SQL 入门

这部分内容主要介绍 MySQL 数据库的 **基本使用方法**

Tables & Databases

Concepts

关系型数据库

~~简单来说关系型数据库就是几张关联起来的二维表。~~

在本节示例中，我们 **随机生成** 了这样几张表

- 学生列表，表 `students`

编号	姓名	注册时间	入学年份	地区
0	William Villeneuve	1656871020	2020	Germany
1	Reginald Stitt	1656873247	2013	Indonesia
2	Mamie Boney	1656857580	2021	Zambia

- 课程列表，表 `courses`

编号	名称	教师	学分
6306068993	信号处理原理	Christina Elms	2
6642366221	编译原理	Martin Volante	4
8028895797	计算机组成原理	Mark Butler	1
2586295528	人工神经网络	Benito Davila	1
2442847225	数据库系统概论	William Terrell	1

- 学生选课情况，表 `association`

编号	学生编号	课程编号	成绩
681	47	4949903882	A-
682	47	9328332090	W
683	47	4085573588	B+
684	47	3748453943	D+

`students` 和 `courses` 本身没有直接关联，但 `association` 提供了将 `student` 的编号 与 `courses` 编号的关联，从而我们可以知道哪些学生选了哪些课程这样的信息、每门课程有哪些学生选课、以选课学生的成绩等信息。

基本概念

> SQL (Structured Query Language)，即结构化查询语言，是关系型数据库操作的利器。
>
> SQL 也是一种编程语言，只是用途非常的特别。SQL本身有 ANSI 标准，不同数据库引擎会为自己的数据库而做出一些语言上的调整，但是核心部分是一致的，这也是本课程的关注点。
>
> SQL 在现今是直接或间接管理数据库必不可少的语言，常用于网站后端（非静态站点），而 SQL 写得好与坏也可以造成相同硬件配置下极大的性能差异。[1]

SQL 特性预警

- 脚本语言。可以交互执行也可以事先写好后一次性执行
- 弱类型。如

```
mysql> SELECT 1 + "1";
+---------+
| 1 + "1" |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)
```

- 关键字不区分大小写；表名、数据库名、字段名等区分大小写。如

```
mysql> cREAtE dAtAbAsE hello;
Query OK, 1 row affected (0.01 sec)

mysql> cREAtE dAtAbAsE Hello;
Query OK, 1 row affected (0.02 sec)
```

  创建了两个名为 `Hello` 和 `hello` 的**两个**数据库。

DataTypes

在这里介绍一些常用的

- CHAR：定长字符串，长度取值范围 $[0, 255]$ **字符**
- VARCHAR：变长字符串，长度取值范围 $[0, 65535]$ **字符**
- INT：$[-2^{31}, 2^{31}-1]$
- BIGINT：$[-2^{63}, 2^{63}-1]$

  > 1秒增加1e8条记录，则再经过2924年后才会溢出 \doge

- FLOAT
- DOULBE
- DATETIME，DATE

// 在存储日期时间时，使用 时间戳 通常比使用 DATE, DATETIME 更加方便

特别地，SQL 用 `NULL` 表示空值。

MySQL 数据库和数据表操作

CREATE

创建新的数据库，然后使用它

```
mysql> CREATE DATABASE summer;
Query OK, 1 row affected (0.02 sec)
mysql> USE summer;
Database changed
```

`CREATE TABLE` 语句用于创建新的数据表，下面我们创建今天用到的三张表

使用 `CREATE TABLE` 语句创建表

```
CREATE TABLE students (
    id BIGINT NOT NULL,
    name VARCHAR(255),
    created_at DOUBLE,
    enroll INT,
    region VARCHAR(50)
);

CREATE TABLE courses (
    id CHAR(10) NOT NULL,
    title VARCHAR(255),
    teacher VARCHAR(255),
    credit INT
);

CREATE TABLE association (
    id BIGINT NOT NULL,
    sid BIGINT NOT NULL,
    cid BIGINT NOT NULL,
    grade CHAR(2)
);
```

`CREATE TABLE` 语句中，还可以给字段制定约束、字符集等操作，在此处略去（~~由于我经常用软件直接建表所以这些语句也不太熟~~）

查看当前数据库中的所有表

```
mysql> SHOW TABLES;
```

可以使用 `DESC` 语句 查看表的字段情况

```
DESC users;
```

ALTER

ALTER 语句可以用于给数据库添加、删除字段，修改字段的类型或属性，下面举 3 个简单的例子。

```
ALTER TABLE table_name
ADD column_name datatype; -- 添加字段

ALTER TABLE table_name
DROP COLUMN column_name;  -- 删除字段

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;  -- 修改字段类型
```

DROP

删表

```
mysql> DROP TABLE students;
Query OK, 0 rows affected (0.05 sec)
```

删库

```
mysql> DROP DATABASE summer;
Query OK, 0 rows affected (0.02 sec)
```

你看，库没咯!

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

CRUD

Wikipedia

> In computer programming, **create, read, update, and delete** (**CRUD**) are the four basic operations of persistent storage.

INSERT INTO

假设我们创建了上述三张表。我们先使用 `INSERT INTO` 语句向这空表中加些数据。

可以 使用创建时指定的顺序添加字段

```
INSERT INTO `students` VALUES (9999, 'Roscoe Hirt', 1346799650, 2012, 'Aruba');
```

也可以指定字段名

```
INSERT INTO `students`(id, name, region) VALUES (9999, 'Roscoe Hirt', 'Aruba');
```

没有指定值的字段将默认被分配 `NULL` 或者 `DEFAULT` 值，如果字段有 `NOT NULL` 属性，则会引发错误

也可以一次性插入许多记录

```
INSERT INTO `students`(id, name, region) VALUES
    (9999, 'Roscoe Hirt', 'Aruba'),
    (10000, 'Roscoe Hirt ++', 'Aruba ++');
```

这时我们不妨插入一些预置的数据，便于我们继续学习。

先退到你的 shell

```
curl -fsSL https://cloud.tsinghua.edu.cn/f/68b8862a97e1475a800f/?dl=1 -o
summer.sql
```

然后进入 MySQL 命令行

```
source <file path>;
```

即可执行脚本，导入一些预先随机生成的学生、课程和选课记录。

SELECT

SELECT 是从数据库中取用数据的核心操作。本节介绍 SELECT 的一些常用语法。

https://dev.mysql.com/doc/refman/8.0/en/select.html

首先我们来看 SELECT 的全部语法

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
```

```
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

很长，有很多可选操作，但其中必备的只有

```
SELECT select_expr
```

这一项；

基本操作

暴力地拿出整张表

```
SELECT * FROM courses;
```

如果这张表很大，或者某个字段特别长，可以想到这是一个很离谱的操作。

我们可以指定查看的列，

```
SELECT id, title FROM courses;
```

这里，说明一下对字段的表达，严格来说，表名、字段名这些 literal 应该用反引号包起来，且列名应该指定对应的表名，如上述查询可以写做

```
SELECT `courses`.`id`, `courses`.`title` FROM `courses`;
```

但在不引起歧义的情况下可以并且在手写时通常省略。例如想要创建一张名为 `table` 的表

```
create table table (id int);
```

就存在语法错误，而用

```
create table `table` (id int);
```

就可以完成创建。

WHERE

我们可以使用 `WHERE` 语句进行筛选，并在查询时指定列名。比如查找所有 2016 年及以后入学，且来自于 Austria学生

```
SELECT * FROM students WHERE enroll >= 2020 AND region = 'Austria';
```

IN

在使用 `WHERE` 进行过滤时我们可以指定范围，如我们只想查询 2017, 2019, 2022 这三年入学的学生

```
SELECT * FROM students WHERE enroll IN (2017, 2019, 2022);
```

LIKE

可以使用 `LIKE` 进行模糊匹配。例如寻找名字以 `Jo` 开头的学生

```
SELECT * FROM students WHERE name LIKE 'Jo%';
```

更多例子可以参见 SQL LIKE Operator , SQL Wildcards

NULL

有时我们使用 `NULL` 这个空值来表示记录的某个字段没有数据，可以使用 `ISNULL` 来判断。如找出所有尚未出分的选课记录，可以使用

```
SELECT * FROM association WHERE ISNULL(grade);
```

注意这样的语句是行不通的

```
mysql> SELECT * FROM association WHERE grade = NULL;
Empty set (0.00 sec)
```

判断非空可以使用 `NOT INSULL(<column_name>)` , 如

```
SELECT * FROM association WHERE NOT ISNULL(grade);
```

而不是

```
mysql> SELECT * FROM association WHERE grade <> NULL;
Empty set (0.00 sec)
```

COUNT, MAX, MIN, SUM, AVG

我们可以看看其中有多少条记录

```
SELECT COUNT(*) FROM students;
```

类似地可以查看最大，最小，平均值，比如查看 `enroll` 字段的最大值

```
SELECT MAX(enroll) FROM students;
```

GROUP BY

我们想看看每年有多少学生入学

```
SELECT enroll, COUNT(*) FROM students GROUP BY enroll;
```

在查询过程中，我们可以给字段、表和查询赋予一个临时的，仅在这次查询中使用的名称。

例如对于上面的查询，我们给 `COUNT(*)` 一个临时名称 `student_count`

```
SELECT enroll, COUNT(*) student_count FROM students GROUP BY enroll;
```

注意在 GROUP BY 时，你通常不能选择没有受到 GROUP BY 修饰的字段，例如下面的查询将得到一个错误

```
SELECT name, enroll, COUNT(*) student_count FROM students GROUP BY enroll;
```

```
ERROR 1055 (42000): Expression #1 of SELECT list is not in GROUP BY clause and
contains nonaggregated column 'summer.students.name' which is not functionally
dependent on columns in GROUP BY clause; this is incompatible with
sql_mode=only_full_group_by
```

因为同一年可能 enroll 了多个学生，此时 `name` 字段值的定义可能并不良好。

可以选择其他列的情况在 JOIN 一节说明。

HAVING

可能想要对聚合算子的结果进行过滤，比如试图选出入学人数超过 20 的年份

```
mysql> SELECT enroll, COUNT(*) FROM students WHERE COUNT(*) > 20 GROUP BY enroll
;
ERROR 1111 (HY000): Invalid use of group function
```

遇到了一个错误。

此时可以转而使用 `HAVING` 来过滤聚合算子的结果

```
SELECT enroll, COUNT(*) FROM students GROUP BY enroll HAVING COUNT(*) > 20;
```

DISTINCT

有时我们希望查询的结果中没有重复的值，此时可以使用 DISTINCT 。如查询学生来源于多少个不同的地区。

```
SELECT DISTINCT region FROM students;
```

也可以将 DISTINCT 与 聚合算子联用。如仅查看有多少个不同的地区，而不关心他们是什么

```
SELECT COUNT(DISTINCT region) FROM students;
```

ORDER BY

我们可以对查询结果进行排序，如对学生的入学时间排序

```
SELECT name, enroll FROM students ORDER BY enroll;
```

它默认排了升序，我们也可以规定排降序；

```
SELECT name, enroll FROM students ORDER BY enroll DESC;
```

我们也可以进行对多个字段排序，如优先对地区排升序，地区排序相同的再按入学时间排降序

```
SELECT name, enroll, region FROM students ORDER BY region ASC, enroll DESC;
```

也可以对 `GROUP BY` 的结果排序，比如说列出每个年份入学学生的数量，排升序

```
SELECT enroll, COUNT(*) student_count FROM students GROUP BY enroll ORDER BY
student_count ASC;
```

LIMIT

`LIMIT` 可以对结果指定偏移与范围，如我们查找序号最小的 10 位学生

```
SELECT name, id FROM students ORDER BY id ASC LIMIT 10;
```

假如这样的查询是要分给一个后端，每页10个，我们想要去查第 4 页的结果

```
SELECT name, id FROM students ORDER BY id ASC LIMIT 30, 10;
```

表示从第 `30` 位学生开始，查询 `10` 条记录。

Sub Query

对于每次查询得到的结果集合，我们可以将它视作一个临时的数据表，可以（必须）对他起一个临时名称（别名）后继续进行 `SELECT` 等操作。而这样嵌套在查询中的查询称为 Sub Query，具有很强的表达能力，而且十分符合人类的思维直觉。

我们可以直接从子查询的表中查询相应的列。如希望查询选课数超过了平均值的学生的选课记录，可以将查询分为 3 步。

1. 查询学生选课的数量

   ```
   SELECT COUNT(*) cnt FROM association GROUP BY sid;
   ```

2. 对这些数量求平均

```
SELECT AVG(q1.cnt) FROM (
    SELECT COUNT(*) cnt FROM association GROUP BY sid
) q1;
```

其中 `q1` 是对这子查询的别名。

3. 从 association 中筛选出相应的记录。

```
SELECT sid, COUNT(*) cnt FROM association GROUP BY sid HAVING cnt > (
    SELECT AVG(q1.cnt) FROM (
        SELECT COUNT(*) cnt FROM association GROUP BY sid
    ) q1
);
```

这里我们没有给最外层的子查询别名。

子查询也可以用在 `WHERE` 子句中。如想知查询选课数量排前 10 名的学生的全部信息，可以将查询分为 3 步

1. 从 `association` 表查出选课数量排前 10 名的学生 `id`

```
SELECT sid, COUNT(*) cnt
    FROM association
    GROUP BY sid
    ORDER BY cnt DESC
    LIMIT 10;
```

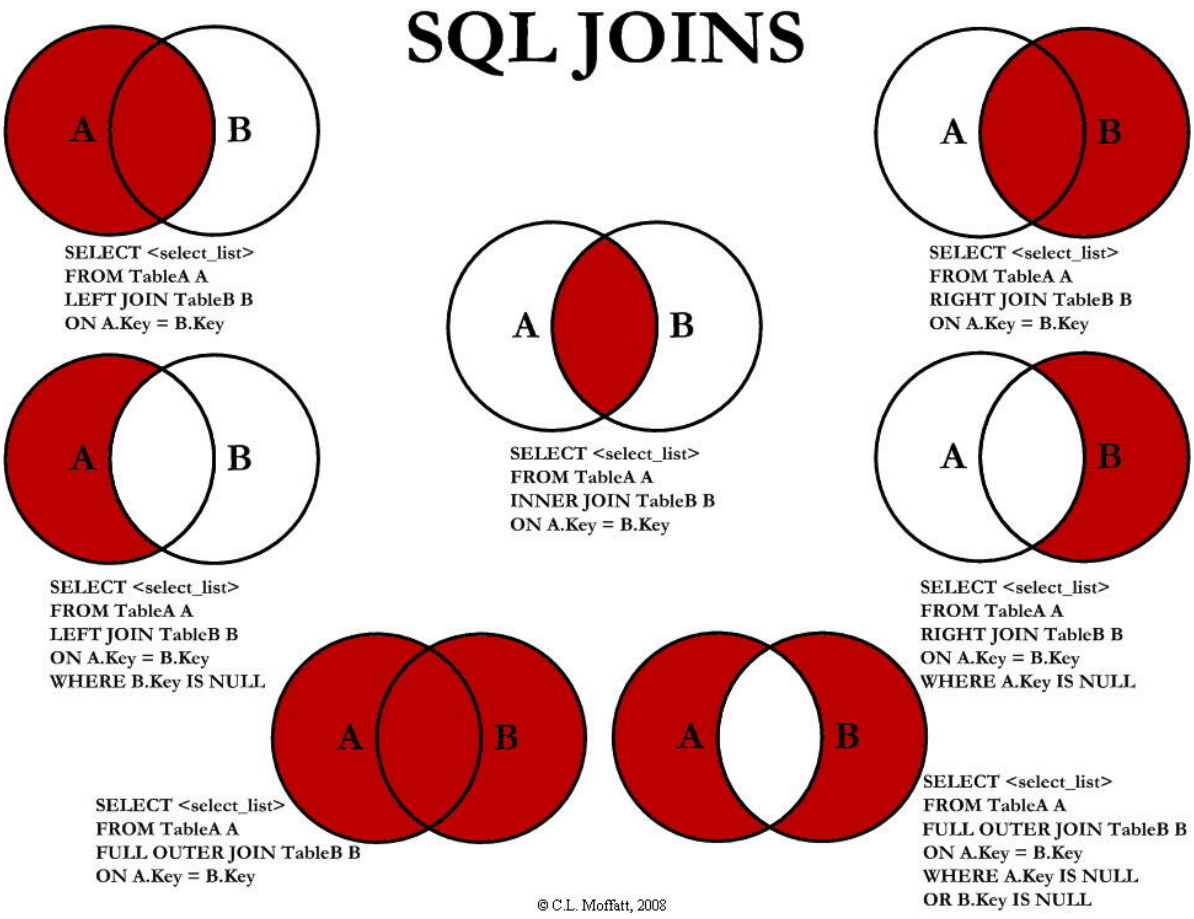
2. 从中选出 `id`

```
SELECT sid FROM (
    SELECT sid, COUNT(*) cnt FROM association GROUP BY sid ORDER BY cnt DESC
LIMIT 10
) top_s;
```

3. 从 `students` 表中查出这些同学的信息。

```
SELECT * FROM students WHERE id in (
    SELECT top_s.sid FROM (
            SELECT sid, COUNT(*) cnt
            FROM association
            GROUP BY sid
            ORDER BY cnt DESC
            LIMIT 10
    ) top_s
);
```

JOIN



一开始我们先来看最基本的 INNER JOIN。

https://dev.mysql.com/doc/refman/8.0/en/join.html

JOIN 使得我们可以将多个表的结果按照某些列的约束关系进行拼接。

例如我们希望查看在选课记录中查看的学号、课程号和成绩的同时，也显示学生的名字。

```
SELECT students.id, students.name, cid , grade
    FROM students            # 第一张表
    INNER JOIN association   # 另一张表
    ON students.id = sid;    # 约束关系
```

INNER JOIN 给出两张表在指定列交集上的笛卡尔积，用某些关键列将表衔接起来。

你可以认为 INNER JOIN 等价于

```
results = []
for record1 in TABLE1:
    for record2 in TABLE2:
        if P(record1, record2):
            results += [record1.concat(record2)]
return result
```

也就选出两张表笛卡尔积中符合谓词 `P` 的记录集合。数据库在执行时未必是这样实现的。

上面的 INNER JOIN 可以隐式表达为

```
SELECT students.id, students.name, cid , grade
    FROM students, association
    WHERE students.id = sid;
```

也可以 JOIN 更多的表，如在显示学生信息与成绩时一并显示课程名。

```
SELECT students.id, students.name, courses.title , association.grade
    FROM students, association, courses
    WHERE students.id = association.sid AND association.cid = courses.id;
```

即使是对于两张表，也可以按需 JOIN 更多的列，此时只不过是 JOIN 的条件 `P` 对多个字段进行了筛选。 例如找出每个年份中，注册最早的学生。与之前 Sub Query 的思路类似，我们首先得到每个年份中注册日期的最小值

```
SELECT enroll, MIN(created_at) ca FROM students GROUP BY enroll;
```

此时的查询的结果有两列 `enroll` 和 `ca` 。将这个查询的结果与 `students` JOIN 起来

```
SELECT students.*
    FROM students, (
        SELECT enroll, MIN(created_at) ca FROM students GROUP BY enroll
    ) early_map
    WHERE students.enroll = early_map.enroll AND students.created_at =
early_map.ca;
```

这时 JOIN 两张表的条件

除了 INNER JOIN 外还有其他 JOIN 方式。例如我们想要看看每位同学都选了几门课，于是写了这样的查询

```
SELECT students.id, COUNT(association.id)
    FROM students, association
    WHERE students.id = association.sid
    GROUP BY students.id;
```

结果选了 0 门课的学生没有返回。这是因为 INNER JOIN 只返回至少存在一个匹配的记录。此时我们可以使用 LEFT JOIN，即无论是否存在匹配，都显示左侧表中的全部记录。对于不存在对应右表记录的左表记录，填充 `NULL`；

```
SELECT students.id, COUNT(association.id)
    FROM students
    LEFT JOIN association
    ON students.id = association.sid
    GROUP BY students.id;
```

这样则会显示全部学生。

有时我们可以用 JOIN 来构造与 SubQuery 类似的查询。

查询选课数量排名前 10 的学生姓名，学号

```
SELECT students.id, students.name, COUNT(*) cnt
    FROM students, association
    WHERE students.id  = sid
    GROUP BY students.id
    ORDER BY cnt DESC
    LIMIT 10;
```

根据我们之前的经验，这个查询会报错。然而我们知道这个查询对 `students.name` 的值是确定的，因为 `students.id` 是唯一的。此时可以给他加上一个 `UNIQUE` 约束

```
ALTER TABLE students
    ADD UNIQUE (id);
```

此时查询就可以执行了。

Functions

作为一门（编程）语言，MySQL 提供了极为丰富的函数库，下面举几个字符串函数的例子。

SUBSTRING

查看姓名从第 1 个字符开始的 3 个字符。

```
SELECT SUBSTRING(name, 1, 3) FROM students;
```

CHAR_LENGTH

看看谁的名字比较长

```
SELECT CHAR_LENGTH(name) name_len, name FROM students ORDER BY name_len DESC
LIMIT 10;
```

更多例子详见文档，不过函数的使用一般不会很多。

REPLACE

如其名称所示，就是对字符串进行替换。比如说我认为 `原理` 是个敏感词，我们在查询时把它换成 `**`

```
SELECT id, REPLACE(title, '原理', '**'), teacher FROM courses;
```

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
```

```
    [LIMIT row_count]

value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

例如我要更新学生 `58` 的入学年份。

```
UPDATE students SET enroll=1999 WHERE id = 58;
```

看一下效果

```
SELECT * FROM students WHERE id = 58;
```

UPDATE 过程中可以直接使用表中的值，如给学生 `58` 的 `enroll` + 10 （虽然我也不知道这有何意义...

```
UPDATE students SET enroll = enroll + 10 WHERE id = 58;
```

DELETE

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]
    [PARTITION (partition_name [, partition_name] ...)]
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

例如将名称中含子串 "数据库" 的课程删除。

```
DELETE FROM courses where title like '%数据库%';
```

相比直接将数据删除，给表记录打上一个标记表示被记录删除更是一种常见的做法。因为这样可以保持数据在逻辑上的完成性，并且让各种记录可以追溯。

之前讨论的 Sub Query, JOIN 等多表操作也可以用在 `UPDATA` 和 `DELETE` 中。

例如如直接清除 `高级摆烂技巧` 的选课记录，但仍保留这门课本身。

```
DELETE association FROM association, courses
    WHERE cid = courses.id AND courses.title = '高级摆烂技巧';
```

在多表删除中，**只有 `DELETE` 和 `FROM` 之间列举的表中的记录会被删除**，`FROM` 后出现 但不在 `DELETE` 和 `FROM` 之间的表仅用做查询的参考。

Space Utilization

值得注意的是删除并不会立即释放磁盘空间。我们不妨进入 [Information Schema] 中看看

```
SELECT TABLE_NAME, DATA_LENGTH
    FROM information_schema.TABLES
    WHERE TABLE_SCHEMA = "s2";
```

其中 DATA_LENGTH 用字节表示。

我们删除一些数据

```
DELETE FROM association;
```

`association` 空了，但表的大小纹丝未动。

为了释放空间，可以使用

```
ANALYZE TABLE record;
```

EXPLAIN

你可能关心查询是如何被数据库执行的，这时可以使用 `EXPLAIN ANALYZE`，让 MySQL 打印执行计划。更多示例详见 文档

```
EXPLAIN ANALYZE
SELECT students.id, students.name, courses.title , association.grade
    FROM students, association, courses
    WHERE students.id = association.sid AND association.cid = courses.id;
```

```
 -> Inner hash join (students.id = association.sid)  (cost=84979.22 rows=80962) (actual time=1.439..1.720 rows=1336 loops=1)
    -> Table scan on students  (cost=0.00 rows=202) (actual time=0.008..0.120 rows=202 loops=1)
    -> Hash
        -> Inner hash join (cast(association.cid as double) = cast(courses.id as double))  (cost=4013.21 rows=4008) (actual time=0.093..1.158 rows=1336 loops=1)
            -> Table scan on association  (cost=0.51 rows=1336) (actual time=0.008..0.806 rows=1336 loops=1)
            -> Hash
                -> Table scan on courses  (cost=3.25 rows=30) (actual time=0.026..0.056 rows=30 loops=1)
```

SQL Contraints

MySQL 中可以对字段指定数据规则，称为约束。

NOT NULL

UNIQUE

`UNIQUE` 约束要求收到约束的列不能存在相同的数据。可以仅约束一列或者约束多列（联合唯一）。

PRIMARY KEY

PRIMARY KEY 约束要求被约束的列唯一且非空。每个表只能有一个主键约束但主键约束可以施加在多个列上（联合主键）

主键

一种常见的操作是给主键指定 `AUTO INCREMENT`

这样在插入记录时，将会自动产生一个自增的新值。

DEFAULT

使用 `DEFAULT` 约束时，如果插入时没有指定该字段的值，将会分配指定的默认值。

我们指定个默认约束。

FOREIGN KEY

`FOREIGN KEY` 约束用于保证多表之间的关系完整。其通常指向另一张表中的主键（或唯一约束的字段）。`FOREIGN KEY` 约束用于预防破坏表之间连接的行为。

在 Python 中使用 MySQL

通常如果使用 `Django` 等框架，你无需手写 SQL 语句，而可以直接使用它的 ORM 框架。

由于今天主要讲解 SQL 语法，下面简单介绍一下如何在 Python 中 `PyMySQL` 的使用。

可以使用 `PyMySQL` 与 MySQL 数据库进行连接，下面的例子摘自 PyMySQL 文档

```
import pymysql.cursors

# 连接数据库
connection = pymysql.connect(host='localhost',
                             user='user',
                             password='passwd',
                             database='db',
                             charset='utf8mb4',
                             cursorclass=pymysql.cursors.DictCursor)

with connection:
    with connection.cursor() as cursor:
        sql = "INSERT INTO `users` (`email`, `password`) VALUES (%s, %s)"
        cursor.execute(sql, ('webmaster@python.org', 'very-secret'))
    connection.commit()  # 提交记录更新

    with connection.cursor() as cursor:
        sql = "SELECT `id`, `password` FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('webmaster@python.org',))
        result = cursor.fetchone()
```

```
        print(result)
```

这样的查询会返回如下格式

```
{'id': 1, 'password': 'very-secret'}
```

Previous Query

有时你可能想要了解 `cursor.execute` 执行了怎样的语句

```
cursor._last_executed
```

将返回上一条执行的 SQL 语句。但在某些执行失败的情况下，该值也可能不存在。

Different Cursor

如果想要使用一个默认 `Curosr` 之外的 `Cursor`，也可以在执行时指定

```
with conn.cursor(sql.cursors.DictCursor) as cur:
    cur.execute(f"SELECT * FROM `video_info` WHERE author=%s;", ("alice",))
    res = cur.fetchall()
```

`DictCursor` 会将查询结果以 `dict` 形式返回，使用起来较为方便。

Get the value of `AUTO INCREMENT`

```
cursor.lastrowid
```

Cursor Operation

对于记录取用，我们通常使用带缓存的 `Cursor` （默认）。 `Cursor` 可以看做是记录中的一个指针。这里不妨直接看 `PyMySQL` 的代码实现

```
def fetchone(self):
    """Fetch the next row"""
    self._check_executed()
    if self._rows is None or self.rownumber >= len(self._rows):
        return None
    result = self._rows[self.rownumber]
    self.rownumber += 1
    return result

def fetchmany(self, size=None):
    """Fetch several rows"""
    self._check_executed()
    if self._rows is None:
        return ()
    end = self.rownumber + (size or self.arraysize)
    result = self._rows[self.rownumber : end]
    self.rownumber = min(end, len(self._rows))
    return result
```

```
def fetchall(self):
    """Fetch all the rows"""
    self._check_executed()
    if self._rows is None:
        return ()
    if self.rownumber:
        result = self._rows[self.rownumber :]
    else:
        result = self._rows
        self.rownumber = len(self._rows)
        return result

def scroll(self, value, mode="relative"):
    self._check_executed()
    if mode == "relative":
        r = self.rownumber + value
    elif mode == "absolute":
        r = value
    else:
        raise err.ProgrammingError("unknown scroll mode %s" % mode)

        if not (0 <= r < len(self._rows)):
            raise IndexError("out of range")
            self.rownumber = r
```

SQL Injection

假设我有这样一张表

```
CREATE TABLE `rdts_repository`  (
  `id` char(10),
  `title` varchar(255),
  `description` longtext,
  `createdAt` double ,
  `disabled` tinyint,
  `createdBy_id` bigint,
  `project_id` bigint,
  `url` varchar(255),
)
```

有这样一些数据

id	title	description	createdAt	disabled	createdBy_id	project_id	url
1	My Repo 1	A repo	1649836834.109174	1	1	2	
2	FrontEnd	New Repo!	1649848505.240757	1	1	2	
3	BackEnd	New Repo!	1649848595.963635	1	1	2	
4	Doc	New Repo!	1649848639.957997	1	1	2	
5	F	New Repo!	1650029036.351989	0	1	2	gitlab.secoder.net
6	B	New Repo!	1650029061.328037	0	1	2	gitlab.secoder.net
7	D	New Repo!	1650029077.123462	0	1	2	gitlab.secoder.net
8	gll	New Repo!	1650038980.48267	0	28	27	gitlab.secoder.net
9	lalalalaa	New Repo!	1650116931.127209	1	1	2	gitlab.secoder.net
10	shit!	New Repo!	1650370905.42494	1	25	17	gitlab.secoder.net
11	test-repo	New Repo!	1650375829.77901	1	1	1	gitlab.secoder.net
12	SampleRepo1	New Repo!	1650378232.022374	1	1	28	gitlab.secoder.net
13	REmote	New Repo!	1650379316.104355	1	1	29	gitlab.secoder.net
14	S	New Repo!	1650379574.254712	1	1	29	gitlab.secoder.net
15	name	New Repo!	1650938559.602208	1	25	26	gitlab.secoder.net
16	holy shit!	New Repo!	1650938882.376347	1	25	26	gitlab.secoder.net
17	torch-secoder	New Repo!	1651028161.293827	1	25	34	gitlab.secoder.net
18	torch-clone	New Repo!	1651052124.361267	1	25	34	gitlab.secoder.net
19	new	-	0	0	2	34	gitlab.secoder.net

试图从中获取指定 `id` 的数据，假设使用了这样一段代码

```
with conn.cursor(sql.cursors.DictCursor) as cur:
    cur.execute(f"SELECT * FROM `rdts_repository` WHERE `id`='%s';" %
user_input)
    res = cur.fetchall()
```

其中 `user_input` 是用户提供的字符串。然而用户输入了

```
user_input = "1' OR 1=1 OR '"
```

拼接起来就是

```
SELECT * FROM `rdts_repository` WHERE '1' OR 1=1 OR '';
```

此时用户就会得整张表的全部数据。

再如用户输入

```
user_input = "1'; DROP TABLE `rdts_repository`; SELECT '1"
```

拼接起来就是

```
SELECT * FROM `rdts_repository` WHERE `id`='1'; DROP TABLE `rdts_repository`;
SELECT '1';
```

从而库没了耶ヾ(✿ﾟ▽ﾟ)ノ

因此，正确地使用 `PyMySQL` 会为你的输入进行自动转义，上述查询应该写成这样

```
with conn.cursor(sql.cursors.DictCursor) as cur:
    cur.execute(
        f"SELECT * FROM `rdts_repository` WHERE `id`=%s;",
        (user_input,)
    )
    res = cur.fetchall()
```

此时 `execute` 函数会负责对你的输入进行转义，上面两条危险输入就会对应到

```
SELECT * FROM `rdts_repository` WHERE `id`='1\' OR 1=1 OR \'';
SELECT * FROM `rdts_repository` WHERE `id`='1\'; DROP TABLE *; SELECT \'1';
```

从而变得人畜无害。

Reference

1. rls 2021 暑培 SQL 讲义
2. MySQL Documentation
3. w3school SQL Tutorial