使用的SQL语句都运行在MySQL下，在其它数据库下可能会有所不同。

ROUND(column\_name, n)函数：将数值四舍五入为指定的小数位数。

n为正数时，column\_name的值就是正常的四舍五入为小数点后n位

n为0表示不保留小数

n为负值时，将会把column\_name小数点左边的值四舍五入，例如值为929.23，ROUND(929.23, -2)返回的值为900。

ROUND()函数在MySQL中返回的是一个BIGINT。

COUNT()函数：主要有三种形式，COUNT(column\_name)，COUNT(\*)，COUNT(DISTINCT column\_name)

COUNT(column\_name)：对指定列具有的行数进行统计，会自动剔除掉值为NULL的行，如果一列值全为NULL，结果返回0.

COUNT(\*)：对表中行的数目进行统计，包括值为NULL和重复行。如果表中所有记录都为NULL(表肯定没有主键)，增加一行NULL值，COUNT(\*)的计数不会增加，否则其它情况下，无论添加一行记录是否都为NULL，计数都会加1.

COUNT(DISTINCT column\_name)：返回指定列不同值的行数。

DATE\_FORMAT返回的是字符串格式，而其它的时间日期函数，如NOW()，CURDATE()，CURTIME()等返回的都是MySQL的时间日期格式。

计算日期差：

DATEDIFF()：DATEDIFF(时间1, 时间2)，时间1减去时间2的天数，MySQL默认计算的是天数。

如果希望计算其它类型的时间差，需要使用TIMESTAMPDIFF()。

TIMESTAMPDIFF(类型, 时间1, 时间2)：类型包括YEAR, MONTH, WEEK, DAY, HOUR等，计算的是时间2减去时间1，这和DATEDIFF是不同的。

sql非空约束NOT NULL

创建表时添加非空约束：

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255) NOT NULL,

Age int

);

给已创建的表的字段添加非空约束：

ALTER TABLE Persons

MODIFY Age int NOT NULL;

给已创建表的字段删除非空约束：

ALTER TABLE Persons

MODIFY Age int NULL;

NULL值与空串不是一回事。空串是一个有效值，是NOT NULL。

sql唯一约束UNIQUE：唯一约束允许为NULL，但NULL值只能有1条记录。

UNIQUE约束唯一标识数据库表中的每一条记录

UNIQUE和主键约束均为列提供了唯一性保证

每个表只能有一个主键约束，但可以有多个UNIQUE约束。

创建表时的UNIQUE约束：

CREATE TABLE Persons

{

id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

UNIQUE(id, LastName)

}

给已创建的表的列添加UNIQUE

ALTER TABLE Persons ADD UNIQUE(id)

删除唯一约束：

ALTER TABLE Persons DROP INDEX id

主键约束(PRIMARY KEY)：唯一标识数据库中的每条记录，并且字段值不能为NULL，同NOT NULL UNIQUE有相似之处，但也有不同：

1. 如果PRIMARY KEY和NOT NULL UNIQUE只定义了一列，则效果基本相同，但如果定义了多列，NOT NULL UNIQUE定义的每一列的值都是唯一的，但PRIMARY KEY只保证多列和是唯一的，某一列则有可能有重复值。
2. PRIMARY KEY可以和外键FOREIGN KEY联用，从而形成主从表的关系。

PRIMARY KEY一般在逻辑设计中用作记录标识，这也是设置PRIMARY KEY的本来用意，而UNIQUE只是为了保证域/域组的唯一性。

创建表时添加主键：

CREATE TABLE Persons

(

P\_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY(P\_Id)

)

多个列定义为PRIMARY KEY约束

CREATE TABLE Persons

(

P\_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY(P\_Id, LastName)

)

给已创建的表添加PRIMARY KEY约束：

ALTER TABLE Persons ADD PRIMARY KEY(P\_Id)

给已创建的表添加多列PRIMARY KEY约束：

ALTER TABLE Persons ADD CONSTRAINT pk\_PersonID PRIMARY KEY(P\_Id, LastName)

撤销PRIMARY KEY约束：

ALTER TABLE Persons DROP PRIMARY KEY

外键约束(FOREIGN KEY)：

CREATE TABLE Orders

(

O\_Id int NOT NULL,

OrderNo int NOT NULL,

P\_Id int,

PRIMARY KEY(O\_Id),

FOREIGN KEY (P\_Id) REFERENCES Persons

)

给已创建的表增加外键约束：

ALTER TABLE Orders ADD FOREIGN KEY(O\_Id) REFERENCES Persons(P\_Id)

检查约束(CHECK)：

CREATE TABLE courses

(

id int,

name varchar(255),

student\_count int,

teacher\_id int,

CHECK(student\_count > 0 AND teacher\_id > 0)

)

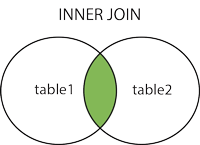
为已创建的表添加检查约束

ALTER TABLE courses ADD CHECK (student\_count > 0)

多表连接：

1. 内连接INNER JOIN：

内连接就是取两个表的交集，返回的结果就是连接的两张表中都满足条件的部分。



INNER JOIN在书写时可以省略INNER。

SELECT table1.column1, table2.colum2 …

FROM table1 JOIN table2 ON table1.common\_field1 = table2.common\_field2

lintcode 2062



左外连接：LEFT OUTER JOIN，其中OUTER可以省略。

在新关系中执行匹配条件时，以关键字LEFT JOIN左边的表为参考表，左外连接匹配的结果包含LFET JOIN指定的左表的所有行，而不仅仅是连接列匹配的行，加上右表中匹配到的行。

Lintcode 2051:

SELECT c.name AS course\_name, t.name AS teacher\_name FROM teachers t LEFT JOIN courses c ON c.teacher\_id = t.id WHERE t.country = 'CN'

右外连接：RIGHT JOIN

在新关系中执行匹配条件时，以关键字RIGHT JOIN右边的表尾参考表，如果右表的某行在左表中没有匹配值，就返回NULL。

Lintcode 2053:

SELECT c.name AS course\_name, t.name AS teacher\_name, t.email AS teacher\_email FROM courses c RIGHT JOIN teachers t ON c.teacher\_id = t.id WHERE t.country = 'CN'

全外连接(MySQL不支持全外连接)：FULL JOIN

只要左表和右表其中一个表存在匹配，则返回行。

由于MySQL不支持全外连接，所以需要使用UNION将左外连接和右外连接的结果结合起来。

UNION：两次查询的列数必须一致，并且每条SELECT语句中列的顺序必须相同。

SELECT column name1, column name2, … column name n

FROM table1 LEFT JOIN table2 ON condition

UNION

SELECT column name1, column name2, … column name n

FROM table1 RIGHT JOIN table2 ON condition

Lintcode 2055:

SELECT c.name AS course\_name, t.name AS teacher\_name, t.country AS teacher\_country FROM courses c LEFT JOIN teachers t ON c.teacher\_id = t.id

UNION

SELECT c.name AS course\_name, t.name AS teacher\_name, t.country AS teacher\_country FROM courses c RIGHT JOIN teachers t ON c.teacher\_id = t.id

交叉连接：CROSS JOIN

交叉连接没有ON子句，返回左表中的所有行，左表中的每一行都与右表中的所有行组合。查询的行数为左表中的行数乘以右表中的行数。

隐式交叉连接：

SELECT c.name AS course\_name, t.name AS teacher\_name FROM courses c, teachers t

显式交叉连接：

SELECT c.name AS course\_name, t.name AS teacher\_name FROM courses c CROSS JOIN teachers t

分组查询：

GROUP BY子句：实现分组的操作

例：统计不同国家教师的人数，并将结果按照不同国籍教师人数从小到大排列

SELECT country, COUNT(country) AS teacher\_count FROM teachers GROUP BY country ORDER BY teacher\_count, country desc

排序先按teacher\_count升序，再按country降序

Lintcode 2078：

SELECT age, COUNT(age) AS age\_count FROM teachers GROUP BY age ORDER BY age DESC

多表的GROUP BY：

统计每个教师教授课程的学生总数

SELECT t.name AS teacher\_name, IFNULL(SUM(c.student\_count), 0) AS student\_count FROM courses c RIGHT JOIN teachers t ON t.id = c.teacher\_id GROUP BY t.id

Lintcode 2082：

SELECT t.name AS teacher\_name, COUNT(c.teacher\_id) AS course\_count FROM courses c RIGHT JOIN teachers t ON t.id = c.teacher\_id GROUP BY t.id ORDER BY course\_count DESC, t.name ASC

HAVING 子句：WHERE子句和GROUP BY联用时，必须在GROUP BY之前，HAVING子句和GROUP BY联用时，必须在其后面。

HAVING子句和GROUP BY联用时，对分组的结果进行过滤。

例：统计不同教师所开课程的学生总数，对于没有任课的老师，学生总数计为 0 ，最后查询学生总数少于 3000 的教师姓名及学生总数 （别名为 student\_count ），结果按照学生总数升序排列，如果学生总数相同，则按照教师姓名升序排列

SELECT t.name AS name, IFNULL(SUM(c.student\_count), 0) AS student\_count FROM courses c RIGHT JOIN teachers t ON t.id = c.teacher\_id GROUP BY t.id HAVING student\_count < 3000 ORDER BY student\_count, t.name

Lintcode 2076：

SELECT \* FROM teachers WHERE country IN (SELECT country FROM teachers GROUP BY country HAVING AVG(age) > (SELECT AVG(age) FROM teachers))

子查询：

SELECT语句中的子查询：

例：查询Western Venom 老师所教的所有课程的所有信息

SELECT \* FROM courses WHERE teacher\_id = (SELECT id FROM teachers WHERE name = ‘Western Venom’)

Lintcode 2060：

SELECT name FROM teachers WHERE id = (SELECT teacher\_id FROM courses WHERE name = 'Big Data')

INSERT子句中的子查询：

例：将教师表 teachers 中的全部信息复制到相同表结构的备份表 teachers\_bkp 中

INSERT INTO teachers\_bkp SELECT \* FROM teachers

Lintcode 2056：

INSERT INTO teachers\_bkp SELECT \* FROM teachers WHERE age > 20

UPDATE语句中的子查询：

例：将Western Venom 创建的课程名称修改为 Java，

UPDATE courses SET name = ‘Java’ WHERE teacher\_id = (SELECT id FROM teachers WHERE name = ‘Western Venom’)

Lintcode 2057：

UPDATE courses SET name = 'PHP', student\_count = 300 WHERE teacher\_id = (SELECT id FROM teachers WHERE name = 'Eastern Heretic')

DELETE语句中的子查询：

例：删除课程表中所有教师年龄小于 21 岁（不包括 21 岁）的课程

DELETE FROM courses WHERE teacher\_id IN (SELECT id FROM teachers WHERE age < 21)

Lintcode 2059：

DELETE FROM teachers WHERE id IN(SELECT teacher\_id FROM courses WHERE created\_at < '2020-01-01')

内联视图子查询：

例：查询最年长且国籍是美国的教师信息

SELECT \* FROM (SELECT \* FROM teachers WHERE country = ‘USA’) AS T WHERE age = (SELECT MAX(age) FROM teachers)

Lintcode 2077：

SELECT \* FROM (SELECT c.name AS course\_name, c.student\_count, t.name AS teacher\_name FROM courses c JOIN teachers t ON t.id = c.teacher\_id) AS a WHERE student\_count = (SELECT MAX(student\_count) FROM courses)

IN操作符的子查询：

Lintcode 2065：查询所有年龄大于 20 岁的老师所教的所有课程的课程名

SELECT name FROM courses WHERE teacher\_id IN (SELECT t.id FROM teachers t WHERE t.age > 20)

ANY操作符的子查询：

例：查询学生上课人数超过 “Eastern Heretic” 的任意一门课的学生人数的课程信息

SELECT \* FROM courses WHERE student\_count > ANY (SELECT student\_count FROM courses WHERE teacher\_id = (SELECT id FROM teachers WHERE name = ‘Eastern Heretic’)) AND teacher\_id <> (SELECT id FROM teachers WHERE name = ‘Eastern Heretic’)

上述查询时，AND后面的条件是保证最终课程信息中没有Eastern Heretic的课程。

Lintcode 2070：

SELECT name FROM courses WHERE created\_at > ANY (SELECT created\_at FROM courses WHERE teacher\_id = (SELECT id FROM teachers WHERE name = 'Southern Emperor')) AND teacher\_id <> (SELECT id FROM teachers WHERE name = 'Southern Emperor')

ALL操作符的子查询：

例：查询学生人数超过 ”Western Venom“ 所有课程学生人数的课程信息

SELECT \* FROM courses WHERE student\_count > ALL (SELECT student\_count FROM courses WHERE teacher\_id = (SELECT id FROM teachers WHERE name = ‘Western Venom’))

Lintcode 2066：查询课程学生数超过最年长教师所有课程学生数的课程信息

SELECT \* FROM courses WHERE student\_count > ALL (SELECT student\_count FROM courses WHERE teacher\_id IN (SELECT id FROM teachers WHERE age IN (SELECT MAX(age) FROM teachers)))

多列子查询：

例：查找每个国家年龄最大的教师

SELECT name, age, country FROM teachers WHERE (country, age) IN (SELECT country, MAX(age) FROM teachers GROUP BY country)

Lintcode 2069：

SELECT name, student\_count FROM courses WHERE (teacher\_id, student\_count) IN (SELECT teacher\_id, MAX(student\_count) FROM courses GROUP BY teacher\_id)

HAVING语句中的子查询：

例：需要计算每位教师所开课程的平均学生人数与全部课程的平均学生人数，比较其大小，最后返回超过全部课程平均学生人数的教师姓名

SELECT name FROM teachers WHERE id IN (SELECT teacher\_id FROM courses GROUP BY teacher\_id HAVING AVG(student\_count) > (SELECT AVG(student\_count) FROM courses))

Lintcode 2086：

SELECT t.country AS country, SUM(c.student\_count) AS student\_count FROM courses c JOIN teachers t ON c.teacher\_id = t.id WHERE t.country LIKE 'U%' GROUP BY t.country HAVING student\_count BETWEEN 2000 AND 5000 ORDER BY student\_count DESC, country ASC

MySQL中的事务(transaction)：考虑一个银行应用，从一个银行的一个账户转账到另一个银行的另一个账户。我们需要更新这两个账户的余额，把需要转移的资金额从一个账户划走，并把它加到另一个账户上。如果在从第一个账户划走金额后，但在把这笔钱加到第二个账户前，系统发生崩溃，那么银行账户就会不一致。同理，如果在第一个账户划走资金之前先往第二个账户存款，然后马上系统崩溃，同样会出现银行账户不一致的问题。

为了避免上述的情况发生，需要使用事务操作(只有InnoDB引擎支持)。事务可以保证数据库的完整性，保证一组SQL语句要么全部执行完成，要么全部不执行，如果执行过程中出现类型系统崩溃的情况，就会进行回滚到初始状态。

事务一般管理insert，update，delete语句。

事务的4个特性：

1. 原子性：一个事务中的所有操作，要么全部完成，要么全部不完成。如果执行过程中发生错误，就会回滚到事务开始前的状态，就像这个事务没有执行过一样。
2. 一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。
3. 隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）
4. 持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失，即内存的数据持久化到了硬盘上。

开启事务：START TRANSACTION，或者BEGIN,BEGIN WORK

提交事务使其永久化，COMMIT

回滚：ROLLBACK

锁：

计算机协调多个进程或线程并发访问某一资源的机制。

锁的缺点：加锁消耗资源，包括获得锁、检测锁状态、释放锁都会消耗资源。