必学必会SQL

数据类型

MySQL中的数据类型

类型	含义
tinyint	1字节，带符号的范围是-128到127。无符号的范围是0到255。bool或boolean，就是tinyint，0表示假，非0表示真
smallint	2字节，带符号的范围是-32768到32767。无符号的范围是0到65535
int	整型，4字节，同Integer，带符号的范围是-2147483648到2147483647。无符号的范围是0到4294967295
bigint	长整型，8字节，带符号的范围是-9223372036854775808到9223372036854775807。无符号的范围是0到18446744073709551615
float	单精度浮点数精确到大约7位小数位
double	双精度浮点数精确到大约15位小数位
DATE	日期。支持的范围为'1000-01-01'到'9999-12-31'
DATETIME	支持的范围是'1000-01-01 00:00:00'到'9999-12-31 23:59:59'
TIMESTAMP	时间戳。范围是'1970-01-01 00:00:00'到2037年
char(M)	固定长度，右边填充空格以达到长度要求。M为长度，范围为0~255。M指的是字符个数
varchar(M)	变长字符串。M 表示最大列长度。M的范围是0到65,535。但不能突破行最大字节数65535
text	大文本。最大长度为65535(2^16-1)个字符
BLOB	大字节。最大长度为65535(2^16–1)字节的BLOB列

LENGTH函数返回字节数。而char和varchar定义的M是字符数限制。
char可以将字符串定义为固定长度，空间换时间，效率略高；varchar为变长，省了空间。

关系操作

关系：在关系数据库中，关系就是二维表。
关系操作就是对表的操作。

选择（selection）：又称为限制，是从关系中选择出满足给定条件的元组。
投影（projection）：在关系上投影就是从选择出若干属性列组成新的关系。
连接（join）：将不同的两个关系连接成一个关系。

使用DDL建一张表用来测试

```sql
CREATE TABLE `reg` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `loginname` varchar(20) NOT NULL,
  `name` varchar(48) DEFAULT NULL,
  `password` varchar(64) NOT NULL,
  PRIMARY KEY (`id`)
  UNIQUE KEY `loginname` (`loginname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

DML —— CRUD 增删改查

Insert语句

```sql
INSERT INTO table_name (col_name,...) VALUES (value1,...);
-- 向表中插入一行数据，自增字段、缺省值字段、可为空字段可以不写

INSERT INTO table_name SELECT ... ;
-- 将select查询的结果插入到表中

INSERT INTO table_name (col_name1,...) VALUES (value1,...) ON DUPLICATE KEY UPDATE col_name1=value1,...;
-- 如果主键冲突、唯一键冲突就执行update后的设置。这条语句的意思，就是主键不在新增记录，主键在就更新部分字段。

INSERT IGNORE INTO table_name (col_name,...) VALUES (value1,...);
-- 如果主键冲突、唯一键冲突就忽略错误，返回一个警告。
```

```sql
INSERT INTO reg (loginname, `name`, `password`) VALUES ('tom', 'tom', 'tom');
INSERT INTO reg (id, loginname, `name`, `password`) VALUES (5, 'tom', 'tom', 'tom');
INSERT INTO reg (id, loginname, `name`, `password`) VALUES (1, 'tom', 'tom', 'tom') ON DUPLICATE KEY UPDATE name = 'jerry';
```

Update语句

```sql
UPDATE [IGNORE] tbl_name SET col_name1=expr1 [, col_name2=expr2 ...] [WHERE where_definition]
-- IGNORE 意义同Insert语句

UPDATE reg SET name='张三' WHERE id=5;
```

```sql
-- 注意这一句非常危险，会更新所有数据
UPDATE reg SET name = 'ben';

-- 更新一定要加条件
UPDATE reg SET name = 'ben', password = 'benpwd' WHERE id = 1;
```

Delete语句

```
DELETE FROM tbl_name [WHERE where_definition]
-- 删除符合条件的记录
```

```
-- 删除一定要有条件
DELETE FROM reg WHERE id = 1;
```

Select语句

```
SELECT
    [DISTINCT]
    select_expr, ...
    [FROM table_references
    [WHERE where_definition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_definition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC] , ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

查询

查询的结果成为结果集recordset。

```
SELECT 1;
select 1 as id;

-- 最简单的查询
SELECT * FROM employees;
-- 字符串合并
SELECT emp_no, first_name + last_name FROM employees;
SELECT emp_no, CONCAT(first_name,' ',last_name) FROM employees;

-- AS 定义别名，可选。写AS是一个好习惯
SELECT emp_no as `no`, CONCAT(first_name,' ',last_name) name FROM employees
emp;
```

Limit子句

```
-- 返回5条记录
SELECT * FROM employees emp LIMIT 5;

-- 返回5条记录，偏移18条
SELECT * FROM employees emp LIMIT 5 OFFSET 18;
SELECT * FROM employees emp LIMIT 18, 5;
```

Where子句

运算符	描述
=	等于
<>	不等于
>、<、>=、<=	大于、小于、大于等于、小于等于
BETWEEN	在某个范围之内，between a and b等价于[a, b]
LIKE	字符串模式匹配，%表示任意多个字符，_表示一个字符 LIKE忽略大小写；LIKE BINARY区分大小写
IN	指定针对某个列的多个可能值
AND	与
OR	或

注意：如果很多表达式需要使用AND、OR计算逻辑表达式的值的时候，由于有结合律的问题，建议使用小括号来避免产生错误

```
-- 条件查询
SELECT * FROM employees WHERE emp_no < 10015 and last_name LIKE 'P%';
SELECT * FROM employees WHERE emp_no BETWEEN 10010 AND 10015 AND last_name LIKE 'P%';
SELECT * FROM employees WHERE emp_no in (10001, 10002, 10010);
```

Order by子句

对查询结果进行排序，可以升序ASC、降序DESC。

先以第一个字段排序，第一字段相同再以第二个字段排序。

```
-- 降序
SELECT * FROM employees WHERE emp_no in (10001, 10002, 10010) ORDER BY emp_no DESC;

SELECT * FROM dept_emp ORDER BY emp_no, dept_no DESC;
```

DISTINCT

不返回重复记录

```
-- DISTINCT使用
SELECT DISTINCT dept_no from dept_emp;
SELECT DISTINCT emp_no from dept_emp;
SELECT DISTINCT dept_no, emp_no from dept_emp;
```

聚合函数

函数	描述
COUNT(expr)	返回记录中记录的数目，如果指定列，则返回非NULL值的行数
COUNT(DISTINCT expr, [expr...])	返回不重复的非NULL值的行数
AVG([DISTINCT] expr)	返回平均值，返回不同值的平均值
MIN(expr), MAX(expr)	最小值，最大值
SUM([DISTINCT] expr)	求和，Distinct返回不同值求和

```
-- 聚合函数
SELECT COUNT(*), AVG(emp_no), SUM(emp_no), MIN(emp_no), MAX(emp_no) FROM
employees;
```

分组查询

使用Group by子句，如果有条件，使用Having子句过滤分组、聚合过的结果。

```
-- 聚合所有
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries;
-- 聚合被选择的记录
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries WHERE
emp_no < 10003;

-- 分组
SELECT emp_no FROM salaries GROUP BY emp_no;
SELECT emp_no FROM salaries WHERE emp_no < 10003 GROUP BY emp_no;

-- 按照不同emp_no分组，每组分别聚合
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries WHERE
emp_no < 10003 GROUP BY emp_no;

-- HAVING子句对分组结果过滤
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries GROUP
BY emp_no HAVING AVG(salary) > 45000;
-- 使用别名
SELECT emp_no, SUM(salary), AVG(salary) AS sal_avg, COUNT(emp_no) from
salaries GROUP BY emp_no HAVING sal_avg > 60000;

-- 最后对分组过滤后的结果排序
SELECT emp_no, SUM(salary), AVG(salary) AS sal_avg, COUNT(emp_no) from
salaries GROUP BY emp_no HAVING sal_avg > 60000 ORDER BY sal_avg;
```

```
分组是将数据按照指定的字段分组，最终每组只能出来一条记录。这就带来了问题，每一组谁做代表，其实谁做代表都不合适。
如果只投影分组字段、聚合数据，不会有问题，如果投影非分组字段，显示的时候不能确定是组内谁的数据。

```

```
SELECT emp_no, salary FROM salaries GROUP BY emp_no;
返回如下：
emp_no salary
10001  60117
10002  65828
10003  40006
10004  40054
在某些数据库中，上面这一句不一定能执行成功，会报salary字段错误。

-- 分组
SELECT emp_no, MAX(salary) FROM salaries; -- 10001  88958
SELECT emp_no, MIN(salary) FROM salaries; -- 10001  40006
上例很好的说明了使用了聚合函数，虽然没有显式使用Group By语句，但是其实就是把所有记录当做一组，每组只能出一条，那么一组也只能出一条，所以结果就一条。
但是emp_no就是非分组字段，那么它就要开始覆盖，所以，显示为10001。当求最大值的时候，正好工资表中10001的工资最高，感觉是对的。但是，求最小工资的时候，明明最小工资是10003的40006，由于emp_no不是分组字段，导致最后被覆盖为10001。

SELECT emp_no, MIN(salary) FROM salaries GROUP BY emp_no;
上句才是正确的语义，按照不同员工emp_no工号分组，每一个人一组，每一个人有多个工资记录，按时每组只能按照人头出一条记录。
```

```
-- 单表较为复杂的语句
SELECT
    emp_no,
    avg(salary) AS avg_salary
FROM
    salaries
WHERE
    salary > 70000
GROUP BY
    emp_no
HAVING
    avg(salary) > 50000
ORDER BY
    avg_salary DESC
LIMIT 1;
```


子查询

查询语句可以嵌套，内部查询就是子查询。
子查询必须在一组小括号中。
子查询中不能使用Order by。

```
-- 子查询
SELECT * FROM employees WHERE emp_no in (SELECT emp_no from employees WHERE emp_no > 10015) ORDER BY emp_no DESC;

SELECT emp.emp_no, emp.first_name, gender FROM (SELECT * from employees WHERE emp_no > 10015) AS emp WHERE emp.emp_no < 10019 ORDER BY emp_no DESC;
```

连接Join

交叉连接cross join
笛卡尔乘积，全部交叉
在MySQL中，CROSS JOIN从语法上说与INNER JOIN等同

Join会构建一张临时表

```
-- 工资40行
SELECT * FROM salaries;
-- 20行
SELECT * FROM employees;
-- 800行
SELECT * from employees CROSS JOIN salaries;
-- 隐式连接，800行
SELECT * FROM employees, salaries;
```

注意：salaries和employees表不应该直接做笛卡尔乘积，这样关联只是为了看的清楚

内连接
inner join，省略为join。
等值连接，只选某些field相等的元组（行），使用On限定关联的结果
自然连接，特殊的等值连接，会去掉重复的列。用的少。

```
-- 内连接，笛卡尔乘积 800行
SELECT * from employees JOIN salaries;
SELECT * from employees INNER JOIN salaries;

-- ON等值连接 40行
SELECT * from employees JOIN salaries ON employees.emp_no = salaries.emp_no;

-- 自然连接，去掉了重复列，且自行使用employees.emp_no = salaries.emp_no的条件
SELECT * from employees NATURAL JOIN salaries;
```

外连接
outer join，可以省略为join
分为左外连接，即左连接；右外连接，即右连接；全外连接

```
-- 左连接
SELECT * from employees LEFT JOIN salaries ON employees.emp_no =
salaries.emp_no;
-- 右连接
SELECT * from employees RIGHT JOIN salaries ON employees.emp_no =
salaries.emp_no;
-- 这个右连接等价于上面的左连接
SELECT * from salaries RIGHT JOIN employees ON employees.emp_no =
salaries.emp_no;
```

左外连接、右外连接
`SELECT * from employees RIGHT JOIN salaries ON employees.emp_no = salaries.emp_no;`
结果是先employees后salaries的字段显示，Right是看表的数据的方向，从salaries往employees看，以salaries为准，它的所有数据都显示

自连接

表，自己和自己连接

```
假设有表manager，字段和记录如下
empno    name     mgr
1        tom
2        jerry    1
3        ben      2

-- 有领导的员工
SELECT * from manager WHERE mgr IS NOT NULL

-- 所有有领导的员工及其领导名字
SELECT worker.*, mgr.name as leader from manager worker INNER JOIN manager
mgr ON mgr.id = worker.mgr
```