数据库——MySQL

目录

MySQL 教程

MYSQL 注释符号

- 单行注释

> #：从 # 开始到行尾的内容均为注释。
> --：ANSI标准注释符。

- 多行注释

> /**/：可跨越多行，注释内容需包裹在符号之间。

```sql
SELECT * FROM users; # 单行注释（MySQL特有）
SELECT * FROM users; --单行注释（ANSI标准）
/* 多行注释
可跨越多行 */
```

AI写代码sql

创建数据库

建数据库的基本语法如下：

```sql
CREATE DATABASE [IF NOT EXISTS] database_name
  [CHARACTER SET charset_name]
  [COLLATE collation_name];
```

AI写代码sql

如果希望在创建数据库时指定一些选项，可以使用 CREATE DATABASE 语句的其他参数，例如，你可以指定字符集和排序规则：

```sql
CREATE DATABASE mydatabase
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_general_ci;
```

AI写代码sql

如果数据库已经存在，执行 CREATE DATABASE 将导致错误。为了避免这种情况，你可以在 CREATE DATABASE 语句中添加 IF NOT EXISTS

```sql
 CREATE DATABASE IF NOT EXISTS mydatabase;
```

AI写代码sql

删除数据库

在 MySQL 中，要选择要使用的数据库，可以使用 drop 语法，命令格式：

```sql
DROP DATABASE <database_name>;        -- 直接删除数据库，不检查是否存在
或
DROP DATABASE [IF EXISTS] <database_name>;
```

AI写代码sql

参数说明：IF EXISTS 是一个可选的子句，表示如果数据库存在才执行删除操作，避免因为数据库不存在而引发错误。database_name 是你要删除的数据库的名称。

例如：删除 mydatabase 数据库。

```sql
-- 直接删除数据库，不检查是否存在
DROP DATABASE mydatabase;

-- 删除数据库，如果存在的话
DROP DATABASE IF EXISTS mydatabase;
```

AI写代码sql

选择数据库

在 MySQL 中，要选择要使用的数据库，可以使用 USE 语句，以下是基本的语法：

```sql
USE database_name;
```

AI写代码sql

database_name 是你要选择的数据库的名称。

数据类型

MySQL 中定义数据字段的类型对数据库的优化是非常重要的。MySQL 支持多种类型，大致可以分为三类：数值、日期/时间和字符串(字符)类型。

数值类型

MySQL 支持所有标准 SQL 数值数据类型。这些类型包括严格数值数据类型（INTEGER、SMALLINT、DECIMAL 和 NUMERIC），以及近似数值数据类型（FLOAT、REAL 和 DOUBLE PRECISION）。

关键字 INT 是 INTEGER 的同义词，关键字 DEC 是 DECIMAL 的同义词。BIT数据类型保存位字段值，并且支持 MyISAM、MEMORY、InnoDB 和 BDB表。作为 SQL 标准的扩展，MySQL 也支持整数类型 TINYINT、MEDIUMINT 和 BIGINT。下面的表显示了需要的每个整数类型的存储和范围。

类型	大小（bytes）	用途
TINYINT	1	小整数值
SMALLINT	2	大整数值
MEDIUMINT	3	大整数值
INT或INTEGER	4	大整数值
BIGINT	8	极大整数值
FLOAT	4	单精度浮点数值
DOUBLE	8	双精度浮点数值
DECIMAL(M, N)	M 是最大位数（精度），范围是 1 到 65。可不指定，默认值是 10。 D 是小数点右边的位数（小数位）。范围是 0 到 30，并且不能大于 M，可不指定，默认值是 0。	小数值

日期和时间类型

表示时间值的日期和时间类型为 DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。每个时间类型有一个有效值范围和一个"零"值，当指定不合法的 MySQL 不能表示的值时使用"零"值。TIMESTAMP 类型有专有的自动更新特性，将在后面描述。

类型	大小（bytes）	范围	格式	用途
DATE	3	1000-01-01~9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'~'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901~2155	YYYY	年份值
DATETIME	8	'1000-01-01 00:00:00' ~ '9999-12-31 23:59:59'	YYYY-MM-DD hh:mm:ss	混合日期和时间值
TIMESTAMP	4	'1970-01-01 00:00:01' UTC 到 '2038-01-19 03:14:07' UTC 结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038年1月19日凌晨 03:14:07	YYYY-MM-DD hh:mm:ss	混合日期和时间值，时间戳

字符串类型

字符串类型指 CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM 和 SET。该节描述了这些类型如何工作以及如何在查询中使用这些类型。

类型	大小（bytes）	用途
CHAR	0 - 255	定长字符串
VARCHAR	0 - 65535	变长字符串
TINYBLOB	0 - 255	不超过 255 个字符的二进制字符串
TINYTEXT	0 - 255	短文本字符串
BLOB	0 - 65535	二进制形式的长文本数据
TEXT	0 - 65535	长文本数据
MEDIUMBLOB	0 - 16777215	二进制形式的中等长度文本数据

类型	大小（bytes）	用途
MEDIUMTEXT	0 - 16777215	中等长度文本数据
LONGBLOB	0 - 4294967295	二进制形式的极大文本数据
LONGTEXT	0 - 4294967295	极大文本数据

注意：char(n) 和 varchar(n) 中括号中 n 代表字符的个数，并不代表字节个数，比如 CHAR(30) 就可以存储 30 个字符。CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。BINARY 和 VARBINARY 类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。BLOB 是一个二进制大对象，可以容纳可变数量的数据。有 4 种 BLOB 类型：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。它们区别在于可容纳存储范围不同。有 4 种 TEXT 类型：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。对应的这 4 种 BLOB 类型，可存储的最大长度不同，可根据实际情况选择。

枚举和集合类型

ENUM：枚举类型，用于存储单一值，可以选择一个预定义的集合。

SET：集合类型，用于存储多个值，可以选择多个预定义的集合。

空间数据类型

GEOMETRY，POINT，LINESTRING，POLYGON，MULTIPOINT，MULTILINESTRING，MULTIPOLYGON，GEOMETRYCOLLECTION：用于存储空间数据（地理信息、几何图形等）。

创建数据表

创建 MySQL 数据表需要以下信息：表名，表字段名，定义每个表字段的数据类型。

```sql
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);
```

AI写代码sql

参数说明：table_name 是你要创建的表的名称。column1，column2...是表中的列名。datatype 是每个列的数据类型。

以下是一个具体的实例，创建一个用户表 users：

```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    birthdate DATE,
    is_active BOOLEAN DEFAULT TRUE
);
```

AI写代码sql

实例解析：用户 id，整数类型，自增长，作为主键。用户名 username，变长字符串，不允许为空。用户邮箱 email，变长字符串，不允许为空。用户的生日 birthdate，日期类型。用户是否已经激活 is_active，布尔类型，默认值为 true。以上只是一个简单的实例，用到了一些常见的数据类型包括 INT，VARCHAR，DATE，BOOLEAN，可以根据实际需要选择不同的数据类型。AUTO_INCREMENT 关键字用于创建一个自增长的列，PRIMARY KEY 用于定义主键。

如果希望在创建表时指定数据引擎，字符集和排序规则等，可以使用 CHARACTER SET 和 COLLATE 子句：

```sql
CREATE TABLE mytable (
    id INT PRIMARY KEY,
    name VARCHAR(50)
) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

AI写代码sql

以上代码创建一个使用 utf8mb4 字符集和 utf8mb4_general_ci 排序规则的表。

删除数据表

MySQL中删除数据表是非常容易操作的，但是在进行删除表操作时要非常小心，因为执行删除命令后所有数据都会消失。

以下为删除 MySQL 数据表的通用语法：

```sql
DROP TABLE table_name;      -- 直接删除表，不检查是否存在
或
DROP TABLE [IF EXISTS] table_name;  -- 会检查是否存在，如果存在则删除
```

AI写代码sql

参数说明：table_name 是要删除的表的名称。IF EXISTS 是一个可选的子句，表示如果表存在才执行删除操作，避免因为表不存在而引发错误。

如果你只是想删除表中的所有数据，但保留表的结构，可以使用 TRUNCATE TABLE 语句：

```sql
TRUNCATE TABLE table_name;
```

AI写代码sql

这会清空表中的所有数据，但不会删除表本身。

注意事项：备份数据：在删除表之前，确保已经备份了数据，如果你需要的话。外键约束：如果该表与其他表有外键约束，可能需要先删除外键约束，或者确保依赖关系被处理好。

插入数据

MySQL 表中使用 INSERT INTO 语句来插入数据。以下为向 MySQL 数据表插入数据通用的 INSERT INTO SQL语法：

```sql
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

AI写代码sql

参数说明：table_name 是要插入数据的表的名称。column 是表中的列名。value 是要插入的具体数值。如果数据是字符型，必须使用单引号 ' 或者双引号 "，如：'value1', "value1"。一个简单的实例，插入了一行数据到名为 users 的表中：

```sql
INSERT INTO users (username, email, birthdate, is_active)
VALUES ('test', 'test@qq.com', '1990-01-01', true);
```

AI写代码sql

用户名 username，字符串类型。邮箱地址 email，字符串类型。用户生日 birthdate， 日期类型。是否已激活 is_active，布尔类型。

如果要插入所有列的数据，可以省略列名：

```sql
INSERT INTO users
VALUES (NULL,'test', 'test@qq.com', '1990-01-01', true);
```

AI写代码sql

这里，NULL 是用于自增长列的占位符，表示系统将为 id 列生成一个唯一的值。如果你要插入多行数据，可以在 VALUES 子句中指定多组数值：

```sql
INSERT INTO users (username, email, birthdate, is_active)
VALUES
    ('test1', 'test1@qq.com', '1985-07-10', true),
    ('test2', 'test2@qq.com', '1988-11-25', false),
    ('test3', 'test3@qq.com', '1993-05-03', true);
```

AI写代码sql

以上代码将在 users 表中插入三行数据。

查询数据

单表查询

MySQL 数据库使用 SELECT 语句来查询数据。以下为在 MySQL 数据库中查询数据通用的 SELECT 语法：

```sql
SELECT column1, column2, ...
FROM table_name
[WHERE condition]
[ORDER BY column_name [ASC | DESC]]
[LIMIT number];
```

AI写代码sql

参数说明：column1, column2, ... 是想要选择的列的名称，如果使用 * 表示选择所有列。table_name 是要从中查询数据的表的名称。WHERE condition 是一个可选的子句，用于指定过滤条件，只返回符合条件的行。ORDER BY column_name [ASC | DESC] 是一个可选的子句，用于指定结果集的排序顺序，默认是升序（ASC）。LIMIT number 是一个可选的子句，用于限制返回的行数。

MySQL SELECT 语句简单的应用实例：

```sql
-- 选择所有列的所有行
SELECT * FROM users;

-- 选择特定列的所有行
SELECT username, email FROM users;

-- 添加 WHERE 子句，选择满足条件的行
SELECT * FROM users WHERE is_active = TRUE;

-- 添加 ORDER BY 子句，按照某列的升序排序
SELECT * FROM users ORDER BY birthdate;

-- 添加 ORDER BY 子句，按照某列的降序排序
SELECT * FROM users ORDER BY birthdate DESC;

-- 添加 LIMIT 子句，限制返回的行数
SELECT * FROM users LIMIT 10;
```

AI写代码sql

SELECT 可以根据实际需求组合和使用这些子句，比如同时使用 WHERE 和 ORDER BY 子句，或者使用 LIMIT 控制返回的行数。在 Where 子句中，可以使用各种条件运算符（如 =，>，<，<=，>=，!=），逻辑运算符（如 AND，OR，NOT），以及通配符（如 %）等。

以下是一些进阶的 SELECT 语句实例：

```sql
-- 使用 AND 运算符和通配符
SELECT * FROM users WHERE username LIKE 'j%' AND is_active = TRUE;

-- 使用 OR 运算符
SELECT * FROM users WHERE is_active = TRUE OR birthdate < '1990-01-01';

-- 使用 IN 子句
```

```sql
SELECT * FROM users WHERE birthdate IN ('1990-01-01', '1992-03-15', '1993-05-03');
```

AI写代码sql

去重查询

去重查询是通过关键字 distinct 来实现的

```sql
select distinct column1, column2,...
from table_name;
```

AI写代码sql

多表查询

联合查询就是联合多个表进行查询，为了消除表中字段的依赖关系，设计数据时把表进行拆分，这时就会导致一条 SQL 语句查找出来的数据不够完整，就可以通过联合查询把关系中的数据全部查出来，在一个数据行中显示详细信息。步骤：首先确定哪几张表要参与查询；根据表于表之间的主外键关系，确定过滤条件；精简查询字段。

- 内连接

```sql
select column1,...
from table1 [other_name], table2 [other_name],...
where condition;

或

select column1,...
from table1 [other_name]
[inner] join table2 [other_name] on condition
[inner] join table3 [other_name] on condition
        ...;
```

AI写代码sql

参数说明：column1, ... 是想要选择的列的名称，如果使用 * 表示选择所有列。table 是要从中查询数据的表的名称。other_name 是对 table 进行命名为其他名字。WHERE condition 用于指定过滤条件，只返回符合条件的行。inner 表示内连接，通过关键字 inner join 来实现的，from table1 [other_name] [inner] join table2 [other_name] 意思是从表 1 到表 2 建立内连接，关注点是表 1，从表 1 里面找表 2 的连接，inner 可以省略。

例子：

```sql
-- 1. 无选择条件
select * from student,class;
-- 可能导致有数据是不匹配的

-- 2.通过连接条件过滤掉无效的数据
select *
from student, class
where student.class_id = class.id;

-- 3.通过指定列查询精简结果集
select student.id, student.name, class.name
from student, class
where student.class_id = class.id;

-- 4.加入别名的方式进行简化SQL语句
select s.id, s.name, c.name
from student s, class c
where s.class_id = c.id;

或

select s.id, s.name, c.name
from student s inner join class c on class_id = c.id;

-- 5.多个表内连接
select st.student_id, st.name, c.name, sc.score
from student st
    join score sc on st.student_id = sc.student_id
    join course c on c.course_id = sc.course_id;
```

AI写代码sql

- 外连接

内连接和外连接的区别：内连接只会查询到两个表的交集部分，外连接可以查询左边或右边整个表。
左外连接：以左表为基准，无论右表是否存在匹配记录，左表数据都会被完整保留。
右外连接：以右表为基准，无论左表是否存在匹配记录，右表数据都会被完整保留。
左外连接的右表无匹配字段显示为 NULL，右外连接的左表无匹配字段显示为 NULL。保留侧表的原始数据不受另一表匹配情况影响，确保数据完整性。

```sql
-- 左外连接
select column1,...
from table1 left [outer] join table2 on condition;

-- 右外连接
select column1,...
from table1 right [outer] join table2 on condition;
```

AI写代码sql

A 左连接 B 等价于 B 右连接 A，这种转换使得实际开发中右外连接使用频率较低，多数情况下可用左外连接替代。

```sql
-- 使用右外连接
select student.student_id, student.name, class.class_id, class.name
from student right join class on student.class_id = class.class_id;
-- 从 student 表到 class 表建立右外连接
-- 没有学生的班级id 也会显示出来，用 null 代替
-- 右边 class 表中的数据都显示出来了，左边没有与之对应的行用 null 补充

-- 左外连接
select student.student_id, student.name, score
from student left join score on student.student_id = score.student_id;
-- 查询哪位同学没有参加考试
-- 也就是在 student 表里有记录，在 score 表里没有对应的记录
```

```sql
-- 这时就可以使用左连接，把student表作为基准表
```

AI写代码sql

- 自连接

> 自连接可以实现行与行之间的比较功能
>
> ```sql
> select column1,...
> from table [other_name1] join table [other_name2] on condition;
> ```
>
> AI写代码sql
>
> 例子：
>
> ```sql
> -- 找出计算机原理的成绩大于 Java 的
> select *
> from score s1, score s2
> where s1.student_id = s2.student_id
> and s1.jz_score > s2.java_score;
>
> 或
>
> select *
> from score s1 join score s2
>     on s1.student_id = s2.student_id
>     and s1.jz_score > s2.java_score;
> ```
>
> AI写代码sql
>
> 
>
> 若表是如下：

id	name	job	manager_id
1	张三	总裁	null
2	李四	项目经理	1
3	王五	工程师	2

```sql
-- 查询员工及其所属领导的名字
select a.name, b.name
from emp a, emp b
where a.manager_id = b.id;

-- 查询所有员工及其领导的名字（如果员工没有领导也要表示出来）
select a.name, b.name
from emp a left join emp b
on a.manager_id = b.id;
```

AI写代码sql

- 子查询

SQL语句中嵌套 select 语句称为嵌套查询，又叫子查询，语法如下：

```sql
select column1,...
from t1
where column1 = (
    select columnm1
    from t2
    where...);
```

AI写代码sql

单行子查询例子如下：

```sql
-- 查询与许仙一个班级的同学
-- 如果不适用子查询的话，就需要用多条SQL语句来查询

-- 查找和许仙一个班级的同学
select class_id from student where name = '许仙';
-- 得出许仙的 class_id 是 1
select name from student where class_id = 1;

-- 子查询的形式
select name
from student
where class_id = (
    select class_id
    from student
    where name = '许仙');

-- 也可以对子查询的整体加上再加上过滤条件

select name
from student
where class_id = (
    select class_id
    from student
    where name = '许仙') and name != '许仙';
--要注意的是外层条件的列，与同层查询条件的列必须要匹配
```

AI写代码sql

多行子查询例子如下：

```sql
-- 获取语文和英文的成绩信息
-- 如果不用子查询，还是需要两步进行
-- 获取课程 id
select course_id
```

```
from course
            where name = '语文'
   or name = '英文';

-- 再根据 id 去查找成绩信息
-- m 为上面语文的 id
-- n 为上面英文的 id
select *
from score
where course_id = m
   or course_id = n;

-- 由于这里查询到的是多行的信息
-- 所以在进行子查询的时候需要使用 IN（） 来判断
select *
from score
where course_id in (
    select course_id
    from course
    where name = '语文'
        or name = '英文');
```

AI写代码sql

多列子查询例子如下：

```
-- 查询重复的分数
-- 按照同一个学生，同一门课程，同样的成绩这三个列同时去分组
-- 然后分组之后在 having 字句中用 count(*) 判断分组中的记录数
select student_id, course_id, score
from score
group by student_id, course_id, score;

-- 加上having过滤条件
select student_id, course_id, score, count(*)
from score
```

```sql
group by student_id, course_id, score having count(*) > 1;

-- 用多列分组查询
select *
from score
where (student_id, course_id, score) in (
    select student_id, course_id, score
    from score
    group by student_id, course_id, score
    having count(*) > 1);
```

AI写代码sql

合并查询

Where 子句

从 MySQL 表中使用 SELECT 语句来读取数据。如需有条件地从表中选取数据，可将 WHERE 子句添加到 SELECT 语句中。WHERE 子句用于在 MySQL 中过滤查询结果，只返回满足特定条件的行。以下是 SQL SELECT 语句使用 WHERE 子句从数据表中读取数据的通用语法：

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

AI写代码sql

参数说明：column1, column2, ... 是想要选择的列的名称，如果使用 * 表示选择所有列。table_name 是要从中查询数据的表的名称。WHERE condition 是一个可选的子句，用于指定过滤条件，只返回符合条件的行。

更多说明：查询语句中可以使用一个或者多个表，表之间使用逗号分割，并使用 WHERE 语句来设定查询条件。可以在 WHERE 子句中指定任何条件。可以使用 AND 或者 OR 指定一个或多个条件。WHERE 子句也可以运用于 SQL 的 DELETE 或者 UPDATE 命令。WHERE 子句类似于程序语言中的 if 条件，根据 MySQL 表中的字段值来读取指定的数据。

以下为部分比较运算符列表，可用于 WHERE 子句中。下表中实例假定 A 为 10, B 为 20：

操作符	描述	实例
=	等号，检测两个值是否相等，如果相等返回 true	(A = B) 返回false。
<>, !=	不等于，检测两个值是否相等，如果不相等返回 true	(A != B) 返回 true。
>	大于号，检测左边的值是否大于右边的值, 如果左边的值大于右边的值返回 true	(A > B) 返回false。
<	小于号，检测左边的值是否小于右边的值, 如果左边的值小于右边的值返回 true	(A < B) 返回 true。
>=	大于等于号，检测左边的值是否大于或等于右边的值, 如果左边的值大于或等于右边的值返回 true	(A >= B) 返回false。
<=	小于等于号，检测左边的值是否小于或等于右边的值, 如果左边的值小于或等于右边的值返回 true	(A <= B) 返回 true。

部分操作如下：

```
-- 1. 等于条件：
SELECT * FROM users WHERE username = 'test';

-- 2. 不等于条件：
SELECT * FROM users WHERE username != 'runoob';

-- 3. 大于条件：
SELECT * FROM products WHERE price > 50.00;

-- 4. 小于条件：
SELECT * FROM orders WHERE order_date < '2023-01-01';

-- 5. 大于等于条件：
SELECT * FROM employees WHERE salary >= 50000;

```

```sql
-- 6. 小于等于条件:
SELECT * FROM students WHERE age <= 21;

-- 7. 组合条件（AND、OR）:
SELECT * FROM products WHERE category = 'Electronics' AND price > 100.00;

SELECT * FROM orders WHERE order_date >= '2023-01-01' OR total_amount > 1000.00;

-- 8. 模糊匹配条件（LIKE）:
SELECT * FROM customers WHERE first_name LIKE 'J%';

-- 9. IN 条件:
SELECT * FROM countries WHERE country_code IN ('US', 'CA', 'MX');

-- 10. NOT 条件:
SELECT * FROM products WHERE NOT category = 'Clothing';

-- 11. BETWEEN 条件:
SELECT * FROM orders WHERE order_date BETWEEN '2023-01-01' AND '2023-12-31';

-- 12. IS NULL 条件
SELECT * FROM employees WHERE department IS NULL;

-- 13. IS NOT NULL 条件:
SELECT * FROM customers WHERE email IS NOT NULL;
```

AI写代码sql

UPDATE 更新

如果我们需要修改或更新 MySQL 中的数据，可以使用 UPDATE 命令来操作。以下是 UPDATE 命令修改 MySQL 数据表数据的通用 SQL 语法：

```sql
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

AI写代码sql

参数说明：table_name 是要更新数据的表的名称。column 是要更新的列的名称。value 是新的值，用于替换旧的值。WHERE condition 是一个可选的子句，用于指定更新的行。如果省略 WHERE 子句，将更新表中的所有行。

更多说明：可以同时更新一个或多个字段。可以在 WHERE 子句中指定任何条件。可以在一个单独表中同时更新数据。当需要更新数据表中指定行的数据时 WHERE 子句是非常有用的。

下面实例演示了如何使用 UPDATE 语句：

```sql
-- 1. 更新单个列的值：
UPDATE employees
SET salary = 60000
WHERE employee_id = 101;

-- 2. 更新多个列的值：
UPDATE orders
SET status = 'Shipped', ship_date = '2023-03-01'
WHERE order_id = 1001;

-- 3. 使用表达式更新值：
UPDATE products
SET price = price * 1.1
WHERE category = 'Electronics';
-- 以上 SQL 语句将每个属于 'Electronics' 类别的产品的价格都增加了 10%。

-- 4. 更新符合条件的所有行：
UPDATE students
SET status = 'Graduated';
-- 以上 SQL 语句将所有学生的状态更新为 'Graduated'。

-- 5. 更新使用子查询的值：
UPDATE customers
SET total_purchases = (
    SELECT SUM(amount)
    FROM orders
```

```
    WHERE orders.customer_id = customers.customer_id
)
WHERE customer_type = 'Premium';
-- 以上 SQL 语句通过子查询计算每个 'Premium' 类型客户的总购买金额，
-- 并将该值更新到 total_purchases 列中。
```

AI写代码sql

在使用 UPDATE 语句时，确保提供了足够的条件来确保只有想要更新的行被修改。如果不提供 WHERE 子句，将更新表中的所有行，可能导致不可预测的结果。

DELETE 语句

可以使用 DELETE FROM 命令来删除 MySQL 数据表中的记录。以下是 DELETE 语句从 MySQL 数据表中删除数据的通用语法：

```
DELETE FROM table_name
WHERE condition;
```

AI写代码sql

参数说明：table_name 是要删除数据的表的名称。WHERE condition 是一个可选的子句，用于指定删除的行。如果省略 WHERE 子句，将删除表中的所有行。

更多说明：如果没有指定 WHERE 子句，MySQL 表中的所有记录将被删除。可以在 WHERE 子句中指定任何条件。可以在单个表中一次性删除记录。

以下实例演示了如何使用 DELETE 语句：

```
-- 1. 删除符合条件的行：
DELETE FROM students
WHERE graduation_year = 2021;
-- 以上 SQL 语句删除了 students 表中所有 graduation_year 为 2021 的学生的记录。

-- 2. 删除所有行：
DELETE FROM orders;
-- 以上 SQL 语句删除了 orders 表中的所有记录，但表结构保持不变。

-- 3. 使用子查询删除符合条件的行：
```

```sql
DELETE FROM customers12 WHERE customer_id IN (
    SELECT customer_id
    FROM orders
    WHERE order_date < '2023-01-01'
);
-- 以上 SQL 语句通过子查询删除了 orders 表中在 '2023-01-01' 之前下的订单对应的客户。
```

AI写代码sql

LIKE 子句

我们可以在 SELECT 语句中使用 WHERE 子句来获取指定的记录。WHERE 子句中可以使用等号 = 来设定获取数据的条件，如 "author = 'PP.COM'"。但是有时候我们需要获取 author 字段含有 "COM" 字符的所有记录，这时我们就需要在 WHERE 子句中使用 LIKE 子句。LIKE 子句是在 MySQL 中用于在 WHERE 子句中进行模糊匹配的关键字。它通常与通配符一起使用，用于搜索符合某种模式的字符串。LIKE 子句中使用百分号 % 字符来表示任意字符，类似于 UNIX 或正则表达式中的星号 *。如果没有使用百分号 %，LIKE 子句与等号 = 的效果是一样的。以下是 SQL SELECT 语句使用 LIKE 子句从数据表中读取数据的通用语法：

```sql
SELECT column1, column2, ...
FROM table_name
WHERE column_name LIKE pattern;
```

AI写代码sql

参数说明：column1，column2，... 是想要选择的列的名称，如果使用 * 表示选择所有列。table_name 是要从中查询数据的表的名称。column_name 是要应用 like 子句的列的名称。pattern 是用于匹配的模式，可以包含通配符。

更多说明：可以在 WHERE 子句中指定任何条件。可以在 WHERE 子句中使用 LIKE 子句。可以使用 LIKE 子句代替等号 =。LIKE 通常与 % 一同使用，类似于一个元字符的搜索。可以使用 AND 或者 OR 指定一个或多个条件。可以在 DELETE 或 UPDATE 命令中使用 WHERE...LIKE 子句来指定条件。

以下是一些 LIKE 子句的使用实例：

```sql
-- 1. 百分号通配符 %：
-- % 通配符表示零个或多个字符。例如，'a%' 匹配以字母 'a' 开头的任何字符串。
SELECT *
FROM customers
WHERE last_name LIKE 'S%';
```

```sql
-- 以上 SQL 语句将选择所有姓氏以 'S' 开头的客户。

-- 2. 下划线通配符 _：
-- _ 通配符表示一个字符。例如，'_r%' 匹配第二个字母为 'r' 的任何字符串。
SELECT *
FROM products
WHERE product_name LIKE '_a%';
-- 以上 SQL 语句将选择产品名称的第二个字符为 'a' 的所有产品。

-- 3. 组合使用 % 和 _：
SELECT *
FROM users
WHERE username LIKE 'a%o_';
-- 以上 SQL 语句将匹配以字母 'a' 开头，然后是零个或多个字符，
-- 接着是 'o'，最后是一个任意字符的字符串，如 'aaron'、'apol'。
```

AI写代码sql

LIKE 子句提供了强大的模糊搜索能力，可以根据不同的模式和需求进行定制。在使用时，请确保理解通配符的含义，并根据实际情况进行匹配。

UNION

MySQL UNION 操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合，并去除重复的行。UNION 操作符必须由两个或多个 SELECT 语句组成，每个 SELECT 语句的列数和对应位置的数据类型必须相同。MySQL UNION 操作符语法格式：

```sql
SELECT column1, column2, ...
FROM table1
WHERE condition1
UNION
SELECT column1, column2, ...
FROM table2
WHERE condition2
[ORDER BY column1, column2, ...];
```

AI写代码sql

参数说明：column1，column2，... 是想要选择的列的名称。table1，table2，... 是要从中查询数据的表的名称。condition1，condition2，...是每个 SELECT 语句的过滤条件，是可选的。ORDER BY 子句是一个可选的子句，用于指定合并后的结果集的排序顺序。

以下是一些 MySQL UNION 的使用实例：

```sql
-- 1. 基本的 UNION 操作：
SELECT city FROM customers
UNION
SELECT city FROM suppliers
ORDER BY city;
-- 以上 SQL 语句将选择客户表和供应商表中所有城市的唯一值，并按城市名称升序排序。

-- 2. 使用过滤条件的 UNION：
SELECT product_name FROM products WHERE category = 'Electronics'
UNION
SELECT product_name FROM products WHERE category = 'Clothing'
ORDER BY product_name;
-- 以上 SQL 语句将选择电子产品和服装类别的产品名称，并按产品名称升序排序。

-- 3. UNION 操作中的列数和数据类型必须相同：
SELECT first_name, last_name FROM employees
UNION
SELECT department_name, NULL FROM departments
ORDER BY first_name;
-- 返回的结果包括了 employees 表中的 first_name 和 last_name，
-- 以及 departments 表中的 department_name 和 NULL，
-- 所有的结果都按照 first_name 列排序。

-- 4. 使用 UNION ALL 不去除重复行：
SELECT city FROM customers
UNION ALL
SELECT city FROM suppliers
ORDER BY city;
-- 以上 SQL 语句使用 UNION ALL 将客户表和供应商表中的所有城市合并在一起，不去除重复行。
```

UNION 操作符在合并结果集时会去除重复行，而 UNION ALL 不会去除重复行，因此 UNION ALL 的性能可能更好，但如果希望去除重复行，可以使用 UNION。

ORDER BY 语句

从 MySQL 表中使用 SELECT 语句来读取数据。如果需要对读取的数据进行排序，就可以使用 MySQL 的 ORDER BY 子句来设定想按哪个字段哪种方式来进行排序，再返回搜索结果。MySQL ORDER BY（排序）语句可以按照一个或多个列的值进行升序（ASC）或降序（DESC）排序。

```sql
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], ...;
```

AI写代码sql

参数说明：column1，column2，... 是想要选择的列的名称。table_name 是要从中查询数据的表的名称。ORDER BY 子句是一个可选的子句，用于指定合并后的结果集的排序顺序。

更多说明：可以使用任何字段来作为排序的条件，从而返回排序后的查询结果。可以设定多个字段来排序。可以使用 ASC 或 DESC 关键字来设置查询结果是按升序或降序排列。 默认情况下，它是按升序排列。可以添加 WHERE...LIKE 子句来设置条件。

以下是一些 ORDER BY 子句的使用实例：

```sql
-- 1. 单列排序：
SELECT * FROM products
ORDER BY product_name ASC;
-- 以上 SQL 语句将选择产品表 products 中的所有产品，
-- 并按产品名称升序 ASC 排序。

-- 2. 多列排序：
SELECT * FROM employees
ORDER BY department_id ASC, hire_date DESC;
-- 以上 SQL 语句将选择员工表 employees 中的所有员工，
-- 并先按部门 ID 升序 ASC 排序，
-- 然后在相同部门中按雇佣日期降序 DESC 排序。
```

```sql

-- 3. 使用数字表示列的位置：
SELECT first_name, last_name, salary
FROM employees
ORDER BY 3 DESC, 1 ASC;
-- 以上 SQL 语句将选择员工表 employees 中的名字和工资列，
-- 并按第三列（salary）降序 DESC 排序，
-- 然后按第一列（first_name）升序 ASC 排序。

-- 4. 使用表达式排序：
SELECT product_name, price * discount_rate AS discounted_price
FROM products
ORDER BY discounted_price DESC;
-- 以上 SQL 语句将选择产品表 products 中的产品名称和根据折扣率计算的折扣后价格，
-- 并按折扣后价格降序 DESC 排序。
-- AS 是取别名的意思

-- 5.从 MySQL 8.0.16 版本开始，可以使用 NULLS FIRST 或 NULLS LAST 处理 NULL 值：
SELECT product_name, price
FROM products
ORDER BY price DESC NULLS LAST;
-- 以上 SQL 语句将选择产品表 products 中的产品名称和价格，
-- 并按价格降序 DESC 排序，将 NULL 值排在最后。

-- 相反，如果你想让 NULL 值排在前面，可以这样写：
SELECT product_name, price
FROM products
ORDER BY price DESC NULLS FIRST;
```

AI写代码sql

ORDER BY 子句是一个强大的工具，可以根据不同的业务需求对查询结果进行排序。在实际应用中，注意选择适当的列和排序顺序，以获得符合期望的排序效果。

分组 GROUP BY 语句

GROUP BY 语句根据一个或多个列对结果集进行分组。在分组的列上我们可以使用 COUNT，SUM，AVG 等函数。GROUP BY 语句是 SQL 查询中用于汇总和分析数据的重要工具，尤其在处理大量数据时，它能够提供有用的汇总信息。GROUP BY 语法：

```sql
SELECT column1, aggregate_function(column2)
FROM table_name
WHERE condition
GROUP BY column1;
```

AI写代码sql

参数说明：column1 是想要选择的列的名称。aggregate_function(column2) 对分组后的每个组执行的聚合函数。table_name 是要从中查询数据的表的名称。condition 可选，用于筛选结果的条件。

假设有一个名为 orders 的表，包含以下列：order_id、customer_id、order_date 和 order_amount。我们想要按照 customer_id 进行分组，并计算每个客户的订单总金额，SQL 语句如下：

```sql
SELECT customer_id, SUM(order_amount) AS total_amount
FROM orders
GROUP BY customer_id;
```

AI写代码sql

以上实例中，我们使用 GROUP BY customer_id 将结果按 customer_id 列分组，然后使用 SUM(order_amount) 计算每个组中 order_amount 列的总和。AS total_amount 是为了给计算结果取一个别名，使查询结果更易读。

注意事项：GROUP BY 子句通常与聚合函数一起使用，因为分组后需要对每个组进行聚合操作。SELECT 子句中的列通常要么是分组列，要么是聚合函数的参数。可以使用多个列进行分组，只需在 GROUP BY 子句中用逗号分隔列名即可。

- WITH GRLLUP

> WITH ROLLUP 可以实现在分组统计数据基础上再进行相同的统计（SUM，AVG，COUNT 等）。
> 例如我们将以上的数据表按名字进行分组，再统计每个人登录的次数：
>
> ```sql
> SELECT name, SUM(signin) as signin_count
> FROM  employee_tbl
> ```

```
3 GROUP BY name WITH ROLLUP;
```

AI写代码sql

假设表如下：

```
+----+--------+---------------------+--------+
| id | name   | date                | signin |
+----+--------+---------------------+--------+
|  1 | 小明 | 2016-04-22 15:25:33 |      1 |
|  2 | 小王 | 2016-04-20 15:25:47 |      3 |
|  3 | 小丽 | 2016-04-19 15:26:02 |      2 |
|  4 | 小王 | 2016-04-07 15:26:14 |      4 |
|  5 | 小明 | 2016-04-11 15:26:40 |      4 |
|  6 | 小明 | 2016-04-04 15:26:54 |      2 |
+----+--------+---------------------+--------+
```

AI写代码sql

那么执行 WITH GRLLUP 语句后返回为：

```
+--------+--------------+
| name   | signin_count |
+--------+--------------+
| 小丽 |            2 |
| 小明 |            7 |
| 小王 |            7 |
| NULL |           16 |
+--------+--------------+
```

AI写代码sql

其中记录 NULL 表示所有人的登录次数。我们使用 coalesce 来设置一个可以取代 NULL 的名称，coalesce 语法：

```
select coalesce(a,b,c);
```

AI写代码sql

参数说明：如果 a == null，则选择 b；如果 b == null，则选择 c；如果 a != null，则选择 a；如果 a b c 都为 null ，则返回为 null（没意义）。

以下实例中如果名字为空我们使用总数代替：

```sql
SELECT coalesce(name, '总数'), SUM(signin) as signin_count
FROM  employee_tbl
GROUP BY name WITH ROLLUP;

+-------------------------+--------------+
| coalesce(name, '总数') | signin_count |
+-------------------------+--------------+
| 小丽                    |            2 |
| 小明                    |            7 |
| 小王                    |            7 |
| 总数                    |           16 |
+-------------------------+--------------+
```

AI写代码sql

NULL 值处理

MySQL 使用 SELECT 命令及 WHERE 子句来读取数据表中的数据，但是当提供的查询条件字段为 NULL 时，该命令可能就无法正常工作。在 MySQL 中，NULL 用于表示缺失的或未知的数据，处理 NULL 值需要特别小心，因为在数据库中它可能会导致不同于预期的结果。为了处理这种情况，MySQL提供了三大运算符：IS NULL：当列的值是 NULL，此运算符返回 true；IS NOT NULL：当列的值不为 NULL，运算符返回 true；<=>：比较操作符（不同于 = 运算符），当比较的的两个值相等或者都为 NULL 时返回 true。关于 NULL 的条件比较运算是比较特殊的。不能使用 = NULL 或 != NULL 在列中查找 NULL 值 。在 MySQL 中，NULL 值与任何其它值的比较（即使是 NULL）永远返回 NULL，即 NULL = NULL 返回 NULL 。MySQL 中处理 NULL 使用 IS NULL 和 IS NOT NULL 运算符。

MySQL 中处理 NULL 值的常见注意事项和技巧：

```sql
-- 1. 检查是否为 NULL：
-- 要检查某列是否为 NULL，可以使用 IS NULL 或 IS NOT NULL 条件。
SELECT * FROM employees WHERE department_id IS NULL;
SELECT * FROM employees WHERE department_id IS NOT NULL;

```

```sql
-- 2. 使用 COALESCE 函数处理 NULL：
-- COALESCE 函数可以用于替换为 NULL 的值，
-- 它接受多个参数，返回参数列表中的第一个非 NULL 值：
SELECT product_name, COALESCE(stock_quantity, 0) AS actual_quantity
FROM products;
-- 以上 SQL 语句中，如果 stock_quantity 列为 NULL，
-- 则 COALESCE 将返回 0。

-- 3. 使用 IFNULL 函数处理 NULL：
-- IFNULL 函数是 COALESCE 的 MySQL 特定版本，
-- 它接受两个参数，如果第一个参数为 NULL，则返回第二个参数。
SELECT product_name, IFNULL(stock_quantity, 0) AS actual_quantity
FROM products;

-- 4. NULL 排序：
-- 在使用 ORDER BY 子句进行排序时，NULL 值默认会被放在排序的最后。
-- NULL 值放在最前面，可以使用 ORDER BY column_name ASC NULLS FIRST，
-- 反之使用 ORDER BY column_name DESC NULLS LAST。
SELECT product_name, price
FROM products
ORDER BY price ASC NULLS FIRST;

-- 5. 使用 <=> 操作符进行 NULL 比较：
-- <=> 操作符是 MySQL 中用于比较两个表达式是否相等的特殊操作符，
-- 对于 NULL 值的比较也会返回 TRUE。它可以用于处理 NULL 值的等值比较。
SELECT *
FROM employees
WHERE commission <=> NULL;

-- 6. 注意聚合函数对 NULL 的处理：
-- 在使用聚合函数（如 COUNT, SUM, AVG）时，它们会忽略 NULL 值，
-- 因此可能会得到不同于预期的结果。如果希望将 NULL 视为 0，
-- 可以使用 COALESCE 或 IFNULL。
SELECT AVG(COALESCE(salary, 0)) AS avg_salary
FROM employees;
-- 这样即使 salary 为 NULL，聚合函数也会将其视为 0。
```

AI写代码sql

正则表达式

MySQL 可以通过 **LIKE ...%** 来进行模糊匹配。MySQL 同样也支持其他正则表达式的匹配， MySQL 中使用 REGEXP 和 RLIKE 操作符来进行正则表达式匹配。下表中的正则模式可应用于 REGEXP 操作符中。

模式	描述
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了RegExp 对象的 Multiline 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
.	匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用像 '[.\n]' 的模式。
[...]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^...]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
p1|p2|p3	匹配 p1 或 p2 或 p3。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。

正则表达式匹配的字符类：.：匹配任意单个字符。^：匹配字符串的开始。$：匹配字符串的结束。*：匹配零个或多个前面的元素。+：匹配一个或多个前面的元素。?：匹配零个或一个前面的元素。[abc]：匹配字符集中的任意一个字符。[^abc]：匹配除了字符集中的任意一个字符以外的字符。[a-z]：匹配范围内的任意一个小写

字母。[0-9]：匹配一个数字字符。\w：匹配一个字母数字字符（包括下划线）。\s：匹配一个空白字符。

REGEXP 是用于进行正则表达式匹配的运算符。REGEXP 用于检查一个字符串是否匹配指定的正则表达式模式，以下是 REGEXP 运算符的基本语法：

```sql
SELECT column1, column2, ...
FROM table_name
WHERE column_name REGEXP 'pattern';
```

AI写代码sql

参数说明：column1，column2，... 是想要选择的列的名称。table_name 是要从中查询数据的表的名称。column_name是要进行正则表达式匹配的列的名称。'pattern' 是一个正则表达式模式。

```sql
-- 查找 name 字段中以 'st' 为开头的所有数据：
SELECT name
FROM person_tbl
WHERE name REGEXP '^st';

-- 查找 name 字段中以 'ok' 为结尾的所有数据：
SELECT name
FROM person_tbl
WHERE name REGEXP 'ok$';

-- 查找 name 字段中包含 'mar' 字符串的所有数据：
SELECT name
FROM person_tbl
WHERE name REGEXP 'mar';

-- 查找 name 字段中以元音字符开头或以 'ok' 字符串结尾的所有数据：
SELECT name
FROM person_tbl
WHERE name REGEXP '^[aeiou]|ok$';

-- 选择订单表中描述中包含 "item" 后跟一个或多个数字的记录。
SELECT *
FROM orders
```

```sql
WHERE order_description REGEXP 'item[0-9]+';

-- 使用 BINARY 关键字，使得匹配区分大小写：
SELECT *
FROM products
WHERE product_name REGEXP BINARY 'apple';

-- 使用 OR 进行多个匹配条件，以下将选择姓氏为 "Smith" 或 "Johnson" 的员工记录：
SELECT *
FROM employees
WHERE last_name REGEXP 'Smith|Johnson';
```

AI写代码sql

RLIKE 是 MySQL 中用于进行正则表达式匹配的运算符，与 REGEXP 是一样的，RLIKE 和 REGEXP 可以互换使用，没有区别。以下是使用 RLIKE 进行正则表达式匹配的基本语法：

```sql
SELECT column1, column2, ...
FROM table_name
WHERE column_name RLIKE 'pattern';
```

AI写代码sql

参数说明：column1，column2，... 是想要选择的列的名称。table_name 是要从中查询数据的表的名称。column_name是要进行正则表达式匹配的列的名称。'pattern' 是一个正则表达式模式。

事务

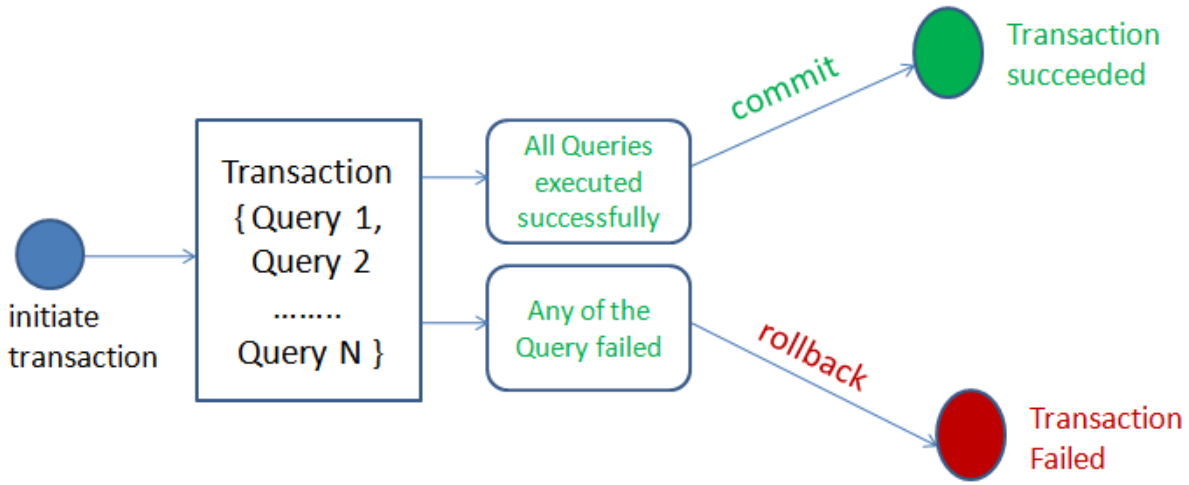
MySQL 事务主要用于处理操作量大，复杂度高的数据。比如说，在人员管理系统中，你删除一个人员，你既需要删除人员的基本资料，也要删除和该人员相关的信息，如信箱，文章等等，这样，这些数据库操作语句就构成一个事务！

在 MySQL 中，事务是一组 SQL 语句的执行，它们被视为一个单独的工作单元。在 MySQL 中只有使用了 Innodb 数据库引擎的数据库或表才支持事务。事务处理可以用来维护数据库的完整性，保证成批的 SQL 语句要么全部执行，要么全部不执行。事务用来管理 insert、update、delete 语句

一般来说，事务是必须满足4个条件（ACID）：：原子性（Atomicity，或称不可分割性）、一致性（Consistency）、隔离性（Isolation，又称独立性）、持久性（Durability）。原子性：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。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。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。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）。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。

在 MySQL 命令行的默认设置下，事务都是自动提交的，即执行 SQL 语句后就会马上执行 COMMIT 操作。因此要显式地开启一个事务务须使用命令 BEGIN 或 START TRANSACTION，或者执行命令 SET AUTOCOMMIT = 0，用来禁止使用当前会话的自动提交。

事务控制语句：BEGIN 或 START TRANSACTION 显式地开启一个事务；COMMIT 也可以使用 COMMIT WORK，不过二者是等价的。COMMIT 会提交事务，并使已对数据库进行的所有修改成为永久性的；ROLLBACK 也可以使用 ROLLBACK WORK，不过二者是等价的。回滚会结束用户的事务，并撤销正在进行的所有未提交的修改；SAVEPOINT identifier，SAVEPOINT 允许在事务中创建一个保存点，一个事务中可以有多个 SAVEPOINT；RELEASE SAVEPOINT identifier 删除一个事务的保存点，当没有指定的保存点时，执行该语句会抛出一个异常；ROLLBACK TO identifier 把事务回滚到标记点；SET TRANSACTION 用来设置事务的隔离级别。InnoDB 存储引擎提供事务的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ 和 SERIALIZABLE。



MYSQL 事务处理主要有两种方法：

用 BEGIN, ROLLBACK, COMMIT 来实现：BEGIN 或 START TRANSACTION：开用于开始一个事务。ROLLBACK 事务回滚，取消之前的更改。COMMIT：事务确认，提交事务，使更改永久生效。

直接用 SET 来改变 MySQL 的自动提交模式：SET AUTOCOMMIT = 0 禁止自动提交。SET AUTOCOMMIT = 1 开启自动提交

```sql
-- BEGIN 或 START TRANSACTION -- 用于开始一个事务：
BEGIN; -- 或者使用 START TRANSACTION;

-- COMMIT -- 用于提交事务，将所有的修改永久保存到数据库：
COMMIT;

-- ROLLBACK -- 用于回滚事务，撤销自上次提交以来所做的所有更改：
ROLLBACK;

-- SAVEPOINT -- 用于在事务中设置保存点，以便稍后能够回滚到该点：
SAVEPOINT savepoint_name;

-- ROLLBACK TO SAVEPOINT -- 用于回滚到之前设置的保存点：
ROLLBACK TO SAVEPOINT savepoint_name;
```

AI写代码sql

下面是一个简单的 MySQL 事务的例子：

```sql
-- 开始事务
START TRANSACTION;

-- 执行一些SQL语句
UPDATE accounts
SET balance = balance - 100
WHERE user_id = 1;
UPDATE accounts
SET balance = balance + 100
WHERE user_id = 2;

-- 判断是否要提交还是回滚
IF (条件) THEN
    COMMIT; -- 提交事务
ELSE
```

```
    ROLLBACK; -- 回滚事务
END IF;
```

AI写代码sql

一个简单的事务实例：

```
use test;
-- 创建表
CREATE TABLE test_transaction_test(id int(5)) engine = innodb;
-- 开始事务
begin;
insert into runoob_transaction_test value(5);
insert into runoob_transaction_test value(6);
-- 提交事务
commit;
select *
from runoob_transaction_test;
+------+
| id   |
+------+
| 5    |
| 6    |
+------+

begin;
insert into runoob_transaction_test values(7);
-- 回滚
rollback;

-- 因为回滚所以数据没有插入
select * from runoob_transaction_test;
+------+
| id   |
+------+
```

```
| 5    |30 | 6    |
+------+
```

AI写代码sql

ALTER 命令

当我们需要修改数据表名或者修改数据表字段时，就需要使用到 MySQL ALTER 命令。MySQL 的 ALTER 命令用于修改数据库、表和索引等对象的结构。ALTER 命令允许添加、修改或删除数据库对象，并且可以用于更改表的列定义、添加约束、创建和删除索引等操作。ALTER 命令非常强大，可以在数据库结构发生变化时进行灵活的修改和调整。以下是 ALTER 命令的常见用法和实例：

- 添加列

```
ALTER TABLE table_name
ADD COLUMN new_column_name datatype;
```

AI写代码sql

以下 SQL 语句在 employees 表中添加了一个名为 birth_date 的日期列：

```
ALTER TABLE employees
ADD COLUMN birth_date DATE;
```

AI写代码sql

- 修改列的数据类型

```
ALTER TABLE TABLE_NAME
MODIFY COLUMN column_name new_datatype;
```

AI写代码sql

以下 SQL 语句将 employees 表中的 salary 列的数据类型修改为 DECIMAL(10, 2)：

```sql
ALTER TABLE employees
MODIFY COLUMN salary DECIMAL(10,2);
```

AI写代码sql

- 修改列名

```sql
ALTER TABLE table_name
CHANGE COLUMN old_column_name new_column_name datatype;
```

AI写代码sql

以下 SQL 语句将 employees 表中的某个列的名字由 old_name 修改为 new_name，并且同时修改数据类型：

```sql
ALTER TABLE employees
CHANGE COLUMN old_name new_name VARCHAR(255);
```

AI写代码sql

- 删除列

```sql
ALTER TABLE table_name
DROP COLUMN column_name;
```

AI写代码sql

以下 SQL 语句将 employees 表中的 birth_date 列删除：

```sql
ALTER TABLE employees
```

```
2 DROP COLUMN birth_date;
```

AI写代码sql

- 添加 PRIMARY KEY

```
1 ALTER TABLE table_name
2 ADD PRIMARY KEY (column_name);
```

AI写代码sql

以下 SQL 语句在 employees 表中添加了一个主键：

```
1 ALTER TABLE employees
2 ADD PRIMARY KEY (employee_id);
```

AI写代码sql

- 添加 FOREIGN KEY

```
1 ALTER TABLE child_table
2 ADD CONSTRAINT fk_name
3 FOREIGN KEY (column_name)
4 REFERENCES parent_table (column_name);
```

AI写代码sql

以下 SQL 语句在 orders 表中添加了一个外键，关联到 customers 表的 customer_id 列：

```
1 ALTER TABLE orders
2 ADD CONSTRAINT fk_customer
```

```
3 FOREIGN KEY (customer_id)
4 REFERENCES customers (customer_id);
```

AI写代码sql

- 修改表名

```
1 ALTER TABLE old_table_name
2 RENAME TO new_table_name;
```

AI写代码sql

以下 SQL 语句将表名由 employees 修改为 staff：

```
1 ALTER TABLE employees
2 RENAME TO staff;
```

AI写代码sql

索引

MySQL 索引是一种数据结构，用于加快数据库查询的速度和性能。MySQL 索引的建立对于 MySQL 的高效运行是很重要的，索引可以大大提高 MySQL 的检索速度。

MySQL 索引类似于书籍的索引，通过存储指向数据行的指针，可以快速定位和访问表中的特定数据。打个比方，如果合理的设计且使用索引的 MySQL 是一辆兰博基尼的话，那么没有设计和使用索引的 MySQL 就是一个人力三轮车。拿汉语字典的目录页（索引）打比方，我们可以按拼音、笔画、偏旁部首等排序的目录（索引）快速查找到需要的字。

索引分单列索引和组合索引：单列索引，即一个索引只包含单个列，一个表可以有多个单列索引。组合索引，即一个索引包含多个列。

创建索引时，需要确保该索引是应用在 SQL 查询语句的条件（一般作为 WHERE 子句的条件）。实际上，索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录。

索引虽然能够提高查询性能，但也需要注意以下几点：索引需要占用额外的存储空间。对表进行插入、更新和删除操作时，索引需要维护，可能会影响性能。过多或不合理的索引可能会导致性能下降，因此需要谨慎选择和规划索引。

- 普通索引

> 索引能够显著提高查询的速度，尤其是在大型表中进行搜索时。通过使用索引，MySQL 可以直接定位到满足查询条件的数据行，而无需逐行扫描整个表。
> 使用 CREATE INDEX 语句可以创建普通索引。普通索引是最常见的索引类型，用于加速对表中数据的查询。CREATE INDEX 的语法：
>
> ```sql
> CREATE INDEX index_name
> ON table_name (column1 [ASC|DESC], column2 [ASC|DESC], ...);
> ```
>
> AI写代码sql
>
> 参数说明：index_name 指定要创建的索引的名称。索引名称在表中必须是唯一的。table_name 指定要在哪个表上创建索引。column1 [ASC|DESC]，column2 [ASC|DESC]... 指定要索引的表列名，可以指定一个或多个列作为索引的组合，这些列的数据类型通常是数值、文本或日期。ASC 和 DESC（可选）: 用于指定索引的排序顺序。默认情况下，索引以升序（ASC）排序。
> 以下实例假设我们有一个名为 students 的表，包含 id、name 和 age 列，我们将在 name 列上创建一个普通索引：
>
> ```sql
> CREATE INDEX idx_name ON students (name);
> ```
>
> AI写代码sql
>
> 上述语句将在 students 表的 name 列上创建一个名为 idx_name 的普通索引，这将有助于提高通过姓名进行搜索的查询性能。需要注意的是，如果表中的数据量较大，索引的创建可能会花费一些时间，但一旦创建完成，查询性能将会显著提高。
> 使用 ALTER TABLE 命令可以在已有的表中创建索引。ALTER TABLE 允许你修改表的结构，包括添加、修改或删除索引。ALTER TABLE 创建索引的语法：
>
> ```sql
> ALTER TABLE table_name
> ADD INDEX index_name (column1 [ASC|DESC], column2 [ASC|DESC], ...);
> ```
>
> AI写代码sql
>
> 参数说明：table_name 指定要修改的表的名称。ADD INDEX 添加索引的子句，ADD INDEX 用于创建普通索引。index_name 指定要创建的索引的名称。索引名称在表中必须是唯一的。column1 [ASC|DESC]，column2 [ASC|DESC]... 指定要索引的表列名，可以指定一个或多个列作为索引的组合，这些列的数据类型通常是数值、文本或日期。ASC 和 DESC（可选）: 用于指定索引的排序顺序。默认情况下，索引以升序（ASC）排序。
> 下面是一个实例，我们将在已存在的名为 employees 的表上创建一个普通索引：

```sql
ALTER TABLE employees 2 ADD INDEX idx_age (age);
```

AI写代码sql

上述语句将在 employees 表的 age 列上创建一个名为 idx_age 的普通索引。
可以在创建表的时候，可以在 CREATE TABLE 语句中直接指定索引，以创建表和索引的组合。

```sql
CREATE TABLE table_name (
  column1 data_type,
  column2 data_type,
  ...,
  INDEX index_name (column1 [ASC|DESC], column2 [ASC|DESC], ...)
);
```

AI写代码sql

下面是一个实例，我们要创建一个名为 students 的表，并在 age 列上创建一个普通索引。

```sql
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT,
  INDEX idx_age (age)
);
```

AI写代码sql

在上述实例中，我们在 students 表的 age 列上创建了一个名为 idx_age 的普通索引。
可以使用 DROP INDEX 语句来删除索引。DROP INDEX 的语法：

```sql
DROP INDEX index_name ON table_name;
```

AI写代码sql

参数说明：index_name 指定要删除的索引的名称。 table_name 指定要在哪个表上删除索引。
使用 ALTER TABLE 语句删除索引的语法如下：

```sql
ALTER TABLE table_name
```

```
2 DROP INDEX index_name;
```

AI写代码sql

以下实例假设我们有一个名为 employees 的表，并在 age 列上有一个名为 idx_age 的索引，现在我们要删除这个索引：

```
1 DROP INDEX idx_age ON employees;
2
3 或
4
5 ALTER TABLE employees
6 DROP INDEX idx_age;
```

AI写代码sql

这两个命令都会从 employees 表中删除名为 idx_age 的索引。如果该索引不存在，执行命令时会产生错误。因此，在删除索引之前最好确认该索引是否存在，或者使用错误处理机制来处理可能的错误情况。

- 唯一索引

在 MySQL 中，你可以使用 CREATE UNIQUE INDEX 语句来创建唯一索引。唯一索引确保索引中的值是唯一的，不允许有重复值。

```
1 CREATE UNIQUE INDEX index_name
2 ON table_name (column1 [ASC|DESC], column2 [ASC|DESC], ...);
```

AI写代码sql

以下是一个创建唯一索引的实例： 假设我们有一个名为 employees的 表，包含 id 和 email 列，现在我们想在 email 列上创建一个唯一索引，以确保每个员工的电子邮件地址都是唯一的。

```
CREATE UNIQUE INDEX idx_email ON employees (email);
```

AI写代码sql

可以使用 ALTER TABLE 命令来创建唯一索引。ALTER TABLE命令允许你修改已经存在的表结构，包括添加新的索引。

```sql
ALTER table table_name  ADD CONSTRAINT unique_constraint_name UNIQUE (column1, column2, ...);
```

AI写代码sql

以下是一个使用 ALTER TABLE 命令创建唯一索引的实例：假设我们有一个名为 employees 的表，包含 id 和 email 列，现在我们想在 email 列上创建一个唯一索引，以确保每个员工的电子邮件地址都是唯一的。

```sql
ALTER TABLE employees
ADD CONSTRAINT idx_email UNIQUE (email);
```

AI写代码sql

以上实例将在 employees 表的 email 列上创建一个名为 idx_email 的唯一索引。请注意，如果表中已经有重复的 email 值，那么添加唯一索引将会失败。在创建唯一索引之前，你可能需要确保表中的 email 列没有重复的值。

也可以在创建表的同时，你可以在 CREATE TABLE 语句中使用 UNIQUE 关键字来创建唯一索引。这将在表创建时同时定义唯一索引约束。语法如下：

```sql
CREATE TABLE table_name (
  column1 data_type,
  column2 data_type,
  ...,
  CONSTRAINT index_name UNIQUE (column1 [ASC|DESC], column2 [ASC|DESC]...)
);
```

AI写代码sql

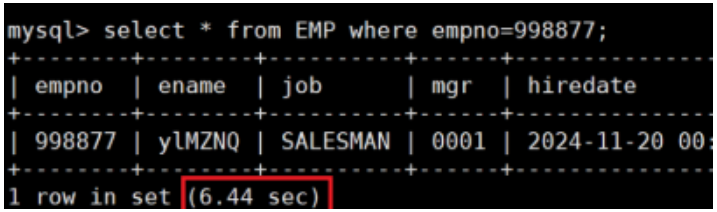
以下是一个在创建表时创建唯一索引的实例：假设我们要创建一个名为 employees 的表，其中包含 id、name 和 email 列，我们希望 email 列的值是唯一的，因此我们要在创建表时定义唯一索引。

```sql
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(100),
  CONSTRAINT index_email UNIQUE (email)
);
```

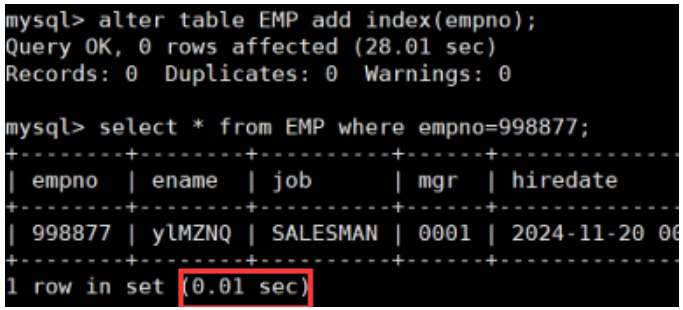
AI写代码sql

使用索引和不使用用所以对比：

不使用：



使用：



临时表

MySQL 临时表在我们需要保存一些临时数据时是非常有用的。临时表只在当前连接可见，当关闭连接时，MySQL 会自动删除表并释放所有空间。在 MySQL 中，临时表是一种在当前会话中存在的表，它在会话结束时会自动被销毁。

如果你使用了其他 MySQL 客户端程序连接 MySQL 数据库服务器来创建临时表，那么只有在关闭客户端程序时才会销毁临时表，当然你也可以手动销毁。

```
CREATE TEMPORARY TABLE temp_table_name (
  column1 datatype,
  column2 datatype,
  ...
);

-- 或者简写

CREATE TEMPORARY TABLE temp_table_name AS
```

```
SELECT column1, column2, ...
FROM source_table
WHERE condition;
```

AI写代码sql

插入数据到临时表：

```
INSERT INTO temp_table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

AI写代码sql

查询临时表：

```
SELECT * FROM temp_table_name;
```

AI写代码sql

修改临时表（临时表的修改操作与普通表类似，可以使用 ALTER TABLE 命令。）：

```
ALTER TABLE temp_table_name
ADD COLUMN new_column datatype;
```

AI写代码sql

删除临时表（临时表在会话结束时会自动被销毁，但也可以使用 DROP TABLE 明确删除它）：

```
DROP TEMPORARY TABLE IF EXISTS temp_table_name;
```

AI写代码sql

实例：

```sql
-- 创建临时表
CREATE TEMPORARY TABLE temp_orders AS
SELECT * FROM orders WHERE order_date >= '2023-01-01';

-- 查询临时表
SELECT * FROM temp_orders;

-- 插入数据到临时表
INSERT INTO temp_orders (order_id, customer_id, order_date)
VALUES (1001, 1, '2023-01-05');

-- 查询临时表
SELECT * FROM temp_orders;

-- 删除临时表
DROP TEMPORARY TABLE IF EXISTS temp_orders;
```

AI写代码sql

临时表对于需要在某个会话中存储中间结果集或进行复杂查询时非常有用。临时表的作用范围仅限于创建它的会话。其他会话无法直接访问或引用该临时表。在多个会话之间共享数据时，可以考虑使用普通表而不是临时表。请注意，临时表在会话结束时会被自动删除，但也可以使用 DROP TEMPORARY TABLE 明确删除它，这样可以更早地释放资源。

复制表

如果我们需要完全的复制 MySQL 的数据表，包括表的结构，索引，默认值等。如果仅仅使用 CREATE TABLE ... SELECT 命令，是无法实现的。步骤如下：使用 SHOW CREATE TABLE 命令获取创建数据表（CREATE TABLE）语句，该语句包含了原数据表的结构，索引等；复制以下命令显示的 SQL 语句，修改数据表名，并执行SQL语句，通过以上命令 将完全的复制数据表结构；如果你想复制表的内容，你就可以使用 INSERT INTO ... SELECT 语句来实现。

例如复制 score 表：

步骤 1：获取数据表的完整结构。

```
mysql> show create table score;
+-------+----------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
| Table | Create Table
                                                                                                   |
+-------+----------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
| score | CREATE TABLE `score` (
  `id` int NOT NULL,
  `jz` double NOT NULL,
  `java` double NOT NULL,
  `class` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+----------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------+
```

步骤2：修改 SQL 语句的数据表名，并执行 SQL 语句。

```
mysql> CREATE TABLE `copy_score`(
    -> `id` int NOT NULL,
    ->   `jz` double NOT NULL,
    ->   `java` double NOT NULL,
    ->   `class` int NOT NULL,
    ->   PRIMARY KEY (`id`)
    -> ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
Query OK, 0 rows affected (0.02 sec)
```

步骤3：执行完第二步骤后，将在数据库中创建新的克隆表 clone_tbl。 如果想拷贝数据表的数据可以使用 INSERT INTO... SELECT 语句来实现。

```
mysql> INSERT INTO copy_score (id, jz, java, class)
    -> SELECT id, jz, java, class
    -> FROM score;
Query OK, 100 rows affected (0.01 sec)
Records: 100  Duplicates: 0  Warnings: 0
```

元数据

MySQL 元数据是关于数据库和其对象（如表、列、索引等）的信息。元数据存储在系统表中，这些表位于 MySQL 数据库的 information_schema 数据库中，通过查询这些系统表，你可以获取关于数据库结构、对象和其他相关信息的详细信息。

想知道 MySQL 以下三种信息：查询结果信息： SELECT，UPDATE 或 DELETE 语句影响的记录数。数据库和数据表的信息： 包含了数据库及数据表的结构信息。MySQL 服务器信息： 包含了数据库服务器的当前状态，版本号等。

以下是一些常用的 MySQL 元数据查询：

```
-- 查看所有数据库：
SHOW DATABASES;

-- 选择数据库：
USE database_name;

-- 查看数据库中的所有表：
SHOW TABLES;

-- 查看表的结构：
DESC table_name;

-- 查看表的索引：
SHOW INDEX FROM table_name;

-- 查看表的创建语句：
SHOW CREATE TABLE table_name;

-- 查看表的行数：
SELECT COUNT(*) FROM table_name;

-- 查看列的信息：
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_KEY
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'your_database_name'
AND TABLE_NAME = 'your_table_name';
-- 以上SQL 语句中的 'your_database_name'
-- 和 'your_table_name' 分别是你的数据库名和表名。

-- 查看外键信息：
SELECT
```

```
    TABLE_NAME,    COLUMN_NAME,
    CONSTRAINT_NAME,
    REFERENCED_TABLE_NAME,
    REFERENCED_COLUMN_NAME
FROM
    INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE
    TABLE_SCHEMA = 'your_database_name'
    AND TABLE_NAME = 'your_table_name'
    AND REFERENCED_TABLE_NAME IS NOT NULL;
-- 上述 SQL 语句中的 'your_database_name'
-- 和 'your_table_name' 为实际的数据库名和表名。
```

AI写代码sql

- information_schema 数据库

> information_schema 是 MySQL 数据库中的一个系统数据库，它包含有关数据库服务器的元数据信息，这些信息以表的形式存储在 information_schema 数据库中。
>
> SCHEMATA 表：存储有关数据库的信息，如数据库名、字符集、排序规则等。
>
> ```
> SELECT *
> FROM information_schema.SCHEMATA;
> ```
>
> AI写代码sql
>
> TABLES 表：包含有关数据库中所有表的信息，如表名、数据库名、引擎、行数等。
>
> ```
> SELECT *
> FROM information_schema.TABLES
> WHERE TABLE_SCHEMA = 'your_database_name';
> ```
>
> AI写代码sql
>
> COLUMNS 表：包含有关表中列的信息，如列名、数据类型、是否允许 NULL 等。

```sql
SELECT *
FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA = 'your_database_name'
AND TABLE_NAME = 'your_table_name';
```

AI写代码sql

STATISTICS 表：提供有关表索引的统计信息，如索引名、列名、唯一性等。

```sql
SELECT *
FROM information_schema.STATISTICS
WHERE TABLE_SCHEMA = 'your_database_name'
AND TABLE_NAME = 'your_table_name';
```

AI写代码sql

KEY_COLUMN_USAGE 表：包含有关表中外键的信息，如外键名、列名、关联表等。

```sql
SELECT * FROM information_schema.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'your_database_name'
AND TABLE_NAME = 'your_table_name';
```

AI写代码sql

REFERENTIAL_CONSTRAINTS 表：存储有关外键约束的信息，如约束名、关联表等。

```sql
SELECT *
FROM information_schema.REFERENTIAL_CONSTRAINTS
WHERE CONSTRAINT_SCHEMA = 'your_database_name'
AND TABLE_NAME = 'your_table_name';
```

AI写代码sql

这些表提供了丰富的元数据信息，可以用于查询数据库结构、表信息、列信息、索引信息等。需要将查询中的 'your_database_name' 和 'your_table_name' 替换为实际的数据库名和表名。

序列

在 MySQL 中，序列是一种自增生成数字序列的对象，是一组整数 1、2、3、...，由于一张数据表只能有一个字段自增主键。尽管 MySQL 本身并没有内建的序列类型，但可以使用 AUTO_INCREMENT 属性来模拟序列的行为，通常 AUTO_INCREMENT 属性用于指定表中某一列的自增性。

一个使用 AUTO_INCREMENT 创建表的例子：

```sql
CREATE TABLE example_table (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50)
);
```

AI写代码sql

以上例子中，id 列被定义为 INT AUTO_INCREMENT，这表示每次插入一行数据时，id 列的值会自动增加。主键约束保证了 id 列的唯一性。当你插入一行数据时，可以不指定 id 列的值，数据库会自动为其分配一个唯一的、自增的值：

```sql
INSERT INTO example_table (name) VALUES ('John');
```

AI写代码sql

处理重复数据

有些 MySQL 数据表中可能存在重复的记录，有些情况我们允许重复数据的存在，但有时候我们也需要删除这些重复的数据。

可以在 MySQL 数据表中设置指定的字段为 PRIMARY KEY（主键） 或者 UNIQUE（唯一） 索引来保证数据的唯一性。

一个实例：下表中无索引及主键，所以该表允许出现多条重复记录。

```sql
CREATE TABLE person_tbl (
    first_name CHAR(20),
    last_name CHAR(20),
    sex CHAR(10)
);
```

AI写代码sql

如果想设置表中字段 first_name，last_name 数据不能重复，你可以设置双主键模式来设置数据的唯一性， 如果你设置了双主键，那么那个键的默认值不能为 NULL，可设置为 NOT NULL。如下所示：

```sql
CREATE TABLE person_tbl (
   first_name CHAR(20) NOT NULL,
   last_name CHAR(20) NOT NULL,
   sex CHAR(10),
   PRIMARY KEY (last_name, first_name)
);
```

AI写代码sql

如果我们设置了唯一索引，那么在插入重复数据时，SQL 语句将无法执行成功，并抛出错。INSERT IGNORE INTO 与 INSERT INTO 的区别就是 INSERT IGNORE INTO 会忽略数据库中已经存在的数据，如果数据库没有数据，就插入新的数据，如果有数据的话就跳过这条数据。这样就可以保留数据库中已经存在数据，达到在间隙中插入数据的目的。以下实例使用了 INSERT IGNORE INTO，执行后不会出错，也不会向数据表中插入重复数据：

```sql
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Jay', 'Thomas');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT IGNORE INTO person_tbl (last_name, first_name)
    -> VALUES( 'Jay', 'Thomas');
Query OK, 0 rows affected (0.00 sec)
```

AI写代码sql

INSERT IGNORE INTO 当插入数据时，在设置了记录的唯一性后，如果插入重复数据，将不返回错误，只以警告形式返回。 而 REPLACE INTO 如果存在 primary 或 unique 相同的记录，则先删除掉。再插入新记录。

另一种设置数据的唯一性方法是添加一个 UNIQUE 索引，如下所示：

```sql
CREATE TABLE person_tbl (
   first_name CHAR(20) NOT NULL,
   last_name CHAR(20) NOT NULL,
   sex CHAR(10),
   UNIQUE (last_name, first_name)
```

```
);
```

AI写代码sql

以下我们将统计表中 first_name 和 last_name的重复记录数：

```
SELECT COUNT(*) as repetitions, last_name, first_name
FROM person_tbl
GROUP BY last_name, first_name
HAVING repetitions > 1;
```

AI写代码sql

以上查询语句将返回 person_tbl 表中重复的记录数。 一般情况下，查询重复的值，请执行以下操作：确定哪一列包含的值可能会重复。在列选择列表使用COUNT(*)列出的那些列。在GROUP BY子句中列出的列。HAVING子句设置重复数大于1

如果需要读取不重复的数据可以在 SELECT 语句中使用 DISTINCT 关键字来过滤重复数据，也可以使用 GROUP BY 来读取数据表中不重复的数据：

```
SELECT DISTINCT last_name, first_name
FROM person_tbl;

-- 或

SELECT last_name, first_name
FROM person_tbl
GROUP BY (last_name, first_name);
```

AI写代码sql

如果想删除数据表中的重复数据，你可以使用以下的SQL语句：

```
CREATE TABLE tmp
SELECT last_name, first_name, sex
FROM person_tbl
GROUP BY (last_name, first_name, sex);
DROP TABLE person_tbl;
```

```sql
ALTER TABLE tmp RENAME TO person_tbl;
```

AI写代码sql

当然也可以在数据表中添加 INDEX（索引） 和 PRIMAY KEY（主键）这种简单的方法来删除表中的重复记录。方法如下：

```sql
ALTER IGNORE TABLE person_tbl
ADD PRIMARY KEY (last_name, first_name);
```

AI写代码sql

MySQL 及 SQL 注入

如果通过网页获取用户输入的数据并将其插入一个 MySQL 数据库，那么就有可能发生 SQL 注入安全的问题。所谓 SQL 注入，就是通过把 SQL 命令插入到 Web 表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。

MySQL 注入是指攻击者通过恶意构造的输入，成功地执行恶意的 SQL 查询，这通常发生在用户输入未经适当验证或转义的情况下，攻击者试图在输入中插入 SQL 代码，以执行意外的查询或破坏数据库。我们永远不要信任用户的输入，我们必须认定用户输入的数据都是不安全的，我们都需要对用户输入的数据进行过滤处理。

假设有一个登录系统，用户通过输入用户名和密码进行身份验证：

```sql
SELECT *
FROM users
WHERE username = 'input_username' AND password = 'input_password';
```

AI写代码sql

如果没有正确的输入验证和防范措施，攻击者可以输入类似于以下内容的用户名：

```sql
' OR '1'='1'; --
```

AI写代码sql

在这种情况下，SQL 查询会变成：

```sql
SELECT *
```

```
FROM users
WHERE username = '' OR '1'='1'; --' AND password = 'input_password';
```

AI写代码sql

这会使查询返回所有用户，因为 1 = 1 总是为真，注释符号 -- 用于注释掉原始查询的其余部分，以确保语法正确。

防范 SQL 注入：使用参数化查询或预编译语句： 使用参数化查询（Prepared Statements）可以有效防止 SQL 注入，因为它们在执行查询之前将输入数据与查询语句分离。输入验证和转义： 对用户输入进行适当的验证，并使用合适的转义函数（如 mysqli_real_escape_string）来处理输入，以防止恶意注入。最小权限原则： 给予数据库用户最小的权限，确保它们只能执行必要的操作，以降低潜在的损害。使用ORM框架： 使用对象关系映射（ORM）框架（如Hibernate、Sequelize）可以帮助抽象 SQL 查询，从而降低 SQL 注入的风险。禁用错误消息显示： 在生产环境中，禁用显示详细的错误消息，以防止攻击者获取有关数据库结构的敏感信息。

导出/导入数据

可以使用 SELECT...INTO OUTFILE 语句来简单的导出数据到文本文件上。SELECT...INTO OUTFILE 是 MySQL 中用于将查询结果导出到文件的语法。允许将查询的结果写入一个文本文件，基本的使用方法：

```
SELECT column1, column2, ...
INTO OUTFILE 'file_path'
FROM your_table
WHERE your_conditions;
```

AI写代码sql

参数说明：column1，column2，...: 是要选择的列。'file_path' 指定输出文件的路径和名称。your_table 要查询的表。your_conditions 查询条件。

以下是一个简单的实例：

```
SELECT id, name, email
INTO OUTFILE '/tmp/user_data.csv'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
FROM users;
```

AI写代码sql

在以上 SQL 语句中，我们从 users 表中选择了 id、name 和 email 列，并将结果写入了 /tmp/user_data.csv 文件。FIELDS TERMINATED BY ',' 指定了列之间的分隔符（逗号），LINES TERMINATED BY '\n' 指定了行之间的分隔符（换行符）。

需要注意的是，执行 SELECT...INTO OUTFILE 需要相应的权限，并且输出文件的目录需要是 MySQL 服务器可以写入的地方。

mysqldump 是 MySQL 提供的用于备份和导出数据库的命令行工具。mysqldump 是 mysql 用于转存储数据库的实用程序。它主要产生一个 SQL 脚本，其中包含从头重新创建数据库所必需的命令 CREATE TABLE INSERT 等。使用 mysqldump 导出数据需要使用 --tab 选项来指定导出文件指定的目录，该目标必须是可写的。
mysqldump 基本的用法：

```
mysqldump -u username -p password -h hostname database_name > output_file.sql
```

AI写代码sql

参数说明：-u 指定 MySQL 用户名。 -p 提示输入密码。-h 指定 MySQL 主机名。database_name 要导出的数据库名称。output_file.sql 导出数据保存到的文件。

使用 mysql 命令导入语法格式为：

```
mysql -u username -p -h your_host -P your_port -D your_database
```

AI写代码sql

source 命令导入数据库需要先登录到数库终端：

```
mysql> create database abc;      # 创建数据库
mysql> use abc;                  # 使用已创建的数据库
mysql> set names utf8;           # 设置编码
mysql> source /home/abc/abc.sql  # 导入备份数据库
```

AI写代码sql

使用 source 命令的好处是，你可以在 MySQL 命令行中直接执行，而无需退出 MySQL 并使用其他命令。

MySQL 中提供了LOAD DATA INFILE语句来插入数据。 以下实例中将从当前目录中读取文件 dump.txt ， 将该文件中的数据插入到当前数据库的 mytbl 表中。

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl;
```

AI写代码sql

如果指定 LOCAL 关键词，则表明从客户主机上按路径读取文件。如果没有指定，则文件在服务器上按路径读取文件。你能明确地在 LOAD DATA 语句中指出列值的分隔符和行尾标记，但是默认标记是定位符和换行符。两个命令的 FIELDS 和 LINES 子句的语法是一样的。两个子句都是可选的，但是如果两个同时被指定，FIELDS 子句必须出现在 LINES 子句之前。

如果用户指定一个 FIELDS 子句，它的子句 （TERMINATED BY、[OPTIONALLY] ENCLOSED BY 和 ESCAPED BY) 也是可选的，不过，用户必须至少指定它们中的一个。

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl
  -> FIELDS TERMINATED BY ':'
  -> LINES TERMINATED BY '\r\n';
```

AI写代码sql

LOAD DATA 默认情况下是按照数据文件中列的顺序插入数据的，如果数据文件中的列与插入表中的列不一致，则需要指定列的顺序。如，在数据文件中的列顺序是a,b,c，但在插入表的列顺序为b,c,a，则数据导入语法如下：

```
mysql> LOAD DATA LOCAL INFILE 'dump.txt'
    -> INTO TABLE mytbl (b, c, a);
```

AI写代码sql

函数

MySQL 有很多内置的函数，以下列出了这些函数的说明。

字符串函数：

函数	描述	实例
ASCII(s)	返回字符串 s 的第一个字符的 ASCII 码。	返回 CustomerName 字段第一个字母的 ASCII 码： `SELECT ASCII(CustomerName) AS NumCodeOfFirstChar`

		FROM Customers;
CHAR_LENGTH(s)	返回字符串 s 的字符数	返回字符串 RUNOOB 的字符数 SELECT CHAR_LENGTH("RUNOOB") AS LengthOfString;
CHARACTER_LENGTH(s)	返回字符串 s 的字符数，等同于 CHAR_LENGTH(s)	返回字符串 RUNOOB 的字符数 SELECT CHARACTER_LENGTH("RUNOOB") AS LengthOfString;
CONCAT(s1,s2...sn)	字符串 s1,s2 等多个字符串合并为一个字符串	合并多个字符串 SELECT CONCAT("SQL ", "Runoob ", "Gooogle ", "Facebook") AS ConcatenatedString;
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，x 可以是分隔符	合并多个字符串，并添加分隔符： SELECT CONCAT_WS("-", "SQL", "Tutorial", "is", "fun!") AS ConcatenatedString;
FIELD(s,s1,s2...)	返回第一个字符串 s 在字符串列表(s1,s2...)中的位置	返回字符串 c 在列表值中的位置： SELECT FIELD("c", "a", "b",

		"c", "d", "e");
FIND_IN_SET(s1,s2)	返回在字符串s2中与s1匹配的字符串的位置	返回字符串 c 在指定字符串中的位置： `SELECT FIND_IN_SET("c", "a,b,c,d,e");`

FORMAT(x,n)	函数可以将数字 x 进行格式化 "#,###.##", 将 x 保留到小数点后 n 位，最后一位四舍五入。	格式化数字 "#,###.##" 形式：
		`SELECT FORMAT(250500.5634, 2);`
		`-- 输出 250,500.56`

INSERT(s1,x,len,s2)	字符串 s2 替换 s1 的 x 位置开始长度为 len 的字符串	从字符串第一个位置开始的 6 个字符替换为 runoob：
		`SELECT INSERT("google.com", 1, 6, "runoob");`
		`-- 输出：runoob.com`

LOCATE(s1,s)	从字符串 s 中获取 s1 的开始位置	获取 b 在字符串 abc 中的位置：
		`SELECT LOCATE('st','myteststring');`
		`-- 5`
		
		返回字符串 abc 中 b 的位置：
		`SELECT LOCATE('b', 'abc')`
		`-- 2`

LCASE(s)	将字符串 s 的所有字母变成小写字母	字符串 RUNOOB 转换为小写：
		`SELECT LCASE('RUNOOB')`
		`-- runoob`

LEFT(s,n)	返回字符串 s 的前 n 个字符	返回字符串 runoob 中的前两个字符：
		`SELECT LEFT('runoob',2)`
		`-- ru`

LOWER(s)	将字符串 s 的所有字母变成小写字母	字符串 RUNOOB 转换为小写：
		`SELECT LOWER('RUNOOB')`
		`-- runoob`

LPAD(s1,len,s2)	在字符串 s1 的开始处填充字符串 s2，使字符串长度达到 len	将字符串 xx 填充到 abc 字符串的开始处：
		`SELECT LPAD('abc',5,'xx')`
		`-- xxabc`

LTRIM(s)	去掉字符串 s 开始处的空格	去掉字符串 RUNOOB开始处的空格：
		`SELECT LTRIM("    RUNOOB")`
		`AS LeftTrimmedString;`
		`-- RUNOOB`

MID(s,n,len)	从字符串 s 的 n 位置截取长度为 len 的子字符串，同 SUBSTRING(s,n,len)	从字符串 RUNOOB 中的第 2 个位置截取 3个 字符：
		`SELECT MID("RUNOOB", 2, 3) AS ExtractString;`
		`-- UNO`

POSITION(s1 IN s)	从字符串 s 中获取 s1 的开始位置	返回字符串 abc 中 b 的位置：
		`SELECT POSITION('b' in 'abc')`
		`-- 2`

REPEAT(s,n)	将字符串 s 重复 n 次	将字符串 runoob 重复三次：
		`SELECT REPEAT('runoob',3)`
		`-- runoobrunoobrunoob`

REPLACE(s,s1,s2)	将字符串 s2 替代字符串 s 中的字符串 s1	将字符串 abc 中的字符 a 替换为字符 x：
		`SELECT REPLACE('abc','a','x')`
		`--xbc`

REVERSE(s)	将字符串s的顺序反过来	将字符串 abc 的顺序反过来：
		`SELECT REVERSE('abc')`
		`-- cba`

RIGHT(s,n)	返回字符串 s 的后 n 个字符	返回字符串 runoob 的后两个字符：
		`SELECT RIGHT('runoob',2)`
		`-- ob`

RPAD(s1,len,s2)	在字符串 s1 的结尾处添加字符串 s2，使字符串的长度达到 len	将字符串 xx 填充到 abc 字符串的结尾处：
		`SELECT RPAD('abc',5,'xx')`
		`-- abcxx`

RTRIM(s)	去掉字符串 s 结尾处的空格	去掉字符串 RUNOOB 的末尾空格：
		`SELECT RTRIM("RUNOOB     ")`
		`AS RightTrimmedString;`
		`-- RUNOOB`

SPACE(n)	返回 n 个空格	返回 10 个空格： `SELECT SPACE(10);`

STRCMP(s1,s2)	比较字符串 s1 和 s2，如果 s1 与 s2 相等返回 0 ，如果 s1>s2 返回 1，如果 s1<s2 返回 -1	比较字符串：
		`SELECT STRCMP("runoob", "runoob");`
		`-- 0`

SUBSTR(s, start, length)	从字符串 s 的 start 位置截取长度为 length 的子字符串	从字符串 RUNOOB 中的第 2 个位置截取 3个 字符：
		`SELECT SUBSTR("RUNOOB", 2, 3)`
		`AS ExtractString;`
		`-- UNO`

SUBSTRING(s, start, length)	从字符串 s 的 start 位置截取长度为 length 的子字符串，等同于 SUBSTR(s, start, length)	从字符串 RUNOOB 中的第 2 个位置截取 3个 字符：
		`SELECT SUBSTRING("RUNOOB", 2, 3)`
		`AS ExtractString;`

		`-- UNO`

SUBSTRING_INDEX(s, delimiter, number)	返回从字符串 s 的第 number 个出现的分隔符 delimiter 之后的子串。
	如果 number 是正数，返回第 number 个字符左边的字符串。
	如果 number 是负数，返回第(number 的绝对值(从右边数))个字符右边的字符串。
		`SELECT SUBSTRING_INDEX('a*b','*',1)`
		`-- a`
		`SELECT`
		`SUBSTRING_INDEX('a*b','*',-1)`
		`-- b`
		`SELECT SUBSTRING_INDEX(`
		`SUBSTRING_INDEX('a*b*c*d*e',`
		`'*',`
		`3),`
		`'*',`
		`-1)`
		`-- c`

TRIM(s)	去掉字符串 s 开始和结尾处的空格	去掉字符串 RUNOOB 的首尾空格：
		`SELECT TRIM('  RUNOOB  ')`
		`AS TrimmedString;`

UCASE(s)	将字符串转换为大写	将字符串 runoob 转换为大写：
		`SELECT UCASE("runoob");`
		`-- RUNOOB`

UPPER(s)	将字符串转换为大写	将字符串 runoob 转换为大写：
		`SELECT UPPER("runoob");`
		`-- RUNOOB`

数字函数

函数名	描述	实例
ABS(x)	返回 x 的绝对值	返回 -1 的绝对值： `SELECT ABS(-1) -- 返回1`
ACOS(x)	求 x 的反余弦值（单位为弧度），x 为一个数值	`SELECT ACOS(0.25);`
ASIN(x)	求反正弦值（单位为弧度），x 为一个数值	`SELECT ASIN(0.25);`
ATAN(x)	求反正切值（单位为弧度），x 为一个数值	`SELECT ATAN(2.5);`
ATAN2(n, m)	求反正切值（单位为弧度）	`SELECT ATAN2(-0.8, 2);`
AVG(expression)	返回一个表达式的平均值，expression 是一个字段	返回 Products 表中Price 字段的平均值： `SELECT AVG(Price) AS AveragePrice FROM Products;`
CEIL(x)	返回大于或等于 x 的最小整数	`SELECT CEIL(1.5) -- 返回2`
CEILING(x)	返回大于或等于 x 的最小整数	`SELECT CEILING(1.5); -- 返回2`

COS(x)	求余弦值(参数是弧度)	`SELECT COS(2);`
COT(x)	求余切值(参数是弧度)	`SELECT COT(6);`
COUNT(expression)	返回查询的记录总数，expression 参数是一个字段或者 * 号	返回 Products 表中 products 字段总共有多少条记录： `SELECT COUNT(ProductID) AS NumberOfProducts FROM Products;`
DEGREES(x)	将弧度转换为角度	`SELECT DEGREES(3.1415926535898) -- 180`
n DIV m	整除，n 为被除数，m 为除数	计算 10 除于 5： `SELECT 10 DIV 5;  -- 2`
EXP(x)	返回 e 的 x 次方	计算 e 的三次方： `SELECT EXP(3) -- 20.085536923188`
FLOOR(x)	返回小于或等于 x 的最大整数	小于或等于 1.5 的整数： `SELECT FLOOR(1.5) -- 返回1`
GREATEST(expr1, expr2, expr3, ...)	返回列表中的最大值	返回以下数字列表中的最大值： `SELECT GREATEST(3, 12, 34, 8, 25); -- 34`

		返回以下字符串列表中的最大值：
		`SELECT GREATEST("Google", "Runoob", "Apple");`
		`-- Runoob`

LEAST(expr1, expr2, expr3, ...)	返回列表中的最小值	返回以下数字列表中的最小值：
		`SELECT LEAST(3, 12, 34, 8, 25);`
		`-- 3`
		返回以下字符串列表中的最小值：
		`SELECT LEAST("Google", "Runoob", "Apple");  -- Apple`

LN	返回数字的自然对数，以 e 为底。	返回 2 的自然对数： `SELECT LN(2);  -- 0.6931471805599453`
LOG(x) 或 LOG(base, x)	返回自然对数(以 e 为底的对数)，如果带有 base 参数，则 base 为指定带底数。	`SELECT LOG(20.085536923188) -- 3` `SELECT LOG(2, 4); -- 2`
LOG10(x)	返回以 10 为底的对数	`SELECT LOG10(100) -- 2`
LOG2(x)	返回以 2 为底的对数	返回以 2 为底 6 的对数： `SELECT LOG2(6);  -- 2.584962500721156`

MAX(expression)	返回字段 expression 中的最大值	返回数据表 Products 中字段 Price 的最大值： `SELECT MAX(Price) AS LargestPrice FROM Products;`
MIN(expression)	返回字段 expression 中的最小值	返回数据表 Products 中字段 Price 的最小值： `SELECT MIN(Price) AS MinPrice FROM Products;`
MOD(x,y)	返回 x 除以 y 以后的余数	5 除于 2 的余数： `SELECT MOD(5,2) -- 1`
PI()	返回圆周率(3.141593)	`SELECT PI() --3.141593`
POW(x,y)	返回 x 的 y 次方	2 的 3 次方： `SELECT POW(2,3) -- 8`
POWER(x,y)	返回 x 的 y 次方	2 的 3 次方： `SELECT POWER(2,3) -- 8`
RADIANS(x)	将角度转换为弧度	180 度转换为弧度： `SELECT RADIANS(180) -- 3.1415926535898`

RAND()	返回 0 到 1 的随机数	`SELECT RAND() --0.93099315644334`

ROUND(x [,y])	返回离 x 最近的整数，可选参数 y 表示要四舍五入的小数位数，如果省略，则返回整数。	`SELECT ROUND(1.23456) --1`
		
		`SELECT ROUND(345.156, 2) --345.16`

SIGN(x)	返回 x 的符号，x 是负数、0、正数分别返回 -1、0 和 1	`SELECT SIGN(-10) -- (-1)`
SIN(x)	求正弦值(参数是弧度)	`SELECT SIN(RADIANS(30)) -- 0.5`
SQRT(x)	返回x的平方根	25 的平方根： `SELECT SQRT(25) -- 5`
SUM(expression)	返回指定字段的总和	计算 OrderDetails 表中字段 Quantity 的总和： `SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;`
TAN(x)	求正切值(参数是弧度)	`SELECT TAN(1.75);  --5.52037992250933`
TRUNCATE(x,y)	返回数值 x 保留到小数点后 y 位的值（与 ROUND 最大的区别是	`SELECT TRUNCATE(1.23456,3) --`

	不会进行四舍五入)	`1.234`

日期函数:

函数名	描述	实例
ADDDATE(d,n)	计算起始日期 d 加上 n 天的日期	`SELECT ADDDATE("2017-06-15", INTERVAL 10 DAY);` `->2017-06-25`

ADDTIME(t,n)	n 是一个时间表达式，时间 t 加上时间表达式 n	加 5 秒:
		`SELECT ADDTIME('2011-11-11 11:11:11', 5);`
		`->2011-11-11 11:11:16 (秒)`
		添加 2 小时, 10 分钟, 5 秒:
		`SELECT ADDTIME("2020-06-15 09:34:21", "2:10:5");`
		`-> 2020-06-15 11:44:26`

CURDATE()	返回当前日期	`SELECT CURDATE();` `-> 2018-09-19`
CURRENT_DATE()	返回当前日期	`SELECT CURRENT_DATE();` `-> 2018-09-19`

函数名	描述	实例
CURRENT_TIME	返回当前时间	SELECT CURRENT_TIME(); -> 19:59:02
CURRENT_TIMESTAMP()	返回当前日期和时间	SELECT CURRENT_TIMESTAMP() -> 2018-09-19 20:57:43
CURTIME()	返回当前时间	SELECT CURTIME(); -> 19:59:02
DATE()	从日期或日期时间表达式中提取日期值	SELECT DATE("2017-06-15"); -> 2017-06-15
DATEDIFF(d1,d2)	计算日期 d1->d2 之间相隔的天数	SELECT DATEDIFF('2001-01-01','2001-02-02') -> -32
DATE_ADD(d，INTERVAL expr type)	计算起始日期 d 加上一个时间段后的日期，type 值可以是： MICROSECOND SECOND MINUTE HOUR DAY WEEK MONTH QUARTER	SELECT DATE_ADD("2017-06-15", INTERVAL 10 DAY); -> 2017-06-25 SELECT DATE_ADD("2017-06-15 09:34:21", INTERVAL 15 MINUTE); -> 2017-06-15 09:49:21 SELECT DATE_ADD("2017-06-15 09:34:21", INTERVAL -3 HOUR);

函数名	描述	实例
	YEAR SECOND_MICROSECOND MINUTE_MICROSECOND MINUTE_SECOND HOUR_MICROSECOND HOUR_SECOND HOUR_MINUTE DAY_MICROSECOND DAY_SECOND DAY_MINUTE DAY_HOUR YEAR_MONTH	->2017-06-15 06:34:21 SELECT DATE_ADD("2017-06-15 09:34:21", INTERVAL -3 MONTH); ->2017-03-15 09:34:21
DATE_FORMAT(d,f)	按表达式 f的要求显示日期 d	SELECT DATE_FORMAT('2011-11-11 11:11:11','%Y-%m-%d %r') -> 2011-11-11 11:11:11 AM
DATE_SUB(date,INTERVAL expr type)	函数从日期减去指定的时间间隔。	Orders 表中 OrderDate 字段减去 2 天: SELECT OrderId,DATE_SUB(OrderDate,INTERVAL 2 DAY) AS OrderPayDate FROM Orders
DAY(d)	返回日期值 d 的日期部分	SELECT DAY("2017-06-15"); -> 15

函数名	描述	实例
DAYNAME(d)	返回日期 d 是星期几，如 Monday,Tuesday	SELECT DAYNAME('2011-11-11 11:11:11') ->Friday
DAYOFMONTH(d)	计算日期 d 是本月的第几天	SELECT DAYOFMONTH('2011-11-11 11:11:11') ->11
DAYOFWEEK(d)	日期 d 今天是星期几，1 星期日，2 星期一，以此类推	SELECT DAYOFWEEK('2011-11-11 11:11:11') ->6
DAYOFYEAR(d)	计算日期 d 是本年的第几天	SELECT DAYOFYEAR('2011-11-11 11:11:11') ->315
EXTRACT(type FROM d)	从日期 d 中获取指定的值，type 指定返回的值。 type可取值为： MICROSECOND SECOND MINUTE HOUR DAY WEEK MONTH QUARTER YEAR	SELECT EXTRACT(MINUTE FROM '2011-11-11 11:11:11') -> 11

函数名	描述	实例
	SECOND_MICROSECOND MINUTE_MICROSECOND MINUTE_SECOND HOUR_MICROSECOND HOUR_SECOND HOUR_MINUTE DAY_MICROSECOND DAY_SECOND DAY_MINUTE DAY_HOUR YEAR_MONTH	
FROM_DAYS(n)	计算从 0000 年 1 月 1 日开始 n 天后的日期	`SELECT FROM_DAYS(1111)` `-> 0003-01-16`
HOUR(t)	返回 t 中的小时值	`SELECT HOUR('1:2:3')` `-> 1`
LAST_DAY(d)	返回给给定日期的那一月份的最后一天	`SELECT LAST_DAY("2017-06-20");` `-> 2017-06-30`
LOCALTIME()	返回当前日期和时间	`SELECT LOCALTIME()` `-> 2018-09-19 20:57:43`
LOCALTIMESTAMP()	返回当前日期和时间	`SELECT LOCALTIMESTAMP()` `-> 2018-09-19 20:57:43`

函数名	描述	实例
MAKEDATE(year, day-of-year)	基于给定参数年份 year 和所在年中的天数序号 day-of-year 返回一个日期	`SELECT MAKEDATE(2017, 3);` `-> 2017-01-03`
MAKETIME(hour, minute, second)	组合时间，参数分别为小时、分钟、秒	`SELECT MAKETIME(11, 35, 4);` `-> 11:35:04`
MICROSECOND(date)	返回日期参数所对应的微秒数	`SELECT MICROSECOND("2017-06-20 09:34:00.000023");` `-> 23`
MINUTE(t)	返回 t 中的分钟值	`SELECT MINUTE('1:2:3')` `-> 2`
MONTHNAME(d)	返回日期当中的月份名称，如 November	`SELECT MONTHNAME('2011-11-11 11:11:11')` `-> November`
MONTH(d)	返回日期d中的月份值，1 到 12	`SELECT MONTH('2011-11-11 11:11:11')` `->11`
NOW()	返回当前日期和时间	`SELECT NOW()` `-> 2018-09-19 20:57:43`

函数名	描述	实例
PERIOD_ADD(period, number)	为 年-月 组合日期添加一个时段	SELECT PERIOD_ADD(201703, 5); -> 201708
PERIOD_DIFF(period1, period2)	返回两个时段之间的月份差值	SELECT PERIOD_DIFF(201710, 201703); -> 7
QUARTER(d)	返回日期d是第几季节，返回 1 到 4	SELECT QUARTER('2011-11-11 11:11:11') -> 4
SECOND(t)	返回 t 中的秒钟值	SELECT SECOND('1:2:3') -> 3
SEC_TO_TIME(s)	将以秒为单位的时间 s 转换为时分秒的格式	SELECT SEC_TO_TIME(4320) -> 01:12:00
STR_TO_DATE(string, format_mask)	将字符串转变为日期	SELECT STR_TO_DATE("August 10 2017", "%M %d %Y"); -> 2017-08-10
SUBDATE(d,n)	日期 d 减去 n 天后的日期	SELECT SUBDATE('2011-11-11 11:11:11', 1) ->2011-11-10 11:11:11 (默认是天)

函数名	描述	实例
SUBTIME(t,n)	时间 t 减去 n 秒的时间	SELECT SUBTIME('2011-11-11 11:11:11', 5) ->2011-11-11 11:11:06 (秒)
SYSDATE()	返回当前日期和时间	SELECT SYSDATE() -> 2018-09-19 20:57:43
TIME(expression)	提取传入表达式的时间部分	SELECT TIME("19:30:10"); -> 19:30:10
TIME_FORMAT(t,f)	按表达式 f 的要求显示时间 t	SELECT TIME_FORMAT('11:11:11','%r') 11:11:11 AM
TIME_TO_SEC(t)	将时间 t 转换为秒	SELECT TIME_TO_SEC('1:12:00') -> 4320

TIMEDIFF(time1, time2)	计算时间差值	mysql> SELECT TIMEDIFF("13:10:11", "13:10:10");
		-> 00:00:01
		mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
		    -> '2000:01:01 00:00:00.000001');
		        -> '-00:00:00.000001'
		mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',

函数名	描述	实例
		-> '2008-12-30 01:01:01.000002'); -> '46:58:57.999999'

TIMESTAMP(expression, interval)	单个参数时，函数返回日期或日期时间表达式；有2个参数时，将参数加和	mysql> SELECT TIMESTAMP("2017-07-23", "13:10:11");
		-> 2017-07-23 13:10:11
		mysql> SELECT TIMESTAMP('2003-12-31');
		-> '2003-12-31 00:00:00'
		mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
		-> '2004-01-01 00:00:00'

TIMESTAMPDIFF(unit,datetime_expr1 ,datetime_expr2)	计算时间差，返回 datetime_expr2 − datetime_expr1 的时间差	mysql> SELECT TIMESTAMPDIFF(DAY,'2003-02-01','2003-05-01'); // 计算两个时间相隔多少天
		-> 89
		mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01'); // 计算两个时间相隔多少月
		-> 3
		mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01'); // 计算两个时间相隔多少年
		-> -1
		mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55'); // 计

函数名	描述	实例
		算两个时间相隔多少分钟 -> 128885
TO_DAYS(d)	计算日期 d 距离 0000 年 1 月 1 日的天数	SELECT TO_DAYS('0001-01-01 01:01:01') -> 366
WEEK(d)	计算日期 d 是本年的第几个星期，范围是 0 到 53	SELECT WEEK('2011-11-11 11:11:11') -> 45
WEEKDAY(d)	日期 d 是星期几，0 表示星期一，1 表示星期二	SELECT WEEKDAY("2017-06-15"); -> 3
WEEKOFYEAR(d)	计算日期 d 是本年的第几个星期，范围是 0 到 53	SELECT WEEKOFYEAR('2011-11-11 11:11:11') -> 45
YEAR(d)	返回年份	SELECT YEAR("2017-06-15"); -> 2017
YEARWEEK(date, mode)	返回年份及第几周（0到53），mode 中 0 表示周天，1表示周一，以此类推	SELECT YEARWEEK("2017-06-15"); -> 201724

高级函数：

函数名	描述	实例
BIN(x)	返回 x 的二进制编码，x 为十进制数	15 的 2 进制编码: `SELECT BIN(15); -- 1111`
BINARY(s)	将字符串 s 转换为二进制字符串	`SELECT BINARY "RUNOOB";` `-> RUNOOB`

`CASE expression`
`WHEN condition1 THEN result1`
`WHEN condition2 THEN result2`
`...`
`WHEN conditionN THEN resultN`
`ELSE result`
`END`
	CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1，如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。	`SELECT CASE`
		`WHEN 1 > 0`
		`THEN '1 > 0'`
		`WHEN 2 > 0`
		`THEN '2 > 0'`
		`ELSE '3 > 0'`
		`END`
		`->1 > 0`

CAST(x AS type)	转换数据类型	字符串日期转换为日期:
		`SELECT CAST("2017-08-29" AS DATE);`
		`-> 2017-08-29`

COALESCE(expr1, expr2, ..., expr_n)	返回参数中的第一个非空表达式（从左向右）	`SELECT COALESCE(NULL, NULL, NULL, 'runoob.com',`

函数名	描述	实例
		NULL, 'google.com'); -> runoob.com
CONNECTION_ID()	返回唯一的连接 ID	SELECT CONNECTION_ID(); -> 4292835
CONV(x,f1,f2)	返回 f1 进制数变成 f2 进制数	SELECT CONV(15, 10, 2); -> 1111
CONVERT(s USING cs)	函数将字符串 s 的字符集变成 cs	SELECT CHARSET('ABC') ->utf-8 SELECT CHARSET(CONVERT('ABC' USING gbk)) ->gbk
CURRENT_USER()	返回当前用户	SELECT CURRENT_USER(); -> guest@%
DATABASE()	返回当前数据库名	SELECT DATABASE(); -> runoob
IF(expr,v1,v2)	如果表达式 expr 成立，返回结果 v1；否则，返回结果 v2。	SELECT IF(1 > 0,'正确','错误')

函数名	描述	实例
		->正确
IFNULL(v1,v2)	如果 v1 的值不为 NULL，则返回 v1，否则返回 v2。	SELECT IFNULL(null,'Hello Word') ->Hello Word
ISNULL(expression)	判断表达式是否为 NULL	SELECT ISNULL(NULL); ->1
LAST_INSERT_ID()	返回最近生成的 AUTO_INCREMENT 值	SELECT LAST_INSERT_ID(); ->6
NULLIF(expr1, expr2)	比较两个字符串，如果字符串 expr1 与 expr2 相等 返回 NULL，否则返回 expr1	SELECT NULLIF(25, 25); ->
SESSION_USER()	返回当前用户	SELECT SESSION_USER(); -> guest@%
SYSTEM_USER()	返回当前用户	SELECT SYSTEM_USER(); -> guest@%

函数名	描述	实例
USER()	返回当前用户	`SELECT USER();` `-> guest@%`
VERSION()	返回数据库的版本号	`SELECT VERSION()` `-> 5.6.34`

MySQL 8.0 版本新增的一些常用函数：

数	描述	实例
JSON_OBJECT()	将键值对转换为 JSON 对象	`SELECT JSON_OBJECT('key1', 'value1', 'key2', 'value2')`
JSON_ARRAY()	将值转换为 JSON 数组	`SELECT JSON_ARRAY(1, 2, 'three')`
JSON_EXTRACT()	从 JSON 字符串中提取指定的值	`SELECT JSON_EXTRACT('{"name": "John", "age": 30}', '$.name')`
JSON_CONTAINS()	检查一个 JSON 字符串是否包含指定的值	`SELECT JSON_CONTAINS('{"name": "John", "age": 30}', 'John', '$.name')`
ROW_NUMBER()	为查询结果中的每一行分配一个唯一的数字	`SELECT ROW_NUMBER() OVER(ORDER BY id) AS row_number, name FROM users`

数	描述	实例
RANK()	为查询结果中的每一行分配一个排名	`SELECT RANK() OVER(ORDER BY score DESC) AS rank, name, score FROM students`

运算符

算术运算符，用于计算。

运算符	作用
+	加法
-	减法
*	乘法
/ 或 DIV	除法
% 或 MOD	取余

在除法运算和模运算中，如果除数为0，将是非法除数，返回结果为NULL。

SELECT 语句中的条件语句经常要使用比较运算符。通过这些比较运算符，可以判断表中的哪些记录是符合条件的。比较结果为真，则返回 1，为假则返回 0，比较结果不确定则返回 NULL。

符号	描述	备注
=	等于	
<>, !=	不等于	
>	大于	
<	小于	

符号	描述	备注
<=	小于等于	
>=	大于等于	
BETWEEN	在两值之间	>=min&&<=max
NOT BETWEEN	不在两值之间	
IN	在集合中	
NOT IN	不在集合中	
<=>	严格比较两个NULL值是否相等	两个操作码均为NULL时，其所得值为1；而当一个操作码为NULL时，其所得值为0
LIKE	模糊匹配	
REGEXP 或 RLIKE	正则式匹配	
IS NULL	为空	
IS NOT NULL	不为空	

逻辑运算符用来判断表达式的真假。如果表达式是真，结果返回 1。如果表达式是假，结果返回 0。

运算符号	作用
NOT 或 !	逻辑非
AND	逻辑与
OR	逻辑或
XOR	逻辑异或

位运算符是在二进制数上进行计算的运算符。位运算会先将操作数变成二进制数，进行位运算。然后再将计算结果从二进制数变回十进制数。

运算符号	作用

&	按位与
|	按位或
^	按位异或
!	取反
<<	左移
>>	右移

命令大全

基础命令

操作	命令
连接到 MySQL 数据库	mysql -u 用户名 -p
查看所有数据库	SHOW DATABASES;
选择一个数据库	USE 数据库名;
查看所有表	SHOW TABLES;
查看表结构	DESCRIBE 表名; 或 SHOW COLUMNS FROM 表名;
创建一个新数据库	CREATE DATABASE 数据库名;
删除一个数据库	DROP DATABASE 数据库名;
创建一个新表	CREATE TABLE 表名 (列名1 数据类型 [约束], 列名2 数据类型 [约束], ...);
删除一个表	DROP TABLE 表名;
插入数据	INSERT INTO 表名 (列1, 列2, ...) VALUES (值1, 值2, ...);
查询数据	SELECT 列1, 列2, ... FROM 表名 WHERE 条件;

操作	命令
更新数据	UPDATE 表名 SET 列1 = 值1, 列2 = 值2, ... WHERE 条件;
删除数据	DELETE FROM 表名 WHERE 条件;
创建用户	CREATE USER '用户名'@'主机' IDENTIFIED BY '密码';
授权用户	GRANT 权限 ON 数据库名.* TO '用户名'@'主机';
刷新权限	FLUSH PRIVILEGES;
查看当前用户	SELECT USER();
退出 MySQL	EXIT;

下面是与 MySQL 数据库操作相关的命令，包括创建、删除和修改数据库等操作：

操作	命令
创建数据库	CREATE DATABASE 数据库名;
删除数据库	DROP DATABASE 数据库名;
修改数据库编码格式和排序规则	ALTER DATABASE 数据库名 DEFAULT CHARACTER SET 编码格式 DEFAULT COLLATE 排序规则;
查看所有数据库	SHOW DATABASES;
查看数据库详细信息	SHOW CREATE DATABASE 数据库名;
选择数据库	USE 数据库名;
查看数据库的状态信息	SHOW STATUS;
查看数据库的错误信息	SHOW ERRORS;
查看数据库的警告信息	SHOW WARNINGS;
查看数据库的表	SHOW TABLES;

操作	命令
查看表的结构	DESC 表名; DESCRIBE 表名; SHOW COLUMNS FROM 表名; EXPLAIN 表名;
创建表	CREATE TABLE 表名 (列名1 数据类型 [约束], 列名2 数据类型 [约束], ...);
删除表	DROP TABLE 表名;
修改表结构	ALTER TABLE 表名 ADD 列名 数据类型 [约束]; ALTER TABLE 表名 DROP 列名; ALTER TABLE 表名 MODIFY 列名 数据类型 [约束];
查看表的创建 SQL	SHOW CREATE TABLE 表名;

以下是与 MySQL 数据表相关的常用命令，包括创建、修改、删除表以及查看表的结构和数据等操作：

操作	命令
创建表	CREATE TABLE 表名 (列名1 数据类型 [约束], 列名2 数据类型 [约束], ...);
删除表	DROP TABLE 表名;
修改表结构	**添加列:** ALTER TABLE 表名 ADD 列名 数据类型 [约束]; **删除列:** ALTER TABLE 表名 DROP 列名; **修改列:** ALTER TABLE 表名 MODIFY 列名 数据类型 [约束]; **重命名列:** ALTER TABLE 表名 CHANGE 旧列名 新列名 数据类型 [约束];
查看表结构	DESC 表名; DESCRIBE 表名; SHOW COLUMNS FROM 表名; EXPLAIN 表名;
查看表的创建 SQL	SHOW CREATE TABLE 表名;
查看表中的所有数据	SELECT * FROM 表名;

操作	命令
插入数据	INSERT INTO 表名 (列1, 列2, ...) VALUES (值1, 值2, ...);
更新数据	UPDATE 表名 SET 列1 = 值1, 列2 = 值2, ... WHERE 条件;
删除数据	DELETE FROM 表名 WHERE 条件;
查看表的索引	SHOW INDEX FROM 表名;
创建索引	CREATE INDEX 索引名 ON 表名 (列名);
删除索引	DROP INDEX 索引名 ON 表名;
查看表的约束	SHOW CREATE TABLE 表名; (约束信息会包含在创建表的 SQL 中)
查看表的统计信息	SHOW TABLE STATUS LIKE '表名';

以下是与 MySQL 事务相关的常用命令：

操作	命令
开始事务	START TRANSACTION; 或 BEGIN;
提交事务	COMMIT;
回滚事务	ROLLBACK;
查看当前事务的状态	SHOW ENGINE INNODB STATUS; (可查看 InnoDB 存储引擎的事务状态)
锁定表以进行事务操作	LOCK TABLES 表名 WRITE; 或 LOCK TABLES 表名 READ;
释放锁定的表	UNLOCK TABLES;
设置事务的隔离级别	SET TRANSACTION ISOLATION LEVEL READ COMMITTED; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;