#MLorDS

```
SYSTEM CALL
GENERATE THERMAL ELEMENT
FORM ELEMENT ARROW SHAPE
DISCHARGE
```

以上是一段来自动漫'Sword Art Online：Alicization'的咒语，这种咒语类似自然语言，因此容易猜到其含义
根据观察，SQL 语句的使用难度和念出动漫中这种咒语的难度差不多（在出题老师不抽风的前提下！）

前置代码，请引入到任何学习使用 mySQL 的笔记本中，并首先执行一次

PYTHON

```python
import pymysql
database = pymysql.connect(host = 'localhost',user = 'root',password =
'030329',database = "test_db")
cursor = database.cursor() # 创建用于管理数据库的游标
```

后置代码注射，请引入到任何学习使用 mySQL 的笔记本中

```
cursor.execute(sql)
all_update = cursor.fetchall()
print("-----------FINISH-----------")
for item in all_update:
    print(item)
```

ChatGPT ， GPT：如果在文段中有此标识，则代表这一段文字由 ChatGPT 生成，需要仔细甄别！
由于 Obsidian 的 Excute Code 插件没有很好地完成，不要试图在笔记本中执行本文中所有 SQL 语句

SQL 初级咒语

模式和表格的建立

SQL

```
CREATE SCHEMA <模式名>
AUTHORIZATION <用户名>
```

定义一个模式相当于重新建立了一个数据仓库，相当于在本笔记软件 (Obsidian)中新建了一个 Vault
模式名可以不写，缺省值为用户名。

ChatGPT 在 SQL 中，SCHEMA（模式）是指将数据库中的表、视图、存储过程和其他对象组织在一起的逻辑容器。SCHEMA 提供了一种组织和管理数据库对象的方式，以便更好地管理和维护数据库。每个 SCHEMA 都有一个名称，并且可以包含多个表、视图、存储过程和其他对象。它可以用于将具有共同主题或功能的对象分组在一起，并在数据库中为其命名空间。在查询中引用 SCHEMA 中的表时，需要在表名前加上 SCHEMA 的名称。

```SQL
CREATE TABLE <模式>.<表格>
(
Sno CHAR(6) NOT NULL,
Cno CHAR(6) NOT NULL,
GRADE INT CHECK (GRADE BETWEEN 0 AND 100),
PRIMARY KEY (Sno,Cno), --添加主键约束
CONSTRAINT SC_FK1 FOREIGN KEY (Sno) REFERENCES Student(Sno), -- 指定本表参考外表
);
```

创建表格需要先使用主句 `CREATE TABLE`，此后的每条子句可以是

- **`<变量名><变量类型><约束类型>`**
- **`CONSTRAINT <约束名><约束类型><约束描述>`**，其中，约束名可以不写，如果是主键约束，`CONSTRAINT` 关键字可以省略

ChatGPT在 SQL 中，可以使用的约束条件包括以下：

- `PRIMARY KEY`：将列标记为主键，使其在表格中具有唯一性并用于链接其他表格。
- `FOREIGN KEY ( ) REFERENCE ( )`：创建外键约束，将该列链接到其他表格中的主键。
- `UNIQUE`：使列的值保持唯一，但不像主键一样强制列为非空。
- `NOT NULL`：确保列中没有 NULL 值。
- `DEFAULT`：为列指定默认值，如果未指定值，则使用默认值。
- `CHECK`：创建一个条件来确保列的值满足特定的条件。
- `INDEX`：创建一个索引以加快对该列的搜索。

表级修改

使用以下的语句给表格增加一列：

```SQL
ALTER TABLE <表格>
ADD <列名> <变量类型>
```

新添加的列将被强制设置为空值。此外，GPT你还可以修改新添加的列的特性

```SQL
ALTER TABLE employees
ALTER COLUMN gender
SET NOT NULL;
```

这使得表格中“gender”一列的值不可为空值

GPT想要删除表格中的一列时，你应当使用如下的语句：

```SQL
ALTER TABLE <表格>
DROP COLUMN <列名>
```

总之，你应该使用 `TABLE` 来调用某一张表，使用 `COLUMN` 来调用某一列

查询指定列、筛选和排序

你应该使用 `SELECT` 语句来启动一次查询
使用以下语句来查询某一列中的内容，并按照指定的条件筛选：

```SQL
SELECT <列名>
FROM <表格>
WHERE <条件>
```

如果你需要排序，你应该在上面的语句后面加上 `ORDER BY <要求>`，其中，`<要求>` 可以选择 `ASC`（升序排列，缺省值）或者 `DESC`（降序排列）

字符串匹配

在查询条件中，你可以使用 `LIKE` 进行字符串匹配，`LIKE` 后面可以接以下格式：

- `<字符串>` 代表完全匹配整个字符串，此时，`LIKE` 可以换成 `=`
- `_` 代表字符串中的**一个**字符可以是任意的
- `%` 代表字符串中的**任意个**字符可以是任意的
- 如果字符串中已经含有两种通配符，但是你并不想启用通配符，那么你应当使用 `ESCAPE` 子句声明一种转义符进行转义，例如：

```SQL
SELECT Cno FROM Course
WHERE Cname LIKE 'ab\_c' ESCAPE '\'
```

这段语句中，我们将反斜线声明为转义符，跟在它后面的下划线不再具有通配符的含义，而是作为一个真实的字符进行匹配

分组与聚合

使用 `GROUP BY` 语句可以将数据库按照一个列或者多个列进行分组，使用聚合函数可以计算每一组的汇总值。例如：

```SQL
SELECT <列名1>, <列名2>, <聚合函数> (列名3)
FROM <表格>
GROUP BY <列名1>, <列名2>
```

其中，`<列名1>, <列名2>` 为你想要分组的列，`<聚合函数>` 为你想要使用的聚合函数。注意：在执行分组后，你仅可以选出分组所使用的列和被聚合函数处理后的列。可以使用的聚合函数包括：

- `SUM` 求和
- `AVG` 均值
- `MAX,MIN` 最大值最小值
- `COUNT` 计数

注意，有些聚合函数含有参数 `DISTINCT|ALL`，这意味着你要对每组中的所有行（元组）进行聚合，还是只对不重复的元组聚合。例如，`COUNT(DISTINCT col1)` 意味着仅计数 'col 1'中不重复的元素

要想对**分组后的各个组**进行筛选，你应当在 `GROUP BY` 之后使用 `HAVING` 子句，其用法和 `WHERE` 子句几乎一致，唯一不同的是，在 `HAVING` 子句中，你可以通过聚合函数来提出要求；如果你想事先对表中的各个行进行筛选，以确定哪些行能够参与分组和聚合，你应当在 `GROUP BY` 之前使用 `WHERE` 子句

连接查询

本节中含有大量 GPT 生成内容
之前，我们使用这样的方式来连接两张表：

```SQL
SELECT Student.* ,SC.* FROM STUDENT,SC
WHERE Student.Sno = SC.Sno
```

这种操作叫做隐式内连接 (Implicit Inner Join)，是不被提倡的。现在，我们使用 `JOIN` 子句来连接两张表。

例如，上面这段代码应该被改写为：

```SQL
SELECT *
FROM Student
JOIN SC
ON Student.Sno = Sc.Sno
```

如果想要对三张表做自然连接，你可以直接这样写：

```SQL
SELECT *
FROM  Student
NATURAL JOIN SC,Course
```

`JOIN` 子句的用法如下：

- 等值连接：在 `JOIN` 后面添加 `ON` 子句，指明连接条件
- 自然连接：使用 `NATURAL JOIN` 子句
- 内连接：将两个表中符合连接条件的行连接在一起，只保留满足连接条件的行。如果其中一个表中没有符合条件的行，则该行将不会出现在连接结果中，使用 `INNER JOIN` 子句
- 左外连接：将两个表中符合连接条件的行连接在一起，并保留左表中不符合连接条件的行。如果右表中没有符合条件的行，则在连接结果中填充 NULL 值，使用 `LEFT JOIN` 子句
- 右外连接：同理，使用 `RIGHT JOIN` 子句
- 全外连接：将两个表中所有行连接在一起，并保留不符合连接条件的行，相当于左外连接和右外连接的总和。如果其中一个表中没有符合条件的行，则在连接结果中填充 NULL 值，使用 `FULL OUTER JOIN` 子句
- 交叉连接：将两个表中的所有行进行组合，形成笛卡尔积。交叉连接不需要指定连接条件，但会返回两个表的所有可能组合，使用 `CROSS JOIN` 子句
- 自身连接：这常常被用于查找特殊的信息。例如，有一张表"employee"，其中包含有员工的姓名和上级，那么可以使用自身连接的方式查找到每位员工上级的名字：

```SQL
SELECT e1.name AS employee_name, e2.name AS manager_name
FROM employee e1
JOIN employee e2
ON e1.manager_id = e2.id;
```

这样的操作根据每位员工的上级 id，找到其上级，如果上级存在，则连接就会进行。

增删改

数据插入

要插入几条数据，请使用：

```SQL
INSERT INTO <表格>(<列名>)
VALUES (<待插入的元组>)
```

要插入子查询的结果，请使用

```SQL
INSERT INTO <表格>(<列名>)
SELECT <查询语句>
```

数据修改

要修改数据，请使用 `UPDATE`。注意：只有在做表格级别的修改（修改一列的整体属性）、删除时才会使用 `ALTER`

```SQL
UPDATE <列表>
SET <修改内容>
WHERE <待修改元组的查询条件>
```

数据删除

要删除数据，请使用：

```SQL
DELETE FROM <列表>
WHERE <待删除元组的查询条件>
```

视图创建和查询

CHAT-GPT 在 SQL（结构化查询语言）中，视图（View）是一个虚拟表，它是一个基于 SQL 查询的结果集。视图并不存储实际的数据，而是根据定义它的查询语句从一个或多个基础表（Base Table）中实时计算和显示数据。视图可以像普通表一样进行查询、插入、更新和删除操作（但这些操作受到一定限制）

视图的创建实质上是一种查询语句

```SQL
CREATE VIEW <视图> AS
SELECT <查询语句>
(可选地) WITH CHECK OPTION
```

如果你使用了子句 `WITH CHECK OPTION`，那么通过视图插入、修改数据时，会先检查你的操作是否符合视图的要求，如果不符合就不能操作。例如，你希望有一张视图保存 18 岁以上的学生，使用该子句后，你不能通过这张视图插入 18 岁以下的学生。

视图可以被嵌套地定义，可以通过在两张连接的表上查询来定义。
要想删除视图，使用与删除表格相同的方式：

```SQL
DROP VIEW <视图> (可选地)CASCADE
```

如果使用了 `CASCADE` 子句，那么视图将级联地被删除

你可以通过视图完成任何表格可以完成的事情，包括但不限于：

- 在视图上查询
- 通过视图修改信息（不能修改视图看不到的元组，注意 `WITH CHECK OPTION` 带来的限制）
- 通过视图删除元组（不能删除视图看不到的元组）
- 通过视图添加元组（注意 `WITH CHECK OPTION` 的限制）

授权和回收

如果你要授权，使用如下语句：

```SQL
GRANT <权限>
ON <对象>
TO <用户>
(可选地)WITH GRANT OPTION
```

其中，`WITH GRANT OPTION` 意味着被授权的人可以将此权限向下传递。你可以仅允许用户更改表中的某一列，例如

```SQL
GRANT SELECT(Sname)
ON Student
TO User1
```

回收权限可以使用类似的语句

```SQL
REVOKE <权限>
ON <对象>
FROM <用户>
(可选地) CASCADE/RESTRICT
```

其中，`CASCADE` 子句表示“级联”，可以在收回该用户权限的同时，收回被他授权的所有用户的权限；`RESTRICT` 则会在收回权限时进行检查，如果该用户已经将权限传递给了其他用户，则不会收回该用户的权限。

SQL 高级咒语

嵌套查询

嵌套查询分为两种，相关子查询和不相关子查询。对于不相关子查询，就是相当于查询了两次，这可以通过直接改写 `WHERE` 语句后面判定语句的目标来实现：

```SQL
SELECT Sname FROM Student
WHERE Sno IN
(
SELECT Sno FROM SC
WHERE Cno = '100002'
)
```

这段代码先找到了选择课程'100002'的学生编号，再根据这些编号查出了学生名字

对于相关子查询，相当于有多重循环在运行。DBMS 先从外层循环中取出一个元组，将其参数传入内层循环，内层循环根据传入的参数找到符合条件的元组，再将内层结果传出到外层循环，外层循环根据收到的参数判断当前的元组是否符合条件，如果符合，则输出。例如，下面的代码给出了查询超过平均分的学生的学号：

```SQL
SELECT Sno FROM SC AS x
WHERE x.Grade >
(
SELECT AVG(Grade) FROM SC AS y
WHERE x.Cno = y.Cno
)
```

可以看到，外层循环将每位学生选择的课号传给内层，内层查出该课程所有选课记录并计算平均成绩，再将平均成绩传给外层。外层循环再判断每个元组是否符合要求

逻辑谓词

`IN` 谓词：在集合中

例如，使用嵌套查询查出选择了'Algo'这门课的学生姓名：

```SQL
SELECT Sname FROM Student AS x
WHERE x.Sno IN
(
    SELECT Sno FROM SC As y
    WHERE y.Cno IN
    (
        SELECT Cno FROM Course AS z
        WHERE z.Cname  = 'Algo'
    )
)
```

当然，你也可以使用 `JOIN` 子句完成这个操作：

```SQL
SELECT Sname FROM Student
NATURAL JOIN SC,Course
WHERE Course.Cname = 'Algo'
```

`ANY` 谓词：集合中任意元素

例如，查询比计算机系学生任意一位的年龄更小的学生

```SQL
SELECT Sname FROM Student AS x
WHERE x.Sage < ANY
(
SELECT Sage FROM Student AS y
WHERE y.dept = 'CS'
)
```

`EXISTS` 与 `NOT EXISTS` 谓词：判断条件是否成立

例如，查询选择了 '100001'课程的学生

```SQL
SELECT Sname FROM Student AS x
WHERE EXIST
(
SELECT * FROM SC AS y
WHERE y.Sno = x.Sno AND y.Cno = '100001'
)
```

`EXISTS` 和 `NOT EXISTS` 可以用于组成全称量词 FORALL

例如，查询选修了所有课的学生→查询不存在未选修的课的学生

```SQL
SELECT Sname FROM Student AS x
WHERE NOT EXISTS  --学生不存在没上的课，这一层枚举每个学生
(
    SELECT * FROM Course AS y
    NOT EXISTS --匹配该学生没上的课，这一层枚举所有课
    (
        SELECT * FROM SC AS z
        WHERE z.Cno = y.Cno and x.Sno = z.Sno --判断这门课这个学生有没有上过
    )
)
```

集合运算

- 想要并两个集合，请在两次查询之间插入 `UNION` （SQL 提供该子句）
- 想要交两个集合，如果两次查询的属性不同，请使用 `AND` 逻辑运算符；如果两次查询的属性相同，请用子查询实现
  例如，查询同时选择了两个课的学生

```SQL
SELECT * FROM SC AS x
WHERE x.Cno = '100001' AND x.Sno IN
(
SELECT Sno FROM SC AS y
WHERE y.Cno = '100002'
)
```

- 想要求两个集合的差，请使用子查询
  例如，查询选了课程'100001'但没有选择'100002'的学生

```SQL
SELECT * FROM SC AS x
WHERE x.Cno = '100001' AND x.Sno NOT IN
(
SELECT Sno FROM SC AS y
WHERE y.Cno = '100002'
)
```