `task.cop[tikv]` 来确定运算是否支持表达式下推

## 其他集合运算

### 差集 (EXCEPT) 和交集 (INTERSECT)

## 其他集合运算

- TiDB 支持两种集合运算:差集( `EXCEPT` )和交集( `INTERSECT` )

```
tidb> SELECT id FROM test.a EXCEPT SELECT id FROM test.c;
+-----+
| id |
+-----+
| 3 |
+-----+
tidb> SELECT id FROM test.c EXCEPT SELECT id FROM test.a;
+-----+
| id |
+-----+
| 4 |
+-----+
tidb> SELECT id FROM test.d EXCEPT SELECT id FROM test.a;
+-----+
| ID |
+-----+
| 5 |
| NULL |
+-----+
tidb> (SELECT ID FROM test.d except select id from test.a) ORDER BY ID;
+-----+
| ID |
+-----+
| NULL |
| 5 |
+-----+
```

## ANY 和 ALL

ANY: 比其中任意一个大就可以

```
3
4 ##### 比地球或火星质量大的行星#####
5 • SELECT name from universe.planets
6 WHERE mass > ANY (SELECT mass
7 FROM universe.planets
8 WHERE name in ('Earth', 'Mars'));
9
```

ALL: 比其中所有元素都大

```
13
14 ##### 比地球或火星质量大的行星#####
15 • SELECT name from universe.planets
16 WHERE mass > ALL (SELECT mass
17 FROM universe.planets
18 WHERE name in ('Earth', 'Mars'));
19
```

## IN 和 EXISTS

## IN 和 EXISTS

- IN 和 EXISTS 之间的区别

```
tidb> SELECT * FROM test.t1;
+----+
| id |
+----+
| NULL |
+----+
1 row in set (0.00 sec)
tidb> SELECT * FROM test.t1 WHERE id IN (SELECT id FROM test.t1);
Empty set (0.00 sec)
tidb> SELECT * FROM test.t1 WHERE EXISTS (SELECT id FROM test.t1);
+----+
| id |
+----+
| NULL |
+----+
1 row in set (0.01 sec)
```

in: 先把内层查询先查出来，然后再用外层查询一个一个查是否匹配成功；

exists: 先把外层查询查出来，然后再去内层查询里面查看是否匹配；

## With子句

### WITH 子句

- 针对有复杂逻辑的 SQL 语句，**WITH** 可以大幅减少临时表的数量，提升 SQL 代码的可读性和可维护性

```
WITH low_g_planets AS (
 SELECT
 p.id, p.name, p.gravity
 FROM stars s
 JOIN planets p
 ON p.sun_id = s.id
 WHERE s.name = 'Sun'
 AND p.gravity < 10
 SELECT v1.name, v1.gravity, m.name
 FROM low_g_planets AS v1
 LEFT JOIN moons m
 ON v1.id = m.planet_id;
```