



主键:

在关系数据库中，一张表中的每一行数据被称为一条记录。一条记录就是由多个字段组成的。

对于关系表，有个很重要的约束，就是任意两条记录不能重复。不能重复不是指两条记录不完全相同，而是指能够通过某个字段唯一区分出不同的记录，这个字段被称为主键。

选取主键的一个基本原则是：不使用任何业务相关的字段作为主键。主键也不应该允许NULL。

在一张表中，通过某个字段，可以把数据与另一张表关联起来，这种列称为外键。

外键并不是通过列名实现的，而是通过定义外键约束实现的：

ALTER TABLE students

ADD CONSTRAINT fk\_class\_id

FOREIGN KEY (class\_id)

REFERENCES classes (id);

其中，外键约束的名称fk\_class\_id可以任意，FOREIGN KEY (class\_id)指定了class\_id作为外键，REFERENCES classes (id)指定了这个外键将关联到classes表的id列（即classes表的主键）。

由于外键约束会降低数据库的性能，大部分互联网应用程序为了追求速度，并不设置外键约束，而是仅靠应用程序自身来保证逻辑的正确性。这种情况下，class\_id仅仅是一个普通的列，只是它起到了外键的作用而已。

要删除一个外键约束，也是通过ALTER TABLE实现的：

ALTER TABLE students

DROP FOREIGN KEY fk\_class\_id;

注意：删除外键约束并没有删除外键这一列。删除列是通过DROP COLUMN ...实现的。

索引：

索引是关系数据库中对某一列或多个列的值进行预排序的数据结构。通过使用索引，可以让数据库系统不必扫描整个表，而是直接定位到符合条件的记录，这样就大大加快了查询速度。



如果要经常根据score列进行查询，就可以对score列创建索引：

ALTER TABLE students

ADD INDEX idx\_score (score);

使用ADD INDEX idx\_score (score)就创建了一个名称为idx\_score，使用列score的索引。索引名称是任意的，索引如果有多列，可以在括号里依次写上，例如：

ALTER TABLE students

ADD INDEX idx\_name\_score (name, score);

唯一索引

在设计关系数据表的时候，看上去唯一的列，例如身份证号、邮箱地址等，因为他们具有业务含义，因此不宜作为主键。

但是，这些列根据业务要求，又具有唯一性约束：即不能出现两条记录存储了同一个身份证号。这个时候，就可以给该列添加一个唯一索引。例如，我们假设students表的name不能重复：

ALTER TABLE students

ADD UNIQUE INDEX uni\_name (name);

也可以只对某一列添加一个唯一约束而不创建唯一索引：

ALTER TABLE students

ADD CONSTRAINT uni\_name UNIQUE (name);

基本查询：

SELECT \* FROM <表名>

SELECT是关键字，表示将要执行一个查询，\*表示“所有列”，FROM表示将要从哪个表查询

SELECT查询的结果是一个二维表。

条件查询：

SELECT语句可以通过WHERE条件来设定查询条件，查询结果是满足查询条件的记录。

条件查询的语法就是：

SELECT \* FROM <表名> WHERE <条件表达式>

例如，要指定条件“分数在80分或以上的学生”，写成WHERE条件就是：

SELECT \* FROM students WHERE score >= 80

条件表达式可以用<条件1> AND <条件2>表达满足条件1并且满足条件2。

例如，符合条件“分数在80分或以上”，并且还符合条件“男生”，把这两个条件写出来：

条件1：根据score列的数据判断：score >= 80；

条件2：根据gender列的数据判断：gender = 'M'，注意gender列存储的是字符串，需要用单引号括起来。

就可以写出WHERE条件：

SELECT \* FROM students WHERE score >= 80 AND gender = 'M';

第二种条件是<条件1> OR <条件2>，表示满足条件1或者满足条件2。

例如，把上述AND查询的两个条件改为OR，查询结果就是“分数在80分或以上”或者“男生”，满足任意之一的条件即选出该记录；

第三种条件是NOT <条件>，表示“不符合该条件”的记录。例如，写一个“不是2班的学生”这个条件，可以先写出“是2班的学生”：class\_id = 2，再加上NOT：NOT class\_id = 2：

SELECT \* FROM students WHERE NOT class\_id = 2;

要组合三个或者更多的条件，就需要用小括号()表示如何进行条件运算。

例如，编写一个复杂的条件：分数在80以下或者90以上，并且是男生：

SELECT \* FROM students WHERE (score < 80 OR score > 90) AND gender = 'M';

如果不加括号，条件运算按照NOT、AND、OR的优先级进行，即NOT优先级最高，其次是AND，最后是OR。加上括号可以改变优先级。

投影查询：

使用SELECT \*表示查询表的所有列，使用SELECT 列1, 列2, 列3则可以仅返回指定列，这种操作称为投影。

例如，从students表中返回id、score和name这三列：

SELECT id, score, name FROM students;

使用SELECT 列1, 列2, 列3 FROM ...时，还可以给每一列起个别名，这样，结果集的列名就可以与原表的列名不同。它的语法是SELECT 列1 别名1, 列2 别名2, 列3 别名3 FROM ...。

排序:

按照成绩从低到高进行排序：

SELECT id, name, gender, score FROM students ORDER BY score;

如果要反过来，按照成绩从高到底排序，我们可以加上DESC表示“倒序”：

SELECT id, name, gender, score FROM students ORDER BY score DESC;

如果score列有相同的数据，要进一步排序，可以继续添加列名。例如，使用ORDER BY score DESC, gender表示先按score列倒序，如果有相同分数的，再按gender列排序：

SELECT id, name, gender, score FROM students ORDER BY score DESC, gender;

默认的排序规则是ASC：“升序”，即从小到大。ASC可以省略，即ORDER BY score ASC和ORDER BY score效果一样。

如果有WHERE子句，那么ORDER BY子句要放到WHERE子句后面。例如，查询一班的学生成绩，并按照倒序排序：

SELECT id, name, gender, score FROM students WHERE class\_id = 1 ORDER BY score DESC;

分页:

把结果集分页，每页3条记录。要获取第1页的记录，可以使用LIMIT 3 OFFSET 0：

SELECT id, name, gender, score FROM students ORDER BY score DESC LIMIT 3 OFFSET 0;//第一页

SELECT id, name, gender, score FROM students ORDER BY score DESC LIMIT 3 OFFSET 3;//第二页

SELECT id, name, gender, score FROM students ORDER BY score DESC LIMIT 3 OFFSET 6;//第三页

…

分页查询的关键在于，首先要确定每页需要显示的结果数量pageSize（这里是3），然后根据当前页的索引pageIndex（从1开始），确定LIMIT和OFFSET应该设定的值：

LIMIT总是设定为pageSize；

OFFSET计算公式为pageSize \* (pageIndex - 1)

注：OFFSET超过了查询的最大数量并不会报错，而是得到一个空的结果集。

OFFSET是可选的，如果只写LIMIT 15，那么相当于LIMIT 15 OFFSET 0。

在MySQL中，LIMIT 15 OFFSET 30还可以简写成LIMIT 30, 15。

使用LIMIT <M> OFFSET <N>分页时，随着N越来越大，查询效率也会越来越低。

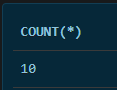
聚合查询:

对于统计总数、平均数这类计算，SQL提供了专门的聚合函数，使用聚合函数进行查询，就是聚合查询，它可以快速获得结果。

以查询students表一共有多少条记录为例，我们可以使用SQL内置的COUNT()函数查询：

SELECT COUNT(\*) FROM students;

COUNT(\*)表示查询所有列的行数，要注意聚合的计算结果虽然是一个数字，但查询的结果仍然是一个二维表，只是这个二维表只有一行一列，并且列名是COUNT(\*)。



COUNT(\*)和COUNT(id)实际上是一样的效果。

通常，使用聚合查询时，我们应该给列名设置一个别名，便于处理结果：

SELECT COUNT(\*) num FROM students;



聚合查询同样可以使用WHERE条件，因此我们可以方便地统计出有多少男生、多少女生、多少80分以上的学生等：

SELECT COUNT(\*) boys FROM students WHERE gender = 'M';



除了COUNT()函数外，SQL还提供了如下聚合函数：



要统计男生的平均成绩，我们用下面的聚合查询：

SELECT AVG(score) average FROM students WHERE gender = 'M';



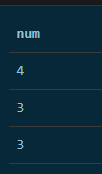
特别注意：如果聚合查询的WHERE条件没有匹配到任何行，COUNT()会返回0，而SUM()、AVG()、MAX()和MIN()会返回NULL：

SELECT AVG(score) average FROM students WHERE gender = 'X';



对于聚合查询，SQL还提供了“分组聚合”的功能。我们观察下面的聚合查询：

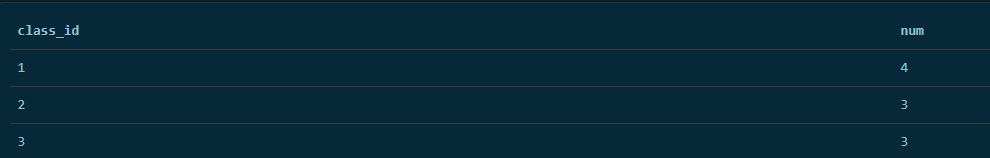
SELECT COUNT(\*) num FROM students GROUP BY class\_id;



执行这个查询，COUNT()的结果不再是一个，而是3个，这是因为，GROUP BY子句指定了按class\_id分组，因此，执行该SELECT语句时，会把class\_id相同的列先分组，再分别计算，因此，得到了3行结果。

但是这3行结果分别是哪三个班级的，不好看出来，所以我们可以把class\_id列也放入结果集中：

SELECT class\_id, COUNT(\*) num FROM students GROUP BY class\_id;



多表查询:

SELECT查询可以从一张表查询数据，还可以从多张表同时查询数据。语法是：SELECT \* FROM <表1> <表2>

例如，同时从students表和classes表的“乘积”，即查询数据，可以这么写：

SELECT \* FROM students, classes;

这种一次查询两个表的数据，查询的结果也是一个二维表，它是students表和classes表的“乘积”，即students表的每一行与classes表的每一行都两两拼在一起返回。结果集的列数是students表和classes表的列数之和，行数是students表和classes表的行数之积。

这种多表查询又称笛卡尔查询，使用笛卡尔查询时要非常小心，由于结果集是目标表的行数乘积，对两个各自有100行记录的表进行笛卡尔查询将返回1万条记录，对两个各自有1万行记录的表进行笛卡尔查询将返回1亿条记录。

上述查询的结果集有两列id和两列name，两列id是因为其中一列是students表的id，而另一列是classes表的id，但是在结果集中，不好区分。两列name同理

要解决这个问题，我们仍然可以利用投影查询的“设置列的别名”来给两个表各自的id和name列起别名：

SELECT students.id sid, students.name, students.gender, students.score, classes.id cid, classes.name cname FROM students, classes;

多表查询时，要使用表名.列名这样的方式来引用列和设置别名，这样就避免了结果集的列名重复问题。

用表名.列名这种方式列举两个表的所有列实在是很麻烦，所以SQL还允许给表设置一个别名，让我们在投影查询中引用起来稍微简洁一点：

SELECT s.id sid, s.name, s.gender, s.score, c.id cid, c.name cname FROM students s, classes c;

FROM子句给表设置别名的语法是FROM <表名1> <别名1>, <表名2> <别名2>

多表查询也是可以添加WHERE条件的:

SELECT

s.id sid,

s.name,

s.gender,

s.score,

c.id cid,

c.name cname

FROM students s, classes c

WHERE s.gender = 'M' AND c.id = 1;

连接查询:

连接查询是另一种类型的多表查询。连接查询对多个表进行JOIN运算，简单地说，就是先确定一个主表作为结果集，然后，把其他表的行有选择性地“连接”在主表结果集上。

最常用的一种内连接——INNER JOIN

SELECT s.id, s.name, s.class\_id, c.name class\_name, s.gender, s.score

FROM students s

INNER JOIN classes c

ON s.class\_id = c.id;

INNER JOIN查询的写法是：

先确定主表，仍然使用FROM <表1>的语法；

再确定需要连接的表，使用INNER JOIN <表2>的语法；

然后确定连接条件，使用ON <条件...>，这里的条件是s.class\_id = c.id，表示students表的class\_id列与classes表的id列相同的行需要连接；

可选：加上WHERE子句、ORDER BY等子句。

使用别名不是必须的，但可以更好地简化查询语句。

INNER JOIN只返回同时存在于两张表的行数据，由于students表的class\_id包含1，2，3，classes表的id包含1，2，3，4，所以，INNER JOIN根据条件s.class\_id = c.id返回的结果集仅包含1，2，3。

RIGHT OUTER JOIN返回右表都存在的行。如果某一行仅在右表存在，那么结果集就会以NULL填充剩下的字段。

LEFT OUTER JOIN则返回左表都存在的行。如果我们给students表增加一行，并添加class\_id=5，由于classes表并不存在id=5的行，所以，LEFT OUTER JOIN的结果会增加一行，对应的class\_name是NULL

FULL OUTER JOIN会把两张表的所有记录全部选择出来，并且，自动把对方不存在的列填充为NULL

关系数据库的基本操作就是增删改查，即CRUD：Create、Retrieve、Update、Delete。

向数据库表中插入一条新记录时，就必须使用INSERT语句

INSERT INTO <表名> (字段1, 字段2, ...) VALUES (值1, 值2, ...);

例如，向students表插入一条新记录，先列举出插入的字段名称，然后在VALUES子句中依次写出对应字段的值：

INSERT INTO students (class\_id, name, gender, score) VALUES (2, '大牛', 'M', 80);

字段顺序不必和数据库表的字段顺序一致，但值的顺序必须和字段顺序一致

还可以一次性添加多条记录，只需要在VALUES子句中指定多个记录值，每个记录是由(...)包含的一组值：

INSERT INTO students (class\_id, name, gender, score) VALUES

(1, '大宝', 'M', 87),

(2, '二宝', 'M', 81);

UPDATE语句用于更新数据库表中的记录

UPDATE <表名> SET 字段1=值1, 字段2=值2, ... WHERE ...;

例如，我们想更新students表id=1的记录的name和score这两个字段，先写出UPDATE students SET name='大牛', score=66，然后在WHERE子句中写出需要更新的行的筛选条件id=1：

UPDATE students SET name='大牛', score=66 WHERE id=1;

可以一次更新多条记录：

UPDATE students SET name='小牛', score=77 WHERE id>=5 AND id<=7;

在UPDATE语句中，更新字段时可以使用表达式。

例如，把所有80分以下的同学的成绩加10分：

UPDATE students SET score=score+10 WHERE score<80;

如果WHERE条件没有匹配到任何记录，UPDATE语句不会报错，也不会有任何记录被更新。

UPDATE语句可以没有WHERE条件，例如：UPDATE students SET score=60;这时，整个表的所有记录都会被更新。

DELETE语句用于删除数据库表中的记录

DELETE FROM <表名> WHERE ...;

例如，我们想删除students表中id=1的记录，就需要这么写：DELETE FROM students WHERE id=1;

DELETE语句的WHERE条件也是用来筛选需要删除的行，和UPDATE类似，DELETE语句也可以一次删除多条记录：

DELETE FROM students WHERE id>=5 AND id<=7;

如果WHERE条件没有匹配到任何记录，DELETE语句不会报错，也不会有任何记录被删除。

要特别小心的是，和UPDATE类似，不带WHERE条件的DELETE语句会删除整个表的数据：DELETE FROM students;