Day05-MySQL基础
今日课程学习目标
今日课程内容大纲
知识点1：MySQL常用内置函数-时间日期函数【熟悉】
知识点2：MySQL常用内置函数-字符串函数【熟悉】
知识点3：MySQL常用内置函数-数学函数【熟悉】
知识点4：事务简介和ACID特性【熟悉】
知识点5：数据库表的存储引擎简介【了解】
知识点6：事务的使用步骤说明【熟悉】
知识点7：事务的使用演示示例【熟悉】
知识点8：索引的作用和相关使用命令【熟悉】
知识点9：索引性能验证-添加测试数据【了解】
知识点10：索引性能验证-添加索前后对比【了解】
知识点11：联合索引和使用的最左原则【熟悉】
知识点12：索引的优缺点及使用原则【熟悉】
知识点13：PyMySQL-模块简介【掌握】
知识点14：PyMySQL-查询操作【掌握】
知识点15：PyMySQL-增删改操作【掌握】

Day05-MySQL基础

今日课程学习目标

```
熟悉 MySQL常用内置函数：时间日期函数、字符串函数、数学函数
熟悉 事务的概念及其应用场景
熟悉 索引的概念及其作用
常握 PyMySQL对数据库的增、删、改、查操作
```

今日课程内容大纲

```
# 1. MySQL常用内置函数
    时间日期函数
    字符串函数
    数学函数
# 2. 事务vs索引
    事务
    索引
# 3. PyMySQL模块
    PyMySQL-模块简介
    PyMySQL-查询操作
    PyMySQL-增删改操作
```

知识点1：MySQL常用内置函数-时间日期函数【熟悉】

我们在做数据处理和开发的过程中，经常需要利用SQL在MySQL或HIVE、SPARK中使用一些SQL函数，对时间日期类型的数据进行操作。

时间日期函数分类：

1）获取当前时间的函数，比如当前的年月日或时间戳

2）计算时间差的相关函数，比如两个日期之间相差多少天，一个日期90天后是几月几号

3）获取年月日的函数，从一个时间中提取具体的年份、月份

4）时间转换函数，比如将2021-10-05转换为时间戳

重点函数举例：

```
-- 获取当前datetime类型的时间
SELECT NOW();

-- 计算指定间隔的时间日期
SELECT DATE_ADD('2007-9-27', INTERVAL 90 DAY );

-- 示例：获取当前时间3个小时以前的时间
SELECT DATE_ADD(NOW(), INTERVAL -3 HOUR);

-- 计算两个时间日期之间的天数差
SELECT DATEDIFF('2021-03-22 09:00:00', '2018-03-20 07:00:00');

-- 提取时间日期中的年月日
-- 获取当前日期中的年份
SELECT YEAR(NOW());

-- 获取2021-10-02 09:00中月份
SELECT MONTH('2021-10-02 09:00');

-- 获取时间日期中的日
SELECT DAY('2021-10-02 09:00');
```

知识点2：MySQL常用内置函数-字符串函数【熟悉】

我们在做数据处理和开发的过程中，经常需要利用SQL在MySQL或HIVE、SPARK中使用一些SQL函数；和时间函数一样，也经常需要对字符串类型的数据进行处理操作。

字符串函数分类：

1）大小写转换、反转

2）对字符串进行拼接、删除前后缀，或做局部替换

3）获取局部的子串，字符串的字符个数以及存储长度

重点函数举例：

```
-- 字符串拼接
SELECT CONCAT('马走日', '象飞田');
SELECT CONCAT(10.15, '%');

-- 指定分隔符拼接字符串
SELECT CONCAT_WS('^_^', '马走日', '象飞田');
```

知识点3：MySQL常用内置函数-数学函数【熟悉】

我们在做数据处理和开发的过程中，经常需要利用SQL在MySQL或HIVE、SPARK中使用一些SQL函数；和时间函数一样，也经常需要对数字类型的数据进行处理操作。

重点函数举例：

```
-- ROUND(X, n)：对 X 进行四舍五入，保留 n 位小数，默认n为0
SELECT ROUND(1.6);
```

```sql
SELECT ROUND(1.333, 2);
SELECT ROUND(2.689, 2);

-- FORMAT(X, n)：对 X 进行四舍五入，保留 n 位小数，以##,###,###.###格式显示
SELECT FORMAT(1001.6, 2);
SELECT FORMAT(123456.333, 2);
SELECT FORMAT(234567.689, 2);

-- FLOOR(x)：向下取整
SELECT FLOOR(-1.5);

-- CEIL(X)：向上取整
SELECT CEIL(2.1);

-- GREATEST(expr1, expr2, expr3, ...)：返回列表中的最大值
SELECT GREATEST(3, 12, 34, 8, 25);

-- LEAST(expr1, expr2, expr3, ...)：返回列表中的最小值
SELECT LEAST(3, 12, 34, 8, 25);
```

知识点4：事务简介和ACID特性【熟悉】

事务应用场景：一组SQL操作，同时成功或同时失败，比如：转账

转账的基本流程：

- 第一步：开启一个事务
- 第二步：减少转出账户的余额：转出 200 元
- 第三步：增加转入账户的余额：增加 200 元
- 第四步：提交事务

事务的四大特性：

- 原子性(Atomicity)：事务是一个整体，不会部分成功，部分失败
- 一致性(Consistency)：事务保存数据库从一个一致性状态转移到另一个一致性状态
- 隔离性(Isolation)：一个事务的操作提交之前，对另一个事务是不可见的
- 持久性(Durability)：事务一旦提交，结果便会被永久保存

知识点5：数据库表的存储引擎简介【了解】

常见的数据库引擎：

- InnoDB：默认的数据库表存储引擎，支持事务操作
- MyISAM：不支持事务，优势是访问速度快，对事务没有要求或者以select、insert为主的都可以使用该存储引擎来创建表

相关命令：

```sql
SHOW ENGINES; -- 查询数据库支持的存储引擎
SHOW CREATE TABLE 表名; -- 查询表的创建SQL
ALTER TABLE 表名 ENGINE = 引擎类型; -- 修改指定表的存储引擎
```

知识点6：事务的使用步骤说明【熟悉】

事务使用步骤：

1）开启事务：`begin` 或 `start transaction`

- **开启事务后执行修改命令，变更数据会保存到 MySQL 服务端的缓存文件中，而不维护到物理表中**

- **MySQL数据库默认采用自动提交(autocommit)模式，如果没有显示的开启一个事务，那么每条sql语句都会被当作一个事务执行提交的操作**

2）事务中的 SQL 操作

3）结束事务：提交事务(`commit`)或回滚事务(`rollback`)

- 提交事务：将本地缓存文件中的数据提交到物理表中，完成数据的更新
- 回滚事务：放弃本地缓存文件中的缓存数据，表示回到开始事务前的状态

知识点7：事务的使用演示示例【熟悉】

> 参考讲义的示例

知识点8：索引的作用和相关使用命令【熟悉】

索引在MySQL中也叫做“键”，它是一个特殊的文件，它保存着数据表里所有记录的位置信息，更通俗的来说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度。

索引应用场景：数据量较大时，提高根据某些字段查询数据的效率

索引命令：

```
SHOW INDEX FROM 表名；-- 查看指定表中有哪些索引
ALTER TABLE 表名 ADD INDEX 索引名 (列名, ...)；-- 在表的指定列上添加索引
ALTER TABLE 表名 DROP INDEX 索引名; -- 删除索引
```

知识点9：索引性能验证-添加测试数据【了解】

创建测试表：

```
# 创建 python 数据库
CREATE DATABASE python CHARSET=utf8;
# 创建 test_index 数据表
USE python;
CREATE TABLE test_index(title VARCHAR(10));
```

添加测试数据：

> 暂时无需理解代码，执行代码添加测试数据即可

```
from pymysql import connect


def main():
    # 创建数据库连接对象
    conn = connect(host='localhost', port=3306,
                   database='123456', user='root', password='mysql')
    # 创建游标对象
    cursor = conn.cursor()
    # 循环向 test_index 表中添加 10 万条测试数据
    for i in range(100000):
        cursor.execute("insert into test_index values('py-%d')" % i)
    # 提交数据
    conn.commit()


if __name__ == "__main__":
```

```
18        main()
```

知识点10：索引性能验证-添加索前后对比【了解】

```
1   # 开启 sql 执行时间监测
2   set profiling=1;
3   # 查找第1万条数据'py-99999'
4   select * from test_index where title='py-99999';
5   # 查看 sql 执行的时间
6   show profiles;
7
8   # 给 title 字段创建索引
9   alter table test_index add index (title);
10  # 再次执行 sql 查询语句
11  select * from test_index where title='py-99999';
12  # 再次查看 sql 执行的时间
13  show profiles;
```

```
mysql> show profiles;
+----------+------------+-------------------------------------------------+
| Query_ID | Duration   | Query                                           |
+----------+------------+-------------------------------------------------+
|        1 | 0.04211575 | select * from test_index where title='py-99999' |
|        2 | 0.33962775 | alter table test_index add index (title)        |
|        3 | 0.00034750 | select * from test_index where title='py-99999' |
+----------+------------+-------------------------------------------------+
```

添加索引之前

添加索引之后

> 结果：给 title 字段添加索引之后，通过 title 字段筛选数据时，查询效率明显提升

知识点11：联合索引和使用的最左原则【熟悉】

- 联合索引又叫复合索引，即一个索引覆盖表中两个或者多个字段，一般用在多个字段一起查询的时候。
- 联合索引能够减少磁盘空间开销，因为每创建一个索引，其实就是创建了一个索引文件，那么会增加磁盘空间的开销。

```
1   # 创建teacher表
2   create table teacher
3   (
4       id int not null primary key auto_increment,
5       name varchar(10),
6       age int
7   );
8
9   # 创建联合索引
10  alter table teacher add index (name, age);
```

> - 联合索引使用要遵循**最左原则**：在使用联合索引的查询数据时候一定要保证联合索引的最左侧字段出现在查询条件里面，否则联合索引失效

```sql
# 下面的查询使用到了联合索引
# 示例1：这里使用了联合索引的name部分
select * from stu where name='张三'
# 示例2：这里完整的使用联合索引，包括 name 和 age 部分
select * from stu where name='李四' and age=10

# 下面的查询没有使用到联合索引
# 示例3： 因为联合索引里面没有这个组合，只有【name】和【name age】这两种组合
select * from stu where age=10
```

知识点12：索引的优缺点及使用原则【熟悉】

优点：

1）加快数据的查询速度

缺点：

2）创建索引会耗费时间和占用磁盘空间，并且随着数据量的增加所耗费的时间也会增加

使用原则：

1）不要滥用索引，只针对经常查询的字段建立索引

2）数据量小的表最好不要使用索引

3）在一个字段上相同值比较多不要建立索引，比如性别

知识点13：PyMySQL-模块简介【掌握】

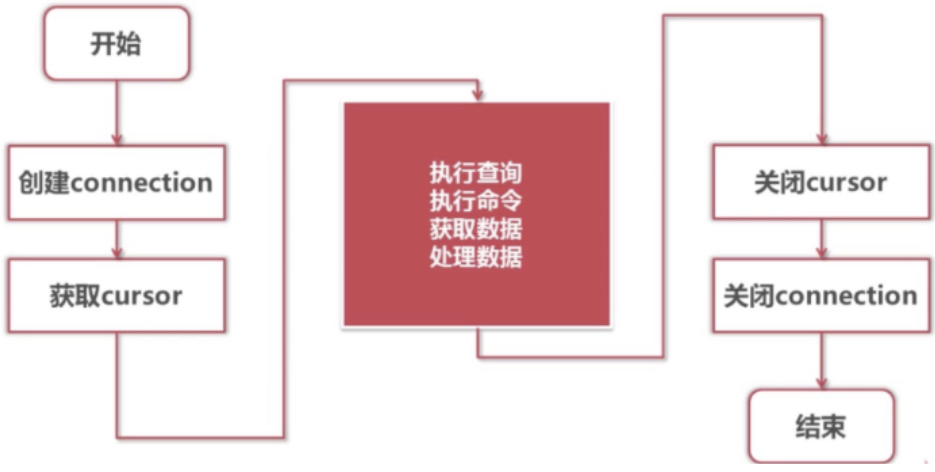
PyMSQL模块是Python中用来操作MySQL数据库的模块，作用就是利用程序操作 MySQL 数据库，进行数据库的增、删、改、查操作。

安装：

```sql
# 安装 pymysql 扩展包
pip install pymysql==1.0.2
```

操作步骤：

> 导入模块 → 创建连接 → 创建游标 → 执行SQL → 关闭游标 → 关闭连接



1）引入模块

```python
# 导入 pymysql 扩展包
import pymysql
```

2）创建连接(创建connection连接对象)

> **作用**：用于和数据库建立连接，调用pymysql模块中的connect()方法

```
# 创建数据库连接对象
conn = pymysql.connect(参数列表)

参数说明：
* 参数host：连接的mysql主机，如果本机是'localhost'
* 参数port：连接的mysql主机的端口，默认是3306
* 参数database：数据库的名称
* 参数user：连接的用户名
* 参数password：连接的密码
* 参数charset：通信采用的编码方式，推荐使用utf8
```

3）创建游标(通过连接对象获取游标)

> 游标的作用：**执行sql语句**（使用频度最高的语句为select、insert、update、delete)

```python
# 通过conn对象的cursor方法，获取游标对象
cursor = conn.cursor()
```

4）执行SQL（这里需要使用游标执行SQL语句）

```python
# 通过游标对象的execute方法，执行SQL语句
# 注意：返回值是受影响的行数
num = cursor.execute(sql)
```

5）关闭游标

```python
# 注意：游标的关闭应该在连接关闭前
cursor.close()
```

6）关闭连接

```python
# 注意：先关闭游标后关闭连接
conn.close()
```

知识点14：PyMySQL-查询操作【掌握】

```python
# 导入 pymysql 模块
import pymysql

# 创建数据库连接
conn = pymysql.connect(host='localhost', port=3306,
                       user='root', password='123456',
                       database='winfunc', charset='utf8')

# 获取游标
cursor = conn.cursor()

# 准备执行的 SQL 语句
sql = 'SELECT * FROM students'

```

```python
# 使用游标执行 SQL 语句
row_count = cursor.execute(sql)
print("SQL语句执行影响的行数：%d" % row_count)

# 关闭游标
cursor.close()

# 关闭连接
conn.close()
```

cursor游标对象中有三个方法，用于获取 SQL 查询的结果：

- fetchall()：获取查询的所有结果，返回值为tuple元祖类型【嵌套元祖】
- fetchone()：每次获取查询结果中的一条记录，返回值为tuple元祖类型【简单元祖】
- fetchmany(num)：每次获取查询结果中的num条记录，返回值为tuple元祖类型【嵌套元祖】

> 注意：调用获取数据的方法时，每次是基于上次的位置继续向后获取，获取不到数据时，返回空元祖

知识点15：PyMySQL-增删改操作【掌握】

PyMySQL 在对 MySQL 数据库进行操作时，**当遇到非查询(即：增、删、改)操作时，自动开启一个默认事务**，后续的 SQL 操作都在这个事务中，在**操作完成之后，如果要结果生效，需要手动进行事务的 commit 提交，否则结果被撤销，即自动进行了事务的 rollback 操作。**

问题：

使用 PyMySQL 操作数据库，如何进行事务的开启、提交、回滚操作?

```
# 通过通过创建的数据库连接对象，进行事务相关操作，方法如下：
conn.begin()：开启一个事务
conn.commit()：进行事务的提交操作，事务操作结果永久生效
conn.rollback()：进行事务的回滚操作，事务操作结果被撤销
```