第二章 金融数据库搭建及使用

方杰、李烜

福建江夏学院金融学院

本章内容

1. 数据库基础知识
   - 数据库的概念及分类
   - 关系型数据库
   - 非关系型数据库
2. 数据存储
   - 创建表格
   - 删除数据
3. 数据查询
   - 单表查询
   - 联表查询
4. 补充内容：使用 Python 进行联表查询

数据库的概念

- 数据库是“按照数据结构来组织、存储和管理数据的仓库”，是一个长期存储在计算机内的、有组织的、可共享的、统一管理的大量数据的集合。
- 从狭义的方面来讲，数据库是用于存储数据的地方，相当于一个电子化的文件柜。一个数据库可能包含多个表或者文件，一个数据库系统中通常包含多个数据库。
- 从广义的方面理解，数据库不仅是一个存储数据的容器，还是一个按数据结构来存储和管理数据的计算机软件系统。

数据库的分类

- 层次式数据库
- 网络式数据库
- 关系型数据库
- 非关系型数据库

关系型数据库

关系型数据库存指采用了关系模型来组织数据，由二维表及其之间的联系组成的一个数据组织。一个关系型数据库中可以包含一个或多个表格，表格间如存在一定的关系，可通过联结 (join) 语句对表建立关系

- 优点
  - 易理解
  - 使用方便
  - 易维护
- 缺点
  - 硬盘 I/O 压力大
  - 不适用于非结构化数据
  - 横向扩展性差
  - 性能欠佳

主流的关系型数据库

- Oracle：甲骨文公司出品的一款以分布式数据库为核心的关系数据库管理系统。
- MySQL：由瑞典 MySQL AB 公司开发，属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一。
- SQL Server：微软公司推出的关系型数据库管理系统，适合大容量的数据存储，可扩展性强、相关软件集成度高，并且操作简单。

SQL 是什么?

SQL 是结构化查询语言（Structured Query Language）的简称，SQL 语言是目前广泛使用的关系型数据库标准语言，是各种数据库交互方式的基础，用于存取数据以及查询、更新和管理关系型数据库系统

非关系型数据库的含义与特点

非关系型数据库 (NoSQL) 指除关系型数据库之外的其他数据存储形式，是非关系型、分布式，且一般不保证遵循 ACID 原则的数据存储系统的统称。

- 优点
  - 格式灵活
  - 速度快
  - 高扩展性
  - 成本低
- 缺点
  - 不支持 SQL 语句
  - 数据结构相对复杂
  - 复杂查询方面比较欠缺
  - 附加功能商业智能 (BI) 和报表等支持也较差

非关系型数据库的分类

- 键值存储数据库：主要采用哈希表技术，存储特定的键和指向特定的数据指针。
- 文档型数据库：以嵌入式版本化文档为数据模型，支持全文检索、关键字查询等功能。
- 列存储数据库：数据访问速度更快，压缩率更高，支持大规模横向扩展。
- 图 (Graph) 数据库：以图论为理论根基，用节点和关系所组成的图为模型。

数据库设计流程

- 创建表格：通过数据库定义语言将事先设计好的数据表物理结构在数据库中实现
- 写入数据：在创建好的表中写入数据，写入的数据需符合数据库中字段的定义要求
- 质检和更新：发现录入数据错误要及时修改或删除，并定期对数据进行更新，以保障数据的质量

建表

SQL 语言中用 CREATE TABLE 语句创建数据库中的表，具体语法:

```
CREATE TABLE 数据表名
(
    字段名称1 数据类型,
    字段名称2 数据类型,
    ......
    字段名称n 数据类型
)
```

数据类型

见 P125—126

SQL 语言的数据类型主要有：字符串型、数值型、日期型

字符串型

类型	用途
CHAR	定长字符串
VARCHAR	可变长字符串
TEXT	长文本数据

数值型

类型	用途
TINYINT	小整数值
INTEGER	大整数值
BIGINT	极大整数值
FLOAT	单精度浮点数值
DOUBLE	双精度浮点数值
DECIMAL(M,D)	小数值

日期型

类型	用途
DATE	日期值
TIME	时间值或持续时间
YEAR	年份值
DATETIME	混合日期和时间值
TIMESTAMP	混合日期和时间值，时间戳

建表举例

```
CREATE TABLE TABLE1
    (   Stock varchar(20),
        Symbol char(6),
        Market varchar(20),
        ShareType char(1),
        CloseTime date,
        ClosePrice decimal(10,2)
    );
```

创建的空表“TABLE1”如下:

Stock	Symbol	Market	ShareType	CloseTime	ClosePrice

插入数据：在一行中插入数据

```
INSERT INTO 表名称（字段1，字段2，...）VALUES（值1，值2，
    ...）
```

注意：

这里的字段 n 要与值 n 之间一一对应。

插入数据：插入一列

```
ALTER TABLE 表名称 ADD (列名称 数据类型)
```

举例

```
ALTER TABLE TABLE1 ADD (Yield decimal(10,5));
```

在表 TABLE1 中插入一列“Yield”，该字段为小数值格式

数据修改

数据库中由于数据源数据缺失或更新、录入操作失误等问题，难免会出现数据空值、数据错误等问题，这时就需要数据进行修正和更新。有的错误数据可以通过查询数据源获取正确数据，有的数据则需要重新计算，然后再将正确数据更新到数据表中。

数据修改：数据更新

对指定列中的所有数据均赋同一个数值

```
UPDATE 表名称 SET 列名 = 数值
```

对满足条件的指定列中的数据赋同一个数值

```
UPDATE 表名称 SET 列1 = 数值1, 列2 = 数值2
              WHERE 列3 = 数值3
```

当列 3 取值为数值 3 时，对该行的列 1 和列 2 分别赋值。

运算符

算数运算符	说明	比较运算符	说明
+	加	= / !=	等于/不等于
-	减	> / >=	大于/大于等于
*	乘	< / <=	小于/小于等于
/	除	BETWEEN/NOT BETWEEN	在两个值之间/不在两个值之间

注意：

多个运算符连用时，可通过圆括号来区分优先顺序

数据修改：数据计算

计算 TABLE1 中平安银行 2022 年 2 月 18 日的股票收益率。用收盘价计算股票当日收益率的计算公式为：

$$\text{当日收益率} = \frac{\text{当日收盘价}}{\text{前一日收盘价}} - 1$$

```
SELECT (SELECT ClosePrice FROM TABLE1 WHERE Symbol='000001'
       AND CloseTime='2022/2/18' )/(SELECT ClosePrice
       FROM TABLE1 WHERE Symbol='000001' AND
       CloseTime='2022/2/17')-1 AS Yield
```

说明：

SELECT语句是 SQL 中的查询语句

数据修改：字段拼接与别名

字段拼接 (concatenate) 是指将数据库表中的多个字段拼接变为一个字段，增加数据可读性、减少歧义

```
SELECT CONCAT(字段1, 字段2, ...) FROM 表名
```

字段拼接时可在字段中间增加分隔符，常用的分割符有“.”、“_”等。

```
SELECT CONCAT(Symbol, '_', Market) FROM TABLE1
```

数据修改：字段拼接与别名 (cont.)

原名 **AS** 别名

举例：

```
SELECT CONCAT(Symbol, '_', Market) AS '股票代码' FROM TABLE1
```

删除行数据

删除某一行

```
DELETE FROM 表名称 WHERE 列名称 = 值
```

删除选定列名称取值的行。

删除表中的所有行

```
DELETE FROM 表名称
DELETE * FROM 表名称
```

注意：此操作只是删除了表中所有的行，但未改变原表的结构、属性和索引。

单表查询

查询所有数据

```
SELECT * FROM 表名称
```

注意：*相当于通配符，表示表中所有的字段。

按字段查询

```
SELECT 字段1，字段2，... FROM 表名称
```

单表查询：条件查询

```
SELECT 字段1，字段2，... FROM 表名称
        WHERE 字段名称 运算符 值
```

说明：

- “字段1，字段2，...”表示被检索列，输出结果只显示这几列；也可以使用“*”表示查询全表，输出结果为全表。
- 运行语句时，先进行WHERE子句的条件判断，再进行数据抽取。

SQL 语言运算符

运算符	描述
=	等于
<> 或 !=	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于
LIKE	搜索某种模式

多条件查询

```
SELECT 字段1，字段2，... FROM 表名称
    WHERE 条件1 AND/OR 条件2
```

- AND表示“并且，和”，A AND B 表示 A 条件和 B 条件都成立
- OR表示“或者”，A OR B 表示 A 条件或 B 条件中只需一个成立

说明:

AND和OR可以联合使用，构成更为复杂的判定语句

多条件查询 (cont.)

```
SELECT 字段1，字段2，... FROM 表名称
    WHERE 字段 BETWEEN A AND B
SELECT 字段1，字段2，... FROM 表名称
    WHERE 字段 NOT BETWEEN A AND B
```

- 使用运算符BETWEEN A AND B，可以选取介于 $[A, B]$ 之间的数据，包含 A 和 B
- 使用NOT BETWEEN语句筛选两个值之外的数据，其中 A、B 均不包含

注意：

不同的数据库对 BETWEEN A AND B 操作符的处理方式是有差异的。某些数据库不包括 A 和 B；某些数据库包括 A 和 B 的数据；而另一些数据库包括 A，但不包括 B 的数据

联表查询

在现实生活中，经常需要通过一次性查询多张表格，获取信息。结合查询的数据结果进行后续处理工作。一次性查询多张表的数据，称为联表查询。

后面将通过t1和t2两张表来演示联表查询。

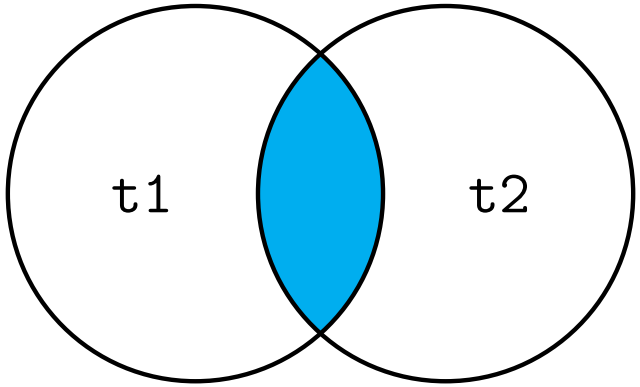
t1 和 t2 两张表的内容

StockId	Stock	Symbol	ExchangeId
2021020100001	上证指数	000001	SHSE
2021020100002	平安银行	000001	SZSE
2021020100003	万科 A	000002	SZSE
202000000004	嘉实增强信用定期债券	000005	NULL

Id	Exchange	ExchangeEn
SZSE	深圳证券交易所	Shenzhen Stock Exchange
SHSE	上海证券交易所	Shanghai Stock Exchange
HKEX	香港联合交易所	Stock Exchange of Hong Kong

内联结

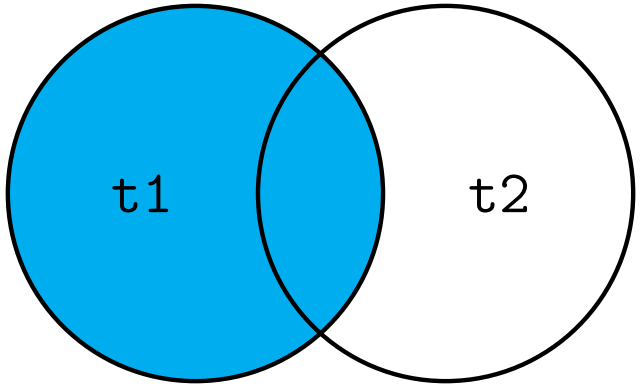
内联结（INNER JOIN）是最常见的连接类型，返回两个联接表中都匹配的行




```
SELECT t1.StockId, t1.Stock, t1.Symbol, t2.Exchange
    FROM t1 INNER JOIN t2 ON t1.ExchangeId = t2.Id
```

外联结：左联结

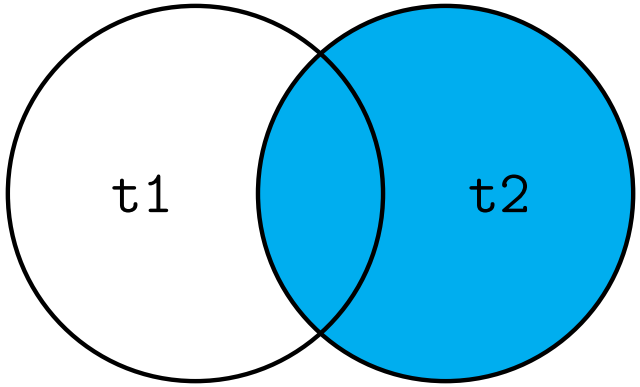
左联结（LEFT JOIN）返回左表中的所有行以及右表中满足连接条件的行。




```
SELECT t1.StockId, t1.Stock, t1.Symbol, t2.Exchange
    FROM t1 LEFT JOIN t2 ON t1.ExchangeId = t2.Id
```

外联结：右联结

右联结（RIGHT JOIN）返回右表中的所有行以及左表中满足连接条件的行。

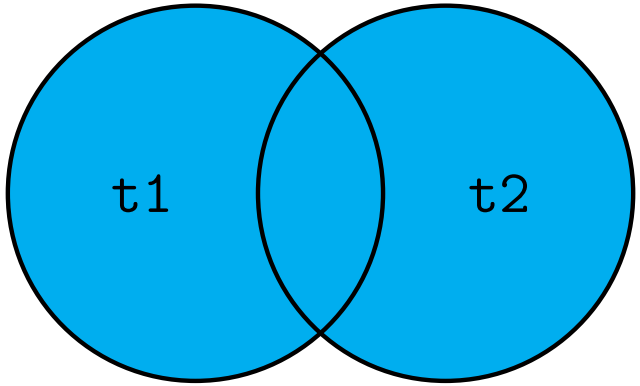



```
SELECT t1.StockId, t1.Stock, t1.Symbol, t2.Exchange
    FROM t1 RIGHT JOIN t2 ON t1.ExchangeId = t2.Id
```

。

外联结：完全联结

完全联结（FULL JOIN）返回两表中的所有行，无论它们是否匹配



在 MySQL 中，不支持 FULL JOIN，所以使用UNION ALL运算符来组合LEFT JOIN和RIGHT JOIN

完全联结 (cont.)

```
SELECT t1.StockId, t1.Stock, t1.Symbol, t2.Exchange
    FROM t1 LEFT JOIN t2 ON t1.ExchangeId = t2.Id
  UNION ALL
SELECT t1.StockId, t1.Stock, t1.Symbol, t2.Exchange
    FROM t1 RIGHT JOIN t2 ON t1.ExchangeId = t2.Id
```

注意:

使用UNION ALL运算符来组合LEFT JOIN和RIGHT JOIN，进而实现完全联结的思路还是存在问题的，会将两个表的内联结部分重复一次，即:

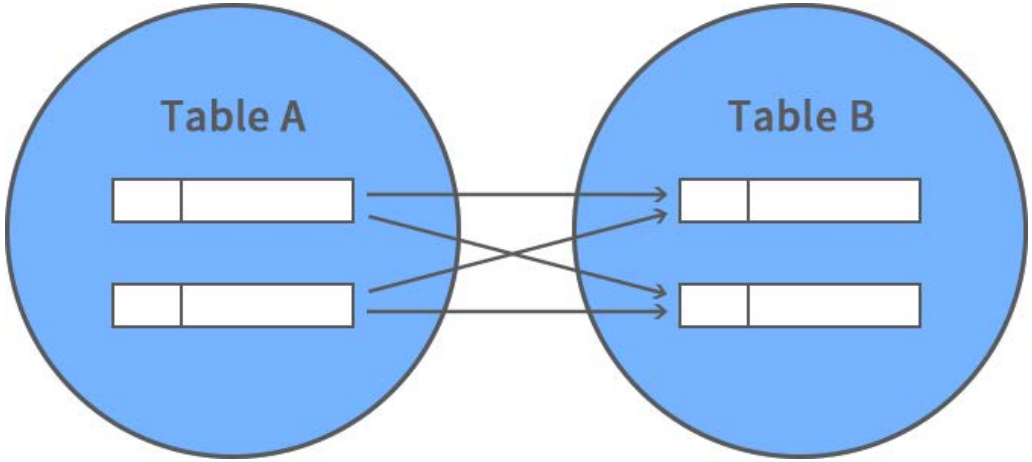
$$t_1 \cup t_2 + t_1 \cap t_2$$

问题所在

StockId	Stock	Symbol	Exchange
2021020100001	上证指数	000001	上海证券交易所
2021020100002	平安银行	000001	深圳证券交易所
2021020100003	万科 A	000002	深圳联合交易所
202000000004	嘉实增强信用定期债券	000005	
2021020100001	上证指数	000001	上海证券交易所
2021020100002	平安银行	000001	深圳证券交易所
2021020100003	万科 A	000002	深圳联合交易所
			香港联合交易所

交叉联结

交叉联结（CROSS JOIN）是将左表的每一行与右表的每一行合并。




```
SELECT t1.StockId, t1.Stock, t1.Symbol, t2.Exchange
    FROM t1 CROSS JOIN t2
```

补充内容：使用 Python 进行联表查询

基于 Python 的 Pandas 包，我们也可以进行联表查询。首先将两个表格转化为 DataFrame

```
import pandas as pd
t1 = pd.read_csv('t1.csv',
                 dtype={'StockId':str, 'Symbol':str})
t2 = pd.read_csv('t2.csv')
```

说明：

由于表t1当中的StockId与Symbol都是字符串，为防止 Python 自动将其视作数值，此处使用字典形式修改这两个字段的类型为字符型（str）

基于 Python 的联结

使用 Python 进行联表查询，要使用 Pandas 中的merge函数，具体命令如下：

```
df1 = pd.merge(t1, t2, how='inner', left_on='ExchangeId',
               right_on='Id') # 内联结
df2 = pd.merge(t1, t2, how='left', left_on='ExchangeId',
               right_on='Id') # 左联结
df3 = pd.merge(t1, t2, how='right', left_on='ExchangeId',
               right_on='Id') # 右联结
df4 = pd.merge(t1, t2, how='outer', left_on='ExchangeId',
               right_on='Id') # 完全联结（外联结）
```

联表查询结果的输出

以内联结为例，联表查询结果的输出代码如下：

```
print(df1[['StockId', 'Stock', 'Symbol', 'Exchange']])
```

这里的输出结果，与 SQL 语言得到的结果完全相同。

注意：

使用 Python 进行完全联结所得到的结果，与前面 SQL 语言得到的结果不同，输出的内容是两个表格的并集，且不存在记录重复的问题。

内联结的代码及输出结果

```
df1 = pd.merge(t1, t2, how='inner', left_on='ExchangeId',
               right_on='Id') # 内联结
print(df1[['StockId', 'Stock', 'Symbol', 'Exchange']])
```

	StockId	Stock	Symbol	Exchange
0	2021020100001	上证指数	000001	上海证券交易所
1	2021020100002	平安银行	000001	深圳证券交易所
2	2021020100003	万科 A	000002	深圳证券交易所

左联结的代码及输出结果

```
df2 = pd.merge(t1, t2, how='left', left_on='ExchangeId',
                right_on='Id') # 左联结
print(df2[['StockId', 'Stock', 'Symbol', 'Exchange']])
```

	StockId	Stock	Symbol	Exchange
0	2021020100001	上证指数	000001	上海证券交易所
1	2021020100002	平安银行	000001	深圳证券交易所
2	2021020100003	万科 A	000002	深圳证券交易所
3	202000000004	嘉实增强信用定期债券	000005	NaN

右联结的代码及输出结果

```
df3 = pd.merge(t1, t2, how='right', left_on='ExchangeId',
               right_on='Id') # 右联结
print(df3[['StockId', 'Stock', 'Symbol', 'Exchange']])
```

	StockId	Stock	Symbol	Exchange
0	2021020100002	平安银行	000001	深圳证券交易所
1	2021020100003	万科 A	000002	深圳证券交易所
2	2021020100001	上证指数	000001	上海证券交易所
3	NaN	NaN	NaN	香港联合交易所

完全联结的代码及输出结果

```
df4 = pd.merge(t1, t2, how='outer', left_on='ExchangeId',
               right_on='Id') # 完全联结（外联结）
print(df4[['StockId', 'Stock', 'Symbol', 'Exchange']])
```

	StockId	Stock	Symbol	Exchange
0	2021020100001	上证指数	000001	上海证券交易所
1	2021020100002	平安银行	000001	深圳证券交易所
2	2021020100003	万科 A	000002	深圳证券交易所
3	202000000004	嘉实增强信用定期债券	000005	NaN
4	NaN	NaN	NaN	香港联合交易所