MySQL笔记

1、基础知识

1.1、常用语法

> **启动和关闭**

```
启动: net start mysql;
关闭: net stop mysql;
登录: mysql -u root -p 123 -h localhost;
退出: quit;
```

> **DDL(数据定义语言，用来定义数据库对象：库、表、列等)**

1.1.1、创建和管理数据库

```
查看所有数据库名称: SHOW DATABASES;
切换数据库: USE mydb1;
创建数据库: CREATE DATABASE [IF NOT EXISTS] mydb1;
删除数据库: DROP DATABASE [IF EXISTS] mydb1;
修改数据库编码: ALTER DATABASE mydb1 CHARACTER SET utf8
```

1.1.2、创建表

```
# 创建表
CREATE TABLE emp(
    eid      INT  PRIMARY KEY AUTO_INCREMENT,
    ename    VARCHAR(50),
    age      INT,
    gender   VARCHAR(6),
    birthday     DATE,
    hiredate     DATE,
    salary   DECIMAL(7,2),
    UNIQUE (ename)
);
```

1.1.3、修改表

```
查看当前数据库中所有表名称: SHOW TABLES;
查看指定表的创建语句: SHOW CREATE TABLE emp;
查看表结构: DESC emp;
删除表: DROP TABLE emp;
修改表:
1.修改之添加列: 给stu表添加classname列:
    ALTER TABLE stu ADD (classname varchar(100));
2.修改之修改列类型: 修改stu表的gender列类型为CHAR(2):
    ALTER TABLE stu MODIFY gender CHAR(2);
3.修改之修改列名: 修改stu表的gender列名为sex:
    ALTER TABLE stu change gender sex CHAR(2);
4.修改之删除列: 删除stu表的classname列:
    ALTER TABLE stu DROP classname;
5.修改之修改表名称: 修改stu表名称为student:
    ALTER TABLE stu RENAME TO student;
```

> **DML(数据操作语言，用来定义数据库记录)**

1.1.4、插入修改删除表数据

```
插入数据
INSERT INTO stu(sid, sname) VALUES('s_1001',
'zhangSan');
INSERT INTO stu VALUES('s_1002', 'liSi', 32,
'female');
修改数据
UPDATE stu SET sname='liSi', age='20' WHERE age > 50
AND gender='male';
删除数据
DELETE FROM stu WHERE sname='chenQi' OR age > 30;
DELETE FROM stu;
truncate 是先DROP TABLE，再CREATE TABLE。而且TRUNCATE删除的记录是无法回滚的，但DELETE删除的记录是可以回滚的
TRUNCATE TABLE stu;
```

TRUNCATE删除的记录是无法回滚的，但DELETE删除的记录是可以回滚的

> **DQL：数据查询语言，用来查询记录（数据）**

```
语法：
SELECT selection_list /*要查询的列名称*/
  FROM table_list /*要查询的表名称*/
  WHERE condition /*行条件*/
  GROUP BY grouping_columns /*对结果分组*/
  HAVING condition /*分组后的行条件*/
  ORDER BY sorting_columns /*对结果分组*/
  LIMIT offset_start, row_count /*结果限定*/
```

模糊查询

- “`_”:匹配任意一个字母，5个“”表示5个任意字母
- “%”:匹配0~n个任何字母 “

```
查询姓名中第2个字母为“i”的学生记录
SELECT * FROM stu
WHERE sname LIKE '_i%';
```

聚合函数

- COUNT(): 统计指定列不为NULL的记录行数;
- MAX(): 计算指定列的最大值，是字符串类型，那么使用字符串排序运算;
- MIN(): 计算指定列的最小值，是字符串类型，那么使用字符串排序运算;
- SUM(): 计算指定列的数值和，**不是数值类型，计算结果为0**;
- AVG(): 计算指定列的平均值，**不是数值类型，那么计算结果为0**;

```
SELECT deptno, SUM(sal) FROM emp
GROUP BY deptno
HAVING SUM(sal) > 9000;
WHERE是对分组前记录的条件，如果某行记录没有满足WHERE子句的条件，那么这行记录不会参加分组；而HAVING是对分组后数据的约束
```

合并结果集

- UNION: 去除重复记录 `SELECT * FROM t1 UNION SELECT * FROM t2;`
- UNION ALL:不去除重复记录`SELECT * FROM t1 UNION ALL SELECT * FROM t2;`

> **完整性约束**

主键 : primary key

```
修改表时指定主键
ALTER TABLE stu ADD PRIMARY KEY(sid);
删除主键
ALTER TABLE stu DROP PRIMARY KEY;
```

主键自增长 : auto_increment (主键必须是整型才可以自增长)

```
修改表时设置主键自增长
ALTER TABLE stu CHANGE sid sid INT AUTO_INCREMENT;
修改表时删除主键自增长
ALTER TABLE stu CHANGE sid sid INT;
```

1.2、比较运算符

> **等号运算符**

- 等号运算符（=）判断等号两边的值、字符串或表达式是否相等，如果**相等则返回1，不相等则返回0。**
- 在使用等号运算符时，遵循如下规则：
-
  - 如果等号两边的值、字符串或表达式都为字符串，则MySQL会按照字符串进行比较，其比较的是每个字符串中字符的ANSI编码是否相等。
  - 如果等号两边的值都是整数，则MySQL会按照整数来比较两个值的大小。
  - 如果等号两边的值**一个是整数，另一个是字符串**，则MySQL会将字符串转化为数字进行比较。
  - 如果等号**两边的值有一个为NULL，则比较结果为NULL。**

```
SELECT 0 = 'x1xc', 1 = 'x1xc' , 1 = '01xc', (5 + 3) = (2 + 6), '' = NULL , NULL = NULL;
结果: 1   0   1   1   NULL    NULL
```

转换开始于字符串的第一个字符，继续直到遇到第一个无法转换为数字的字符。因此，`'01xc'`被转换成数字`1`，所以`1 = '01xc'`评估为`true`。字符串`'x1xc'`在转换为数字时，转换会在第一个字符`'x'`处失败，因为它**不是一个有效的数字。**在这种情况下，**MySQL将整个字符串解释为`0`。**

> **安全等于运算符**

安全等于运算符（⟺）与等于运算符（=）的作用是相似的，唯一的区别是**‘⟺’可以用来对NULL进行判断。在两个操作数均为NULL时，其返回值为1，而不为NULL；当一个操作数为NULL时，其返回值为0。**

```
SELECT 0 <=> 'x1xc', 1 <=> 'x1xc' , 1 <=> '01xc', ''
<=> NULL , NULL <=> NULL;
结果: 1   0   1   0   1
```

不等于运算符

不等于运算符（<>和≠）用于判断两边的数字、字符串或者表达式的值是否不相等，如果不相等则返回1，相等则返回0。**不等于运算符不能判断NULL值。如果两边的值有任意一个为NULL，或两边都为NULL，则结果为NULL。**

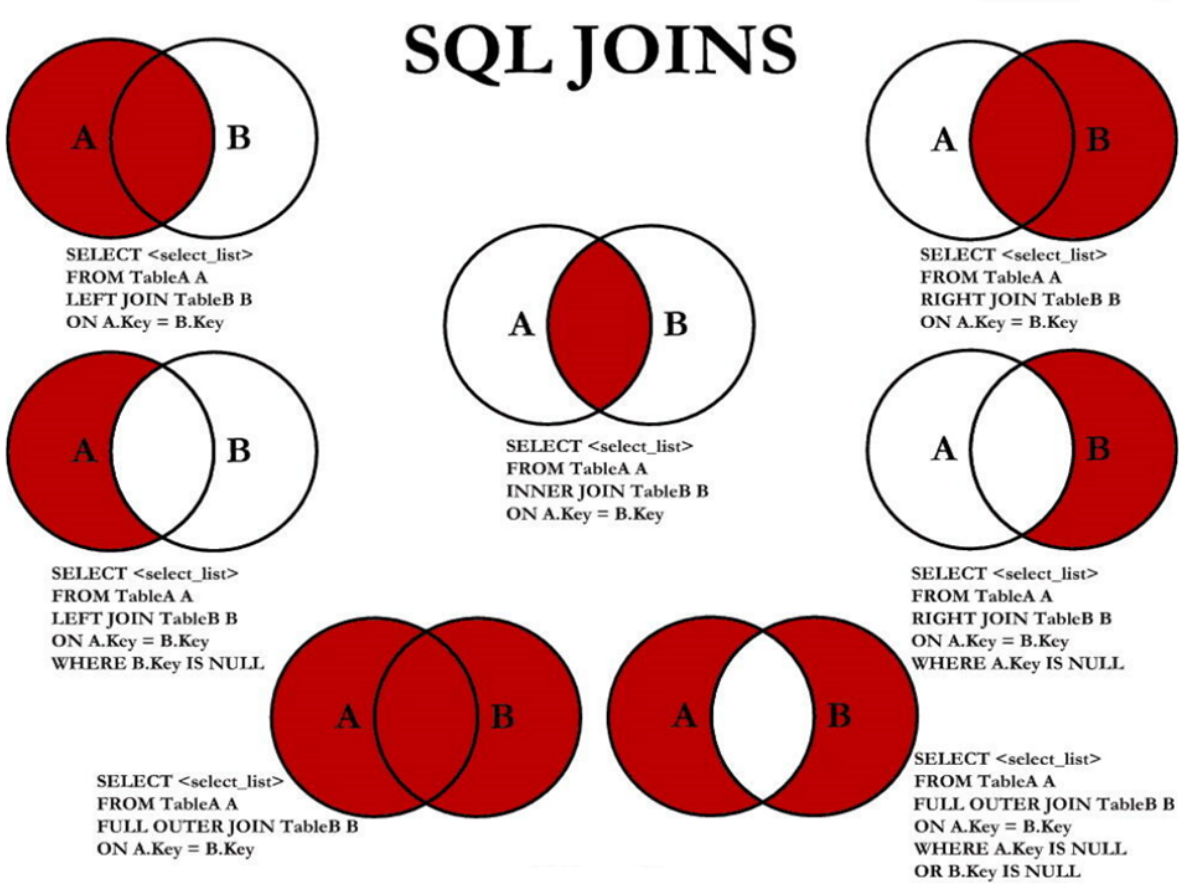
```
SELECT 'a' ≠ NULL, NULL <> NULL;
结果: NULL    NULL
```

BETWEEN AND运算符

BETWEEN运算符用于在两个值之间选择列的值范围，包括边界值。这意味着，如果列C的值位于A和B之间（包括等于A或B），那么该条件为真（true），否则为假（false）。

```
SELECT 1 BETWEEN 0 AND 1, 10 BETWEEN 11 AND 12, 'b'
BETWEEN 'a' AND 'c';
```

1.3、7种JOIN



我们要控制连接表的数量。多表连接就相当于嵌套 for 循环一样，非常消耗资源，会让 SQL 查询性能下降得很严重，因此不要连接不必要的表。在许多 DBMS 中，也都会有最大连接表的限制。

【强制】**超过三个表禁止 join**。需要 join 的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段需要有索引。

1.4、单行函数

1.4.1、基本函数

函数	用法
ABS(x)	返回x的绝对值
SIGN(X)	返回X的符号。正数返回1，负数返回-1，0返回0
PI()	返回圆周率的值
CEIL(x)，CEILING(x)	返回大于或等于某个值的最小整数
FLOOR(x)	返回小于或等于某个值的最大整数
LEAST(e1,e2,e3…)	返回列表中的最小值
GREATEST(e1,e2,e3…)	返回列表中的最大值
MOD(x,y)	**返回X除以Y后的余数**
RAND()	返回0~1的随机值
RAND(x)	返回0~1的随机值，其中x的值用作种子值，相同的X值会产生相同的随机数
ROUND(x)	**返回一个对x的值进行四舍五入后，最接近于X的整数**
ROUND(x,y)	**返回一个对x的值进行四舍五入后最接近X的值，并保留到小数点后面Y位**
TRUNCATE(x,y)	**返回数字x截断为y位小数的结果**
SQRT(x)	返回x的平方根。当X的值为负数时，返回NULL

```
SELECT
SIGN(-23),SIGN(43),PI(),CEIL(32.32),CEILING(-43.23),F
LOOR(32.32),FLOOR(-43.23),MOD(12,5);

结果: -1      1       3.141593        33      -43
32      -44     2

SELECT
ROUND(12.33),ROUND(12.343,2),ROUND(12.324,-1),TRUNCAT
E(12.66,1),TRUNCATE(12.66,-1);

结果: 12      12.34       10      12.6        10
```

1.4.2、进制转换

函数	用法
BIN(x)	返回x的二进制编码
HEX(x)	返回x的十六进制编码
OCT(x)	返回x的八进制编码
CONV(x,f1,f2)	**x是f1进制数，转换成f2进制数并输出**

```
SELECT BIN(10),HEX(10),OCT(10),CONV(10,2,8)
结果: 1010        A       12      2
```

1.4.3、字符串函数

函数	用法
ASCII(S)	返回字符串S中的第一个字符的ASCII码值
CHAR_LENGTH(s)	返回字符串s的字符数。作用与CHARACTER_LENGTH(s)相同
LENGTH(s)	**返回字符串s的字节数，和字符集有关**
CONCAT(s1,s2,.....,sn)	**连接s1,s2,.....,sn为一个字符串**
CONCAT_WS(x, s1,s2,.....,sn)	**同CONCAT(s1,s2,...)函数，但是每个字符串之间要加上x**
INSERT(str, idx, len, replacestr)	将字符串str从第idx位置开始，len个字符长的子串替换为字符串replacestr
REPLACE(str, a, b)	**用字符串b替换字符串str中所有出现的字符串a**
UPPER(s) 或 UCASE(s)	**将字符串s的所有字母转成大写字母**
LOWER(s) 或LCASE(s)	**将字符串s的所有字母转成小写字母**
LEFT(str,n)	返回字符串str最左边的n个字符
RIGHT(str,n)	返回字符串str最右边的n个字符
LPAD(str, len, pad)	用字符串pad对str最左边进行填充，直到str的长度为len个字符
RPAD(str ,len, pad)	用字符串pad对str最右边进行填充，直到str的长度为len个字符
LTRIM(s)	去掉字符串s左侧的空格
RTRIM(s)	去掉字符串s右侧的空格
TRIM(s)	**去掉字符串s开始与结尾的空格**
TRIM(s1 FROM s)	去掉字符串s开始与结尾的s1
TRIM(LEADING s1 FROM s)	去掉字符串s开始处的s1
TRIM(TRAILING s1 FROM s)	去掉字符串s结尾处的s1

函数	用法
REPEAT(str, n)	返回str重复n次的结果
SPACE(n)	返回n个空格
STRCMP(s1,s2)	**比较字符串s1,s2的ASCII码值的大小**
SUBSTR(s,index,len)	**返回从字符串s的index位置其len个字符，作用与SUBSTRING(s,n,len)相同**
LOCATE(substr,str)	返回字符串substr在字符串str中首次出现的位置，作用于POSITION(substr IN str)、INSTR(str,substr)相同。未找到，返回0
ELT(m,s1,s2,…,sn)	返回指定位置的字符串，如果m=1，则返回s1，如果m=2，则返回s2，如果m=n，则返回sn
FIELD(s,s1,s2,…,sn)	返回字符串s在字符串列表中第一次出现的位置
FIND_IN_SET(s1,s2)	返回字符串s1在字符串s2中出现的位置。其中，字符串s2是一个以逗号分隔的字符串
REVERSE(s)	**返回s反转后的字符串**
NULLIF(value1,value2)	比较两个字符串，如果value1与value2相等，则返回NULL，否则返回value1

CONCAT(s1, s2, ..., sn)

```
SELECT CONCAT('Hello', ' ', 'World');
结果: 'Hello World'
```

CONCAT_WS(separator, s1, s2, ..., sn)

```
SELECT CONCAT_WS('-', '2023', '03', '18');
结果: '2023-03-18'
```

LENGTH(s)

```
SELECT LENGTH('Hello World');
结果取决于字符集，如果是UTF-8，结果是11
```

UPPER(s) 或 UCASE(s)

```
SELECT UPPER('Hello World');
结果: 'HELLO WORLD'
```

LOWER(s) 或 LCASE(s)

```
SELECT LOWER('HELLO WORLD');
结果: 'hello world'
```

TRIM(s)

```
SELECT TRIM('    Hello World    ');
结果: 'Hello World'
```

SUBSTR(s, start, length) 或 SUBSTRING(s, start, length)

```
SELECT SUBSTR('Hello World', 7, 5);
结果: 'World'
```

REPLACE(str, from_str, to_str)

```
SELECT REPLACE('Hello World', 'World', 'SQL');
结果: 'Hello SQL'
```

STRCMP(s1, s2)

```
SELECT STRCMP('Hello', 'World');
结果：若'Hello' < 'World'，则为负数；若'Hello' >
'World'，则为正数；若相等，则为0
```

REVERSE(str)

```
select REVERSE('Curry')
结果：yrruC
```

instr()函数

mysql的内置函数instr(filed,str)，作用是**返回str子字符串在filed字符串的第一次出现的位置。当instr(filed,str)=0时，表示子符串str不存在于字符串filed中**，因此可以用来实现mysql中的模糊查询，与like用法类似。如下：

```
instr(filed,str) > 0 ⇒ file like '%str%'
instr(filed,str) = 1 ⇒ file like  'str%'
instr(filed,str) = 0 ⇒ file not like  '%str%'
```

1.4.4、日期函数

函数	用法
YEAR(date) / MONTH(date) / DAY(date)	**返回具体的日期值**
HOUR(time) / MINUTE(time) / SECOND(time)	**返回具体的时间值**
MONTHNAME(date)	返回月份：January，...
DAYNAME(date)	返回星期几：MONDAY，TUESDAY.....SUNDAY
WEEKDAY(date)	**返回周几，注意，周1是0，周2是1，。。。周日是6**
QUARTER(date)	返回日期对应的季度，范围为1～4
WEEK(date) ，WEEKOFYEAR(date)	返回一年中的第几周
DAYOFYEAR(date)	返回日期是一年中的第几天
DAYOFMONTH(date)	返回日期位于所在月份的第几天
DAYOFWEEK(date)	返回周几，注意：周日是1，周一是2，。。。周六是7

```
SELECT
YEAR(CURDATE()),MONTH(CURDATE()),DAY(CURDATE()),
HOUR(CURTIME()),MINUTE(NOW()),SECOND(SYSDATE())
FROM DUAL;

结果: 2024    3    19   13   35   30

SELECT MONTHNAME('2021-10-26'),DAYNAME('2021-10-
26'),WEEKDAY('2021-10-26'),
QUARTER(CURDATE()),WEEK(CURDATE()),DAYOFYEAR(NOW()),
DAYOFMONTH(NOW()),DAYOFWEEK(NOW())
FROM DUAL;

结果: October Tuesday 1    1    11   79   19   3
```

> 时间和秒钟转换的函数

函数	用法
TIME_TO_SEC(time)	将 time 转化为秒并返回结果值。转化的公式为：小时*3600*+分钟 60+秒
SEC_TO_TIME(seconds)	将 seconds 描述转化为包含小时、分钟和秒的时间

```
SELECT TIME_TO_SEC(NOW())
结果: 49262

SELECT SEC_TO_TIME(49262)
结果: 13:41:02
```

> 计算日期和时间的函数

函数	用法
DATE_ADD(datetime, INTERVAL expr type), ADDDATE(date,INTERVAL expr type)	返回与给定日期时间相差INTERVAL时间段的日期时间
DATE_SUB(date,INTERVAL expr type), SUBDATE(date,INTERVAL expr type)	返回与date相差INTERVAL时间间隔的日期

```
SELECT DATE_ADD('2023-03-18', INTERVAL 1 YEAR);
结果: '2024-03-18'

SELECT DATE_SUB('2023-03-18', INTERVAL 1 YEAR);
结果: '2022-03-18'

SELECT DATE_ADD('2023-03-18', INTERVAL 2 MONTH);
结果: '2023-05-18'
```

```
SELECT DATE_SUB('2023-03-18', INTERVAL 2 MONTH);
结果: '2023-01-18'

SELECT DATE_ADD('2023-03-18', INTERVAL 20 DAY);
结果: '2023-04-07'

SELECT DATE_SUB('2023-03-18', INTERVAL 10 DAY);
结果: '2023-03-08'

SELECT DATE_ADD('2023-03-18 12:00:00', INTERVAL 5
HOUR);
结果: '2023-03-18 17:00:00'

SELECT DATE_SUB('2023-03-18 12:00:00', INTERVAL 5
HOUR);
结果: '2023-03-18 07:00:00'

SELECT DATE_ADD('2023-03-18 12:00:00', INTERVAL 30
MINUTE);
结果: '2023-03-18 12:30:00'

SELECT DATE_SUB('2023-03-18 12:00:00', INTERVAL 30
MINUTE);
结果: '2023-03-18 11:30:00'

SELECT DATE_ADD('2023-03-18 12:00:00', INTERVAL 45
SECOND);
结果: '2023-03-18 12:00:45'

SELECT DATE_SUB('2023-03-18 12:00:00', INTERVAL 45
SECOND);
结果: '2023-03-18 11:59:15'
```

函数	用法
ADDTIME(time1,time2)	返回time1加上time2的时间。
SUBTIME(time1,time2)	返回time1减去time2后的时间。
DATEDIFF(date1,date2)	返回date1 - date2的日期间隔天数
TIMEDIFF(time1, time2)	返回time1 - time2的时间间隔
LAST_DAY(date)	返回date所在月份的最后一天的日期

```
SELECT ADDTIME('10:00:00', '02:30:00') AS NewTime; # 12:30:00


SELECT DATEDIFF('2023-04-01', '2023-03-01') AS DiffDays; # 31
SELECT TIMEDIFF('18:00:00', '16:00:00') AS TimeDifference; # 02:00:00
SELECT LAST_DAY('2023-03-15') AS LastDayOfMonth; # 2023-03-31
```

> 日期的格式化与解析

函数	用法
DATE_FORMAT(date,fmt)	**按照字符串fmt格式化日期date值**
TIME_FORMAT(time,fmt)	**按照字符串fmt格式化时间time值**
GET_FORMAT(date_type,format_type)	返回日期字符串的显示格式
STR_TO_DATE(str, fmt)	按照字符串fmt对str进行解析，解析为一个日期

```
SELECT DATE_FORMAT('2023-03-18', '%Y年%m月%d日') AS
FormattedDate;
2023年03月18日

SELECT DATE_FORMAT('2023-03-18', '%W, %M %d, %Y') AS
FormattedDate;
Saturday, March 18, 2023

SELECT TIME_FORMAT('18:45:30', '%H小时%i分%s秒') AS
FormattedTime;
18小时45分30秒
```

1.4.5、流程控制函数

函数	用法
IF(value,value1,value2)	如果value的值为TRUE，返回value1，否则返回value2
IFNULL(value1, value2)	如果value1不为NULL，返回value1，否则返回value2
CASE WHEN 条件1 THEN 结果1 WHEN 条件2 THEN 结果2 .... [ELSE resultn] END	相当于Java的if...else if...else...
CASE expr WHEN 常量值1 THEN 值1 WHEN 常量值1 THEN 值1 .... [ELSE 值n] END	相当于Java的switch...case...

```
SELECT IF(1 > 0,'正确','错误')
正确

SELECT IFNULL(NULL,'Hello Word')
Hello Word

SELECT CASE
```

```
    WHEN 1 > 0 THEN '1 > 0'
    WHEN 2 > 0 THEN '2 > 0'
    ELSE '3 > 0'
END AS ComparisonResult;
1 > 0

SELECT
    employee_id,
    salary,
CASE

        WHEN salary ≥ 15000 THEN
        '高薪'
        WHEN salary ≥ 10000 THEN
        '潜力股'
        WHEN salary ≥ 8000 THEN
        '屌丝' ELSE '草根'
    END "描述"
FROM
    employees;
```

1.4.6、补充 timestampdiff

```
datediff函数，返回值是相差的天数，不能定位到小时、分钟和秒。

-- 相差2天
select datediff('2018-03-22 09:00:00', '2018-03-20
07:00:00');


TIMESTAMPDIFF函数，有参数设置，可以精确到天（DAY）、小时（HOUR），分钟（MINUTE）和秒（SECOND），使用起来比datediff函数更加灵活。对于比较的两个时间，时间小的放在前面，时间大的放在后面。

--相差1天
```

```
select TIMESTAMPDIFF(DAY, '2018-03-20 23:59:00',
'2015-03-22 00:00:00');

--相差49小时
select TIMESTAMPDIFF(HOUR, '2018-03-20 09:00:00',
'2018-03-22 10:00:00');

--相差2940分钟
select TIMESTAMPDIFF(MINUTE, '2018-03-20 09:00:00',
'2018-03-22 10:00:00');

--相差176400秒
select TIMESTAMPDIFF(SECOND, '2018-03-20 09:00:00',
'2018-03-22 10:00:00');
```

1.5、聚合函数

1.5.1、聚合函数（计算）

> AVG和SUM函数

可以对**数值型数据**使用AVG和SUM函数。

```
SELECT AVG(salary), MAX(salary),MIN(salary),
SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

> MIN和MAX函数

可以对**任意数据类型**的数据使用MIN和MAX函数。

```
SELECT MIN(hire_date), MAX(hire_date)
FROM      employees;
```

> COUNT函数

COUNT(*)返回表中记录总数，适用于**任意数据类型**。

```
SELECT COUNT(*)
FROM    employees
WHERE  department_id = 50;
```

COUNT(字段) 返回**字段不为空**的记录总数。

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 50;
```

1.5.2、GROUP BY

可以使用GROUP BY子句将表中的数据分成若干组

```
SELECT column, group_function(column)
FROM table
[WHERE  condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id ;
```

> GROUP BY中使用WITH ROLLUP

使用**WITH ROLLUP关键字**之后，在所有查询出的分组记录之后增加一条记录，该记录**计算查询出的所有记录的总和**，即统计记录数量。

当使用ROLLUP时，不能同时使用ORDER BY子句进行结果排序，即ROLLUP和ORDER BY是互相排斥的。

```
SELECT department_id,AVG(salary)
FROM employees
WHERE department_id > 50
GROUP BY department_id WITH ROLLUP;

第一行显示department_id为60的部门平均工资为6400.00。
第二行显示department_id为90的部门平均工资为19333.33。
最后一行NULL代表所有符合条件(department_id > 50)记录的总体平均工资,为12866.666667。
```

1.5.3、HAVING

过滤分组：HAVING子句

1. 行已经被分组。
2. 使用了聚合函数。
3. 满足HAVING 子句中条件的分组将被显示。
4. **HAVING 不能单独使用，必须要跟 GROUP BY 一起使用。**

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

1.5.4、SELECT执行顺序

> 关键词顺序

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...
ORDER BY ... LIMIT...
```

> 执行顺序

```
FROM → WHERE → GROUP BY → HAVING → SELECT 的字段 -
> DISTINCT → ORDER BY → LIMIT
```

1.6、子查询

操作符	含义
=	equal to
>	greater than
⩾	greater than or equal to
<	less than
⩽	less than or equal to
<>	not equal to

子查询指一个查询语句嵌套在另一个查询语句内部的查询。

```
SELECT last_name,salary
FROM employees
WHERE salary > (
        SELECT salary
        FROM employees
        WHERE last_name = 'Abel'
        );
```

1.6.1、单行子查询

> HAVING 中的子查询

- 首先执行子查询。
- 向主查询中的HAVING子句返回结果。

题目：查询最低工资大于50号部门最低工资的部门id和其最低工资

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```

> CASE中的子查询

在CASE表达式中使用单列子查询：

题目：显式员工的employee_id,last_name和location。其中，若员工department_id与location_id为1800的department_id相同，则location为'Canada'，其余则为'USA'。

如果子查询`(SELECT department_id FROM departments WHERE location_id = 1800)`返回值为0或空字符串，它也将被视为WHEN条件成立，从而返回'Canada'。**这一点是因为SQL在进行条件判断时，将所有非NULL值都视为TRUE，只有NULL被视为FALSE。**

```
SELECT employee_id, last_name,
       (CASE department_id
        WHEN
             (SELECT department_id FROM departments
          WHERE location_id = 1800)
        THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

1.6.2、多行子查询

操作符	含义
IN	等于列表中的**任意一个**
ANY	需要和单行比较操作符一起使用，和子查询返回的**某一个**值比较
ALL	需要和单行比较操作符一起使用，和子查询返回的**所有**值比较
SOME	实际上是ANY的别名，作用相同，一般常使用ANY

题目：查询平均工资最低的部门id

```
#方式1:
SELECT department_id
FROM employees
GROUP BY department_id
HAVING AVG(salary) = (
            SELECT MIN(avg_sal)
            FROM (
                SELECT AVG(salary) avg_sal
                FROM employees
                GROUP BY department_id
                ) dept_avg_sal
            )

#方式2:
SELECT department_id
FROM employees
GROUP BY department_id
HAVING AVG(salary) <= ALL (
                SELECT AVG(salary) avg_sal
                FROM employees
                GROUP BY department_id
)
```

2、MySQL数据类型

2.1、整数类型

整数类型	字节	有符号数取值范围	无符号数取值范围
TINYINT	1	-128~127	0~255
SMALLINT	2	-32768~32767	0~65535
MEDIUMINT	3	-8388608~8388607	0~16777215
INT、INTEGER	**4**	**-2147483648~2147483647**	**0~4294967295**
BIGINT	8	-9223372036854775808~9223372036854775807	0~18446744073709551615

```
CREATE TABLE test_int1 ( x TINYINT,  y SMALLINT,  z MEDIUMINT,  m INT,  n BIGINT );
```

```
CREATE TABLE test_int3( f1 INT UNSIGNED);
```

- TINYINT：一般用于枚举数据，比如系统设定取值范围很小且固定的场景。
- SMALLINT：可以用于较小范围的统计数据，比如统计工厂的固定资产库存数量等。
- MEDIUMINT：用于较大整数的计算，比如车站每日的客流量等。
- INT、INTEGER：取值范围足够大，一般情况下不用考虑超限问题，用得最多。比如商品编号。
- BIGINT：只有当你处理特别巨大的整数时才会用到。比如双十一的交易量、大型门户网站点击量、证券公司衍生产品持仓等。

2.2、浮点数类型

浮点数和定点数类型的特点是可以处理小数，你可以把整数看成小数的一个特例。因此，浮点数和定点数的使用场景，比整数大多了。 MySQL支持的浮点数类型，分别是 FLOAT、DOUBLE、REAL。

- FLOAT 表示单精度浮点数；
- DOUBLE 表示双精度浮点数；

类型	有符号数取值范围	无符号数取值范围	占用字节数
FLOAT	(-3.402823466E+38，-1.175494351E-38)，0，(1.175494351 E-38，3.402823466351 E+38)	0，(1.175494351 E-38，3.402823466 E+38)	4
DOUBLE	(-1.7976931348623157E+308，-2.2250738585072014E-308)，0，(2.2250738585072014E-308，1.7976931348623157E+308)	0，(2.2250738585072014E-308，1.7976931348623157E+308)	8

- REAL默认就是 DOUBLE。如果你把 SQL 模式设定为启用“REAL_AS_FLOAT”，那 么，MySQL 就认为 REAL 是 FLOAT。如果要启用“REAL_AS_FLOAT”，可以通过以下 SQL 语句实现：

```
SET sql_mode = “REAL_AS_FLOAT”;
```

问题1：FLOAT 和 DOUBLE 这两种数据类型的区别是啥呢？

FLOAT 占用字节数少，取值范围小；DOUBLE 占用字节数多，取值范围也大。

问题2：为什么浮点数类型的无符号数取值范围，只相当于有符号数取值范围的一半，也就是只相当于有符号数取值范围大于等于零的部分呢？

MySQL 存储浮点数的格式为：符号(S)、尾数(M)和 阶码(E)。因此，**无论有没有符号，MySQL 的浮点数都会存储表示符号的部分。因此， 所谓的无符号数取值范围，其实就是有符号数取值范围大于等于零的部分。**

MySQL 用 4 个字节存储 FLOAT 类型数据，用 8 个字节来存储 DOUBLE 类型数据。无论哪个，都是采用二进制的方式来进行存储的。比如 9.625，用二进制来表达，就是 1001.101，或者表达成 1.001101×2^3。**如果尾数不是 0 或 5（比如 9.624），你就无法用一个二进制数来精确表达。**进而，就只好在取值允许的范围内进行四舍五入

在编程中，如果用到浮点数，要特别注意误差问题，**因为浮点数是不准确的，所以我们要避免使用“=”来判断两个数是否相等。**同时，在一些对精确度要求较高的项目中，千万不要使用浮点数，不然会导致结果错误，甚至是造成不可挽回的损失。那么，MySQL 有没有精准的数据类型呢？当然有，这就是定点数类型：DECIMAL。

2.3、定点数

数据类型	字节数	含义
DECIMAL(M,D),DEC,NUMERIC	M+2字节	有效范围由M和D决定

- 使用 DECIMAL(M,D) 的方式表示高精度小数。其中，**M被称为精度，D被称为标度。0≤M≤65，0≤D≤30，D<M。例如，定义DECIMAL（5,2）的类型，表示该列取值范围是-999.99~999.99。**
- DECIMAL(M,D)的最大取值范围与DOUBLE类型一样，但是有效的数据范围是由M和D决定的。DECIMAL 的存储空间并不是固定的，由精度值M决定，总共占用的存储空间为M+2个字节。也就是说，在一些对精度要求不高的场景下，比起占用同样字节长度的定点数，浮点数表达的数值范围可以更大一些。
- **定点数在MySQL内部是以字符串的形式进行存储**，这就决定了它一定是精准的。
- 当DECIMAL类型不指定精度和标度时，其默认为DECIMAL(10,0)。当数据的精度超出了定点数类型的精度范围时，则MySQL同样会进行四舍五入处理。
- 浮点数 vs 定点数
    - 浮点数相对于定点数的优点是在长度一定的情况下，浮点类型取值范围大，但是不精准，适用于需要取值范围大，又可以容忍微小误差的科学计算场景（比如计算化学、分子建模、流体动力学等）
    - 定点数类型取值范围相对小，但是精准，没有误差，适合于对精度要求极高的场景 （比如涉及金额计算的场景）

由于 DECIMAL 数据类型的精准性，在我们的项目中，除了极少数（比如商品编号）用到整数类型外，其他的数值都用的是 DECIMAL，原因就是这个项目所处的零售行业，要求精准，一分钱也不能差。

```
CREATE TABLE test_decimal1(
f1 DECIMAL,
f2 DECIMAL(5,2)
);

DESC test_decimal1;

INSERT INTO test_decimal1(f1,f2)
VALUES(123.123,123.456); // 四舍五入 123   123.46

#Out of range value for column 'f2' at row 1
INSERT INTO test_decimal1(f2)
VALUES(1234.34); // 直接报错
```

2.4、位类型：BIT

BIT类型中存储的是二进制值，类似010110。

二进制字符串类型	长度	长度范围	占用空间
BIT(M)	M	1 ≤ M ≤ 64	约为(M + 7)/8个字节

BIT类型，如果没有指定(M)，默认是1位。这个1位，表示只能存1位的二进制值。这里(M)是表示二进制的位数，位数最小值为1，最大值为64。

```
CREATE TABLE test_bit1(
f1 BIT,
f2 BIT(5),
f3 BIT(64)
);

INSERT INTO test_bit1(f1)
VALUES(1);

// Data too long for column 'f1' at row 1, 报错
```

```
INSERT INTO test_bit1(f1)
VALUES(2);

// 10111
INSERT INTO test_bit1(f2)
VALUES(23);
```

注意：在向BIT类型的字段中插入数据时，一定要确保插入的数据在BIT类型支持的范围内。

使用SELECT命令查询位字段时，可以用BIN()或HEX()函数进行读取。

```
SELECT * FROM test_bit1;
+------------+------------+------------+
| f1         | f2         | f3         |
+------------+------------+------------+
| 1          | NULL       | NULL       |
| NULL       | 10111      | NULL       |
+------------+------------+------------+

SELECT BIN(f2),HEX(f2) FROM test_bit1;
+---------+---------+
| BIN(f2) | HEX(f2) |
+---------+---------+
| NULL    | NULL    |
| 10111   | 17      |
+---------+---------+

SELECT f2 + 0 FROM test_bit1;
+--------+
| f2 + 0 |
+--------+
|   NULL |
|     23 |
+--------+
```

可以看到，使用列+0查询数据时，可以直接查询出存储的十进制数据的值。

2.5、日期与时间类型

类型	名称	字节	日期格式	最小值	最大值
YEAR	年	1	YYYY或YY	1901	2155
TIME	时间	3	HH:MM:SS	-838:59:59	838:59:59
DATE	**日期**	**3**	**YYYY-MM-DD**	**1000-01-01**	**9999-12-03**
DATETIME	**日期时间**	**8**	**YYYY-MM-DD HH:MM:SS**	**1000-01-01 00:00:00**	**9999-12-31 23:59:59**
TIMESTAMP	日期时间	4	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00 UTC	2038-01-19 03:14:07UTC

可以看到，不同数据类型表示的时间内容不同、取值范围不同，而且占用的字节数也不一样，你要根据实际需要灵活选取。

为什么时间类型 TIME 的取值范围不是 -23:59:59～23:59:59 呢？原因是 MySQL 设计的 TIME 类型，不光表示一天之内的时间，而且可以用来表示一个时间间隔，这个时间间隔可以超过 24 小时。

> YEAR

YEAR类型用来表示年份，在所有的日期时间类型中所占用的存储空间最小，只需要**1个字节**的存储空间。

以4位字符串或数字格式表示YEAR类型，其格式为YYYY，最小值为1901，最大值为2155。

> DATE类型

DATE类型表示日期，没有时间部分，格式为YYYY-MM-DD，其中，YYYY表示年份，MM表示月份，DD表示日期。需要3个字节的存储空间。在向DATE类型的字段插入数据时，同样需要满足一定的格式条件。

- 以YYYY-MM-DD格式或者YYYYMMDD格式表示的字符串日期，其最小取值为1000-01-01，最大取值为9999-12-03。**YYYYMMDD格式会被转化为YYYY-MM-DD格式。**
- 使用CURRENT_DATE()或者NOW()函数，会插入当前系统的日期。

> TIME类型

TIME类型用来表示时间，不包含日期部分。在MySQL中，需要**3个字节**的存储空间来存储TIME类型的数据，可以使用**“HH:MM:SS”**格式来表示TIME类型，其中，HH表示小时，MM表示分钟，SS表示秒。

> DATETIME类型

DATETIME类型在所有的日期时间类型中占用的存储空间最大，总共需要8个字节的存储空间。在格式上为DATE类型和TIME类型的组合，可以表示为**YYYY-MM-DD HH:MM:SS，其中YYYY表示年份，MM表示月份，DD表示日期，HH表示小时，MM表示分钟，SS表示秒。**

在向DATETIME类型的字段插入数据时，同样需要满足一定的格式条件。以YYYY-MM-DD HH:MM:SS格式或者YYYYMMDDHHMMSS格式的字符串插入DATETIME类型的字段时，最小值为1000-01-01 00:00:00，最大值为9999-12-03 23:59:59。

2.6、文本字符串类型

> CHAR与VARCHAR类型

CHAR和VARCHAR类型都可以存储比较短的字符串。

字符串(文本)类型	特点	长度	长度范围	占用的存储空间
CHAR(M)	固定长度	M	0 ≤ M ≤ 255	M个字节
VARCHAR(M)	可变长度	M	0 ≤ M ≤ 65535	(实际长度 + 1)个字节

CHAR类型:

- CHAR(M) 类型一般需要预先定义字符串长度。如果不指定(M)，则表示长度默认是1个字符。
- **如果保存时，数据的实际长度比CHAR类型声明的长度小，则会在右侧填充空格以达到指定的长度。**当MySQL检索CHAR类型的数据时，CHAR类型的字段会去除尾部的空格。
- 定义CHAR类型字段时，**声明的字段长度即为CHAR类型字段所占的存储空间的字节数。**

VARCHAR类型:

- VARCHAR(M) 定义时，必须指定长度M，否则报错。
- MySQL4.0版本以下，varchar(20)：指的是20字节，如果存放UTF8汉字时，只能存6个（每个汉字3字节） ；**MySQL5.0版本以上，varchar(20)：指的是20字符。**
- 检索VARCHAR类型的字段数据时，会保留数据尾部的空格。**VARCHAR类型的字段所占用的存储空间为字符串实际长度加1个字节。**

哪些情况使用 CHAR 或 VARCHAR 更好

类型	特点	空间上	时间上	适用场景
CHAR(M)	固定长度	浪费存储空间	效率高	存储不大，速度要求高
VARCHAR(M)	可变长度	节省存储空间	效率低	非CHAR的情况

情况1： 存储很短的信息。比如门牌号码101，201……这样**很短的信息应该用char**，因为**varchar还要占个byte用于存储信息长度**，本来打算节约存储的，结果得不偿失。

情况2：固定长度的。比如使用uuid作为主键，那用**char应该更合适**。因为他固定长度，varchar动态根据长度的特性就消失了，而且还要占个长度信息。

InnoDB存储引擎，建议使用VARCHAR类型。**主要影响性能的因素是数据行使用的存储总量**，由于char平均占用的空间多于varchar，所以**除了简短并且固定长度的，其他考虑varchar。**

> **阿里巴巴《Java开发手册》之MySQL数据库**

- **任何字段如果为非负数，必须是 UNSIGNED**
- 【强制】小数类型为 DECIMAL，禁止使用 FLOAT 和 DOUBLE。
-
  - 说明：在存储的时候，FLOAT 和 DOUBLE 都存在精度损失的问题，很可能在比较值的时候，得到不正确的结果。如果存储的数据范围超过 DECIMAL 的范围，建议将数据拆成整数和小数并分开存储。
- 【强制】如果存储的字符串长度几乎相等，使用 CHAR 定长字符串类型。
- 【强制】**VARCHAR 是可变长字符串，不预先分配存储空间，长度不要超过 5000。如果存储长度大于此值，定义字段类型为 TEXT，独立出来一张表，用主键来对应，避免影响其它字段索引效率。**

3、约束

为了保证数据的完整性，SQL规范以约束的方式对**表数据进行额外的条件限制**。从以下四个方面考虑：

- 实体完整性（Entity Integrity）：例如，同一个表中，不能存在两条完全相同无法区分的记录
- 域完整性（Domain Integrity）：例如：年龄范围0-120，性别范围“男/女”
- 引用完整性（Referential Integrity）：例如：员工所在部门，在部门表中要能找到这个部门
- 用户自定义完整性（User-defined Integrity）：例如：用户名唯一、密码不能为空等，本部门经理的工资不得高于本部门职工的平均工资的5倍。

> 什么是约束

约束是表级的强制规定。可以在**创建表时规定约束（通过 CREATE TABLE 语句）**，或者在**表创建之后通过 ALTER TABLE 语句规定约束。**

- **根据约束起的作用**，约束可分为：
- - **NOT NULL非空约束，规定某个字段不能为空**
  - **UNIQUE唯一约束，规定某个字段在整个表中是唯一的**
  - **PRIMARY KEY 主键(非空且唯一)约束**
  - **DEFAULT默认值约束**

> 查询表中的约束

```
SELECT * FROM information_schema.table_constraints
WHERE table_schema = 'mydb1' -- MySQL中的数据库名
AND table_name = 'countries';
```

3.1、非空约束

`NOT NULL`

- 默认，所有的类型的值都可以是NULL，包括INT、FLOAT等数据类型
- 非空约束只能出现在表对象的列上，**只能某个列单独限定非空，不能组合非空**
- 一个表可以有很多列都分别限定了非空
- **空字符串''不等于NULL，0也不等于NULL**

> 建表时

```
语句
CREATE TABLE 表名称(
    字段名   数据类型,
    字段名   数据类型 NOT NULL,
    字段名   数据类型 NOT NULL
);
```

```
举例
CREATE TABLE student(
    sid int,
    sname varchar(20) not null,
    tel char(11) ,
    cardid char(18) not null
);
```

> 建表后

```
语句
alter table 表名称 modify 字段名 数据类型 not null;

举例
alter table student modify sname varchar(20) not
null;
```

> 删除非空约束

```
语句
alter table 表名称 modify 字段名 数据类型 NULL;#去掉not
null，相当于修改某个非注解字段，该字段允许为空
或
alter table 表名称 modify 字段名 数据类型;#去掉not null，
相当于修改某个非注解字段，该字段允许为空

举例
ALTER TABLE emp
MODIFY NAME VARCHAR(15) DEFAULT 'abc' NULL;
```

3.2、唯一性约束

UNIQUE

- 同一个表可以有多个唯一约束。
- **唯一约束可以是某一个列的值唯一，也可以多个列组合的值唯一。**

- 唯一性约束允许列值为空。
- **在创建唯一约束的时候，如果不给唯一约束命名，就默认和列名相同。**
- **MySQL会给唯一约束的列上默认创建一个唯一索引。**

> 建表时

```
语句
create table 表名称(
    字段名  数据类型,
    字段名  数据类型  unique,
    字段名  数据类型  unique key,
    字段名  数据类型
);
create table 表名称(
    字段名  数据类型,
    字段名  数据类型,
    字段名  数据类型,
    [constraint 约束名] unique key(字段名)
);

举例
create table student(
    sid int,
    sname varchar(20),
    tel char(11) unique,
    cardid char(18) unique key
);
```

> 建表后

```
语句
#字段列表中如果是一个字段，表示该列的值唯一。如果是两个或更多个字段，那么复合唯一，即多个字段的组合是唯一的
#方式1:
alter table 表名称 add unique key(字段列表);
alter table 表名称 modify 字段名 字段类型 unique;

举例
alter table student add unique key(tel);
alter table student add unique key(cardid);
```

> 复合唯一约束

```
#选课表
create table student_course(
    id int,
    sid int,
    cid int,
    score int,
    unique key(sid,cid)  #复合唯一
);
```

> 删除唯一约束

删除唯一约束只能通过删除唯一索引的方式删除。

```
show index FROM tableName

-- 上面的key_name列
ALTER TABLE USER
DROP INDEX uk_name_pwd;
```

3.3、主键约束

primary key

- 主键约束列不允许重复，也不允许出现空值(NULL)。
- 一个表最多只能有一个主键约束，建立主键约束可以在列级别创建，也可以在表级别上创建。
- **主键约束对应着表中的一列或者多列（复合主键）**
- **如果是多列组合的复合主键约束，那么这些列都不允许为空值，并且组合的值不允许重复。**
- **MySQL的主键名总是PRIMARY**，就算自己命名了主键约束名也没用。
- 当创建主键约束时，系统默认会在所在的列或列组合上建立对应的**主键索引**（能够根据主键查询的，就根据主键查询，效率更高）。如果删除主键约束了，主键约束对应的索引就自动删除了。

> 建表时

```
语句
create table 表名称(
    字段名   数据类型   primary key, #列级模式
    字段名   数据类型,
    字段名   数据类型
);

举例
create table temp(
    id int primary key,
    name varchar(20)
);

CREATE TABLE emp4(
id INT PRIMARY KEY AUTO_INCREMENT ,
NAME VARCHAR(20)
);
```

> 建表后

```
语句
ALTER TABLE 表名称 ADD PRIMARY KEY(字段列表); #字段列表可以是一个字段，也可以是多个字段

举例
ALTER TABLE student ADD PRIMARY KEY (sid);
ALTER TABLE emp5 ADD PRIMARY KEY(NAME,pwd);
```

> 复合主键

```
create table student_course(
    sid int,
    cid int,
    score int,
    primary key(sid,cid)  #复合主键
);

CREATE TABLE emp6(
id INT NOT NULL,
NAME VARCHAR(20),
pwd VARCHAR(15),
CONSTRAINT emp7_pk PRIMARY KEY(NAME,pwd)
);
```

> 删除主键约束

```
语句
alter table 表名称 drop primary key;

举例
ALTER TABLE student DROP PRIMARY KEY;
```

当你删除一个表的主键约束时，仅仅是移除了这个字段作为表的主键的特性，以及由此带来的唯一性保证。然而，主键字段在创建时被定义为非空（NOT NULL），**即便主键约束被删除，字段的非空属性仍然保留。**

3.4、自增列

auto_increment

（1）**一个表最多只能有一个自增长列**

（2）当需要产生唯一标识符或顺序值时，可设置自增长

（3）**自增长列约束的列必须是键列（主键列，唯一键列）**

（4）**自增约束的列的数据类型必须是整数类型**

> 建表时

```
语句
create table 表名称(
    字段名  数据类型  primary key auto_increment,
    字段名  数据类型  unique key not null,
    字段名  数据类型  unique key,
    字段名  数据类型  not null default 默认值,
);
create table 表名称(
    字段名  数据类型 default 默认值 ,
    字段名  数据类型 unique key auto_increment,
    字段名  数据类型 not null default 默认值,,
    primary key(字段名)
);

举例
create table employee(
    eid int primary key auto_increment,
    ename varchar(20)
);
```

> 建表后

```
语句
alter table 表名称 modify 字段名 数据类型
auto_increment;

举例
alter table employee modify eid int auto_increment;
```

> 删除自增约束

```
语句
alter table 表名称 modify 字段名 数据类型; #去掉
auto_increment相当于删除

举例
alter table employee modify eid int;
```

3.5、DEFAULT约束

DEFAULT

> 建表时

```
语句
create table 表名称(
    字段名  数据类型  primary key,
    字段名  数据类型  unique key not null,
    字段名  数据类型  unique key,
    字段名  数据类型  not null default 默认值,
);
create table 表名称(
    字段名  数据类型 default 默认值 ,
    字段名  数据类型 not null default 默认值,
    字段名  数据类型 not null default 默认值,
    primary key(字段名),
    unique key(字段名)
);
```

```
举例
create table employee(
    eid int primary key,
    ename varchar(20) not null,
    gender char default '男',
    tel char(11) not null default '' #默认是空字符串
);
```

> 建表后

- **如果这个字段原来有非空约束，你还保留非空约束，那么在加默认值约束时，还得保留非空约束，否则非空约束就被删除了**
- **在给某个字段加非空约束也一样，如果这个字段原来有默认值约束，你想保留，也要在modify语句中保留默认值约束，否则就删除了**

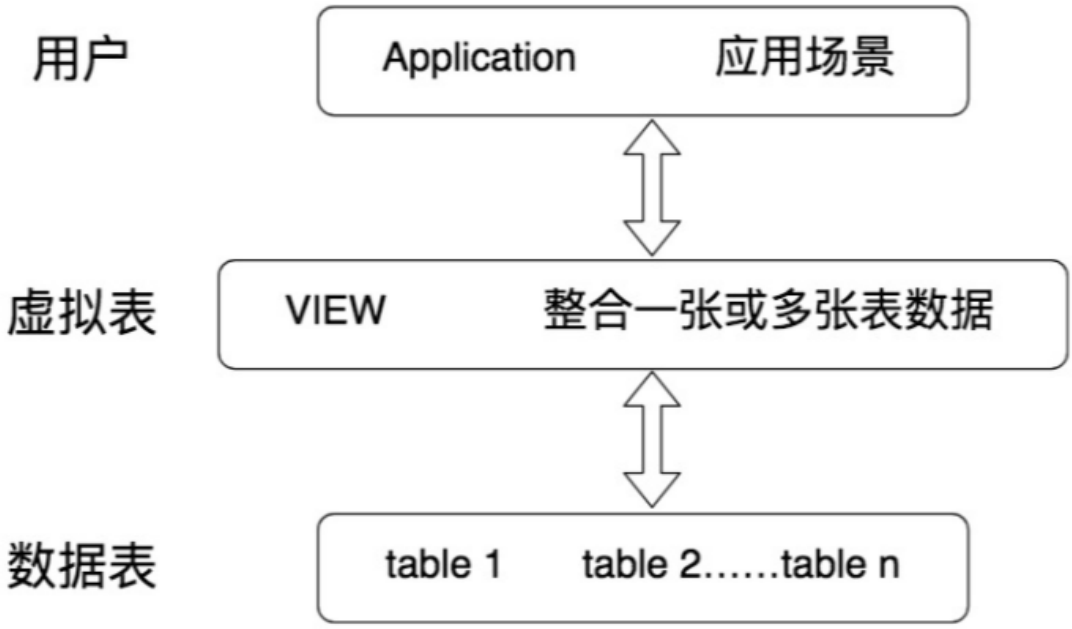
```
语句
alter table 表名称 modify 字段名 数据类型 default 默认值;

alter table 表名称 modify 字段名 数据类型 default 默认值
not null;

例子
create table employee(
    eid int primary key,
    ename varchar(20),
    gender char,
    tel char(11) not null
);
alter table employee modify tel char(11) default '';
#给tel字段增加默认值约束
alter table employee modify tel char(11) default ''
 not null;#给tel字段增加默认值约束，并保留非空约束
```

> 删除默认值约束

```
删除默认值约束，也不保留非空约束
alter table 表名称 modify 字段名 数据类型 ;

删除默认值约束，保留非空约束
alter table 表名称 modify 字段名 数据类型  not null;

删除gender字段默认值约束，如果有非空约束，也一并删除
alter table employee modify gender char;

删除tel字段默认值约束，保留非空约束
alter table employee modify tel char(11)  not null;
```

4、视图

- 视图是一种虚拟表，本身是不具有数据的，占用很少的内存空间，它是 SQL 中的一个重要概念。
- **视图建立在已有表的基础上**，视图赖以建立的这些表称为**基表**。



- 向视图提供数据内容的语句为 SELECT 语句，可以将视图理解为**存储起来的SELECT语句**
-
  - **在数据库中，视图不会保存数据，数据真正保存在数据表中。当对视图中的数据进行增加、删除和修改操作时，数据表中的数据会相应地发生变化；反之亦然。**

- 视图，是向用户提供基表数据的另一种表现形式。通常情况下，小型项目的数据库可以不使用视图，但是在大型项目中，以及数据表比较复杂的情况下，视图的价值就凸显出来了，它可以帮助我们把经常查询的结果集放到虚拟表中，提升使用效率。理解和使用起来都非常方便。

4.1、创建视图

```
CREATE VIEW 视图名称
AS 查询语句
```

4.1.1、基于数据表创建视图

> 创建视图(表原始字段名称)

```
CREATE VIEW empvu80
AS
SELECT  employee_id, last_name, salary
FROM    employees
WHERE   department_id = 90;

SELECT *
FROM    salvu80;
```

> 创建视图(表字段名称别名)

```
CREATE VIEW emp_year_salary (ename,year_salary)
AS
SELECT CONCAT(first_name,last_name),salary*12*
(1+IFNULL(commission_pct,0))
FROM employees;

select * from emp_year_salary
```

说明1：**实际上就是我们在 SQL 查询语句的基础上封装了视图 VIEW**，这样就会基于 SQL 语句的结果集形成一张虚拟表。

说明2：在创建视图时，没有在视图名后面指定字段列表，则视图中字段列表默认和SELECT语句中的字段列表一致。**如果SELECT语句中给字段取了别名，那么视图中的字段名和别名相同。**

4.1.2、基于视图创建视图

当我们创建好一张视图之后，还可以在它的基础上继续创建视图。

举例：联合“emp_dept”视图和“emp_year_salary”视图查询员工姓名、部门名称、年薪信息创建 “emp_dept_ysalary”视图。

```
CREATE VIEW emp_dept_ysalary
AS
SELECT emp_dept.ename,dname,year_salary
FROM emp_dept INNER JOIN emp_year_salary
ON emp_dept.ename = emp_year_salary.ename;
```

4.2、查看视图

> 查看视图的结构

```
DESC / DESCRIBE empvu80;

employee_id int             NO
last_name   varchar(25)     YES
salary      decimal(8,2)    YES
```

> 查看视图的属性信息

记得视图名称要加单引号

```
SHOW TABLE STATUS LIKE 'empvu80'
```

> 查看视图的详细定义信息

```
SHOW CREATE VIEW empvu80

empvu80 CREATE ALGORITHM=UNDEFINED
DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`empvu80` AS select `employees`.`employee_id` AS
`employee_id`,`employees`.`last_name` AS
`last_name`,`employees`.`salary` AS `salary` from
`employees` where (`employees`.`department_id` = 90)
utf8mb4 utf8mb4_0900_ai_ci
```

4.3、修改、删除视图

> 修改视图

ALTER VIEW

```
ALTER VIEW 视图名称
AS
查询语句
```

> 删除视图

删除视图只是删除视图的定义，并不会删除基表的数据。

```
语句
DROP VIEW IF EXISTS 视图名称;

举例
DROP VIEW empvu80;
```

基于视图a、b创建了新的视图c，如果将视图a或者视图b删除，会导致视图c的查询失败。这样的视图c需要手动删除或修改，否则影响使用。