零基础·速成 数据分析SQL 高手

吴 明 老 师

数据分析工作场景

Question：本季度，商品A销量为什么降低了？

- 曝光量低？
- 渠道问题？
- 转化率低？
- 竞争对手？

通过对业务【数据】进行分析，得出确定结论。

数据分析工作流程

支撑公司决策，引导业务发展，创造实际价值

企业数据存储在哪里？

纸质

Excel

数据库

如何从数据库中获取数据？

SQL（Structured Query Language），结构化查询语言

SQL

操作数据库

获取数据

```
select year(order_date), month(order_date), sum(sale) from
sales
group by year(order_date), month(order_date)
order by year(order_date), month(order_date);
```

数据分析与SQL价值

BI数据分析师

1.2-2.4万/月

职位信息

岗位职责：

1、参与设计汽车行业数据治理解决方案，并配合数据团队参与汽车主机厂的数据治理工作；

2、基于结构化数据，进行业务数据模型的分析设计；基于非结构化数据，设计识别算法及分析模型；

3、参与行业BI体系设计；

4、配合技术部门完成数据治理相关的技术实施；

5、 配合技术部门完成识别算法相关的系统架构设计及应用组件选型；

岗位要求：

1、 汽车、统计、数学、自动化等相关专业，本科及以上学历，***院校优先；

2、 比较成熟的结构化思维，善于用简单语言表述复杂结论;良好的数据敏感度，能从海量数据提炼核心结果；

3、熟练使用SQL、Excel、PPT，掌握Python者优先；

4、熟悉数据治理基本架构和方法论；

5、 熟悉主流的算法及应用；

6、强烈的责任感及敬业精神，良好的团队合作精神，具备一定的组织管理能力

职能类别：数据分析师

数据分析师（电商）

1-1.5万/月

职位信息

岗位职责：

1、 设计、构建和完善集团生态用工数据体系，为业务及管理层提供分析依据与决策支持，有效推动业务进展与风险可视化；

2、 能主动发现业务的需求独立完成分析，基于数据分析结果提供对业务的建议和洞见，驱动管理效率提升和运营策略优化；

3、 能完成数据清洗、编写较为复杂的数据提取语句，高效准确地横跨多业务产品线底层数据表提取数据，构建数据宽表、指标、看板，推动数据产品化的实施与落地。

4、 在对业务、场景和用户深刻理解的基础上，将业务问题抽象为数据问题，建立模型，并通过数据化运营手段解决。

职位要求：

1、 大专以上学历，有过开源BI或商业智能（BI）或数据产品工作经验3年以上，具备互联网、大型软件行业公司或知名咨询公司行业经验者优先；

2、数据处理能力强，掌握SQL、Excel、Python、Hive等数据工具；

3、 逻辑清晰，具备复杂业务场景的梳理能力，善于从复杂的业务场景中快速发现问题、挖掘分析思路、提炼结果、给出建议、推进落地；

4、 数学、统计学、计算机相关专业，本科或以上学历，为人踏实认真、乐观皮实、学习适应能力强。

数据表相关概念

数据库（Database）：存储数据的仓库，按【特定格式】存储的【数据文件】集合。

数据表：若干字段和记录组成

字段：数据类型相同的列

记录：一条数据

主键：唯一标识一行记录的字段

数据库：若干数据表组成

什么是关系型数据库？

关系型数据库：数据表之间，存在关联关系。

id	username	password	email
1	张三	zhangsan123	zhangsan@tdd.com
2	李四	lisi123	lisi@tdd.com
3	王五	wangwu123	wangwu@tdd.com
4	赵六	zhaoliu123	zhaoliu@tdd.com

user表

id	name
1	管理员
2	普通用户

role表

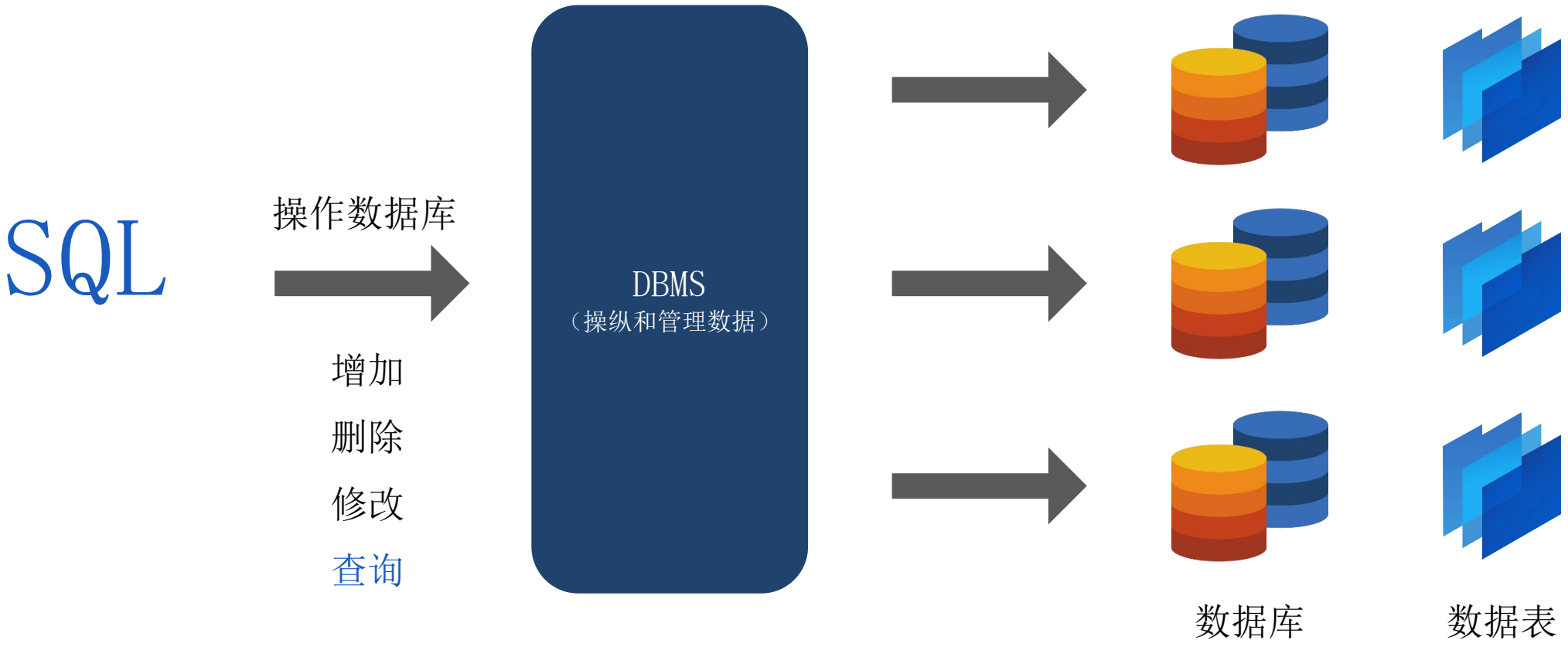
user_id	role_id
1	1
2	2
3	2

user_roles表

什么是DBMS？

DBMS（Database Management System），数据库管理系统。

针对数据库文件及数据进行【统一管理】的【软件系统】。



主流关系数据库产品

MySQL环境搭建

① 官网下载

② path环境变量配置（mysql的bin目录）

③ 执行命令（管理员权限进入命令提示符）

- mysqld --initialize-insecure
- mysqld -install
- net start MySQL

④ 登陆mysql（不用输入密码，直接回车）

- mysql -u root -p

⑤ 修改root用户密码(不要从课件中复制，自己输入，Office会自动把英文符号转换成中文的)

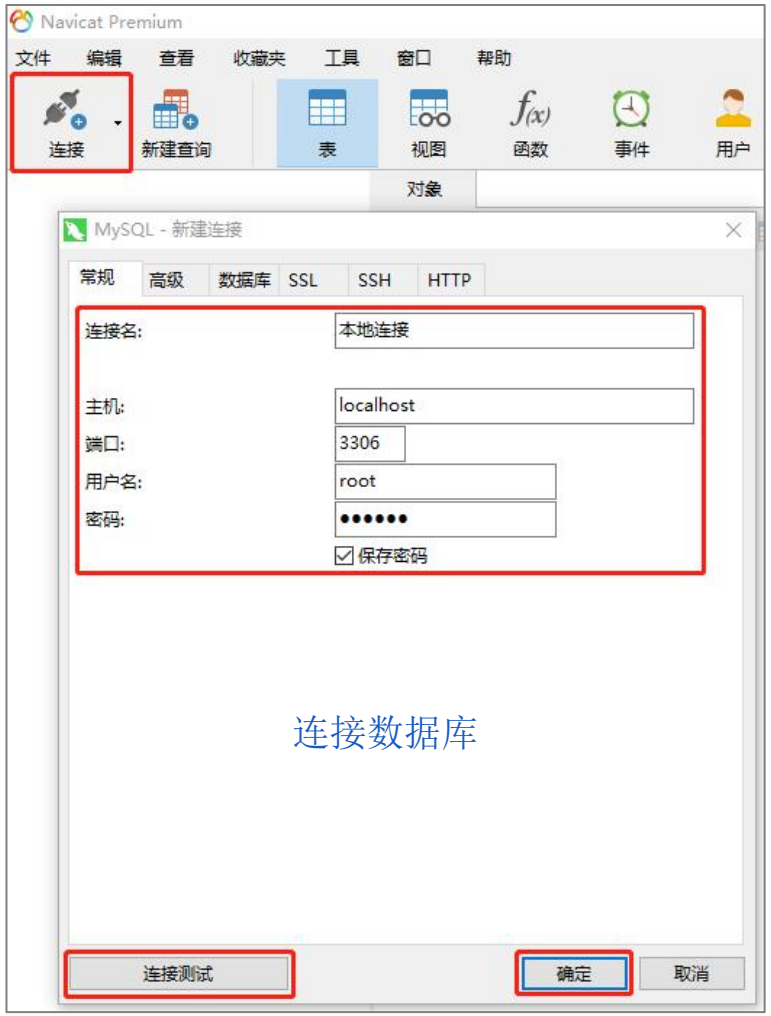
- ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '123456';
- flush privileges;

安装SQL编辑器

Navicat

DBeaver

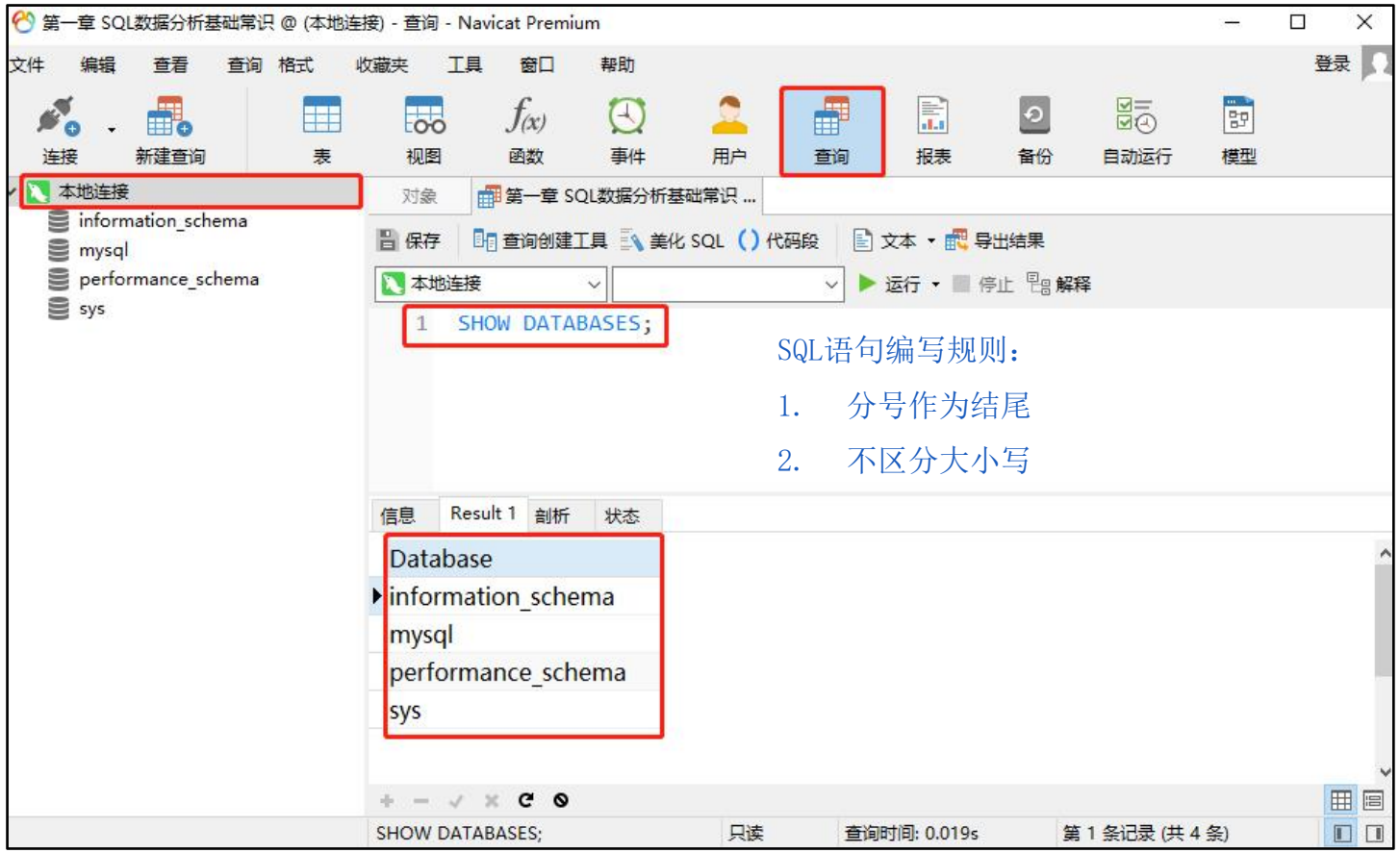
Visual Studio Code



连接数据库

Navicat基础操作

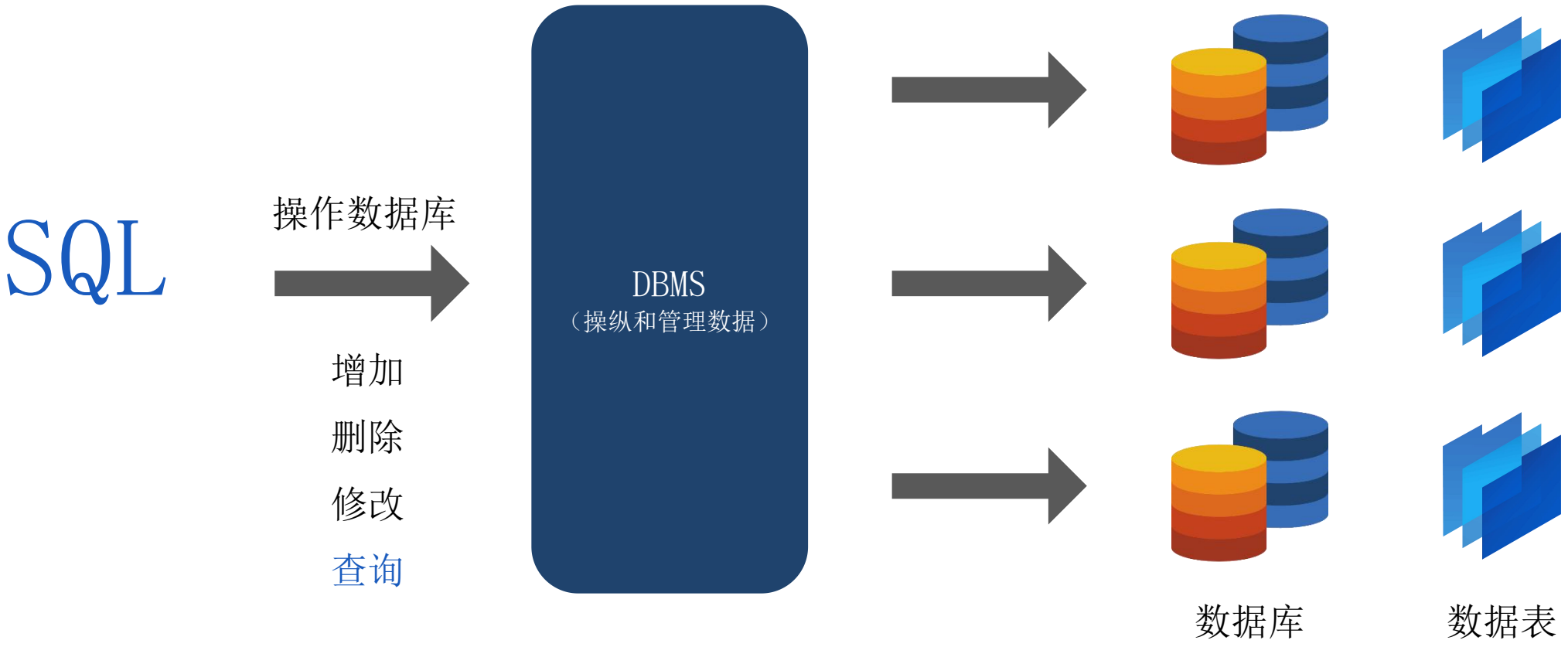
- 连接数据库
- 打开 - 关闭连接
- 使用查询窗口
  - 新建查询
  - 编写SQL语句
  - 运行SQL语句
  - 编写SQL技巧
  - 保存查询
- 设置字号大小



什么是DBMS？

DBMS（Database Management System），数据库管理系统。

针对数据库文件及数据进行【统一管理】的【软件系统】。



数据库相关操作

- 创建数据库

```
CREATE DATABASE chapter02;
```

- 查看创建数据库语句

```
SHOW CREATE DATABASE chapter02;
```

- 指定使用的数据库

```
USE mysql;
```

- 查看当前使用的数据库

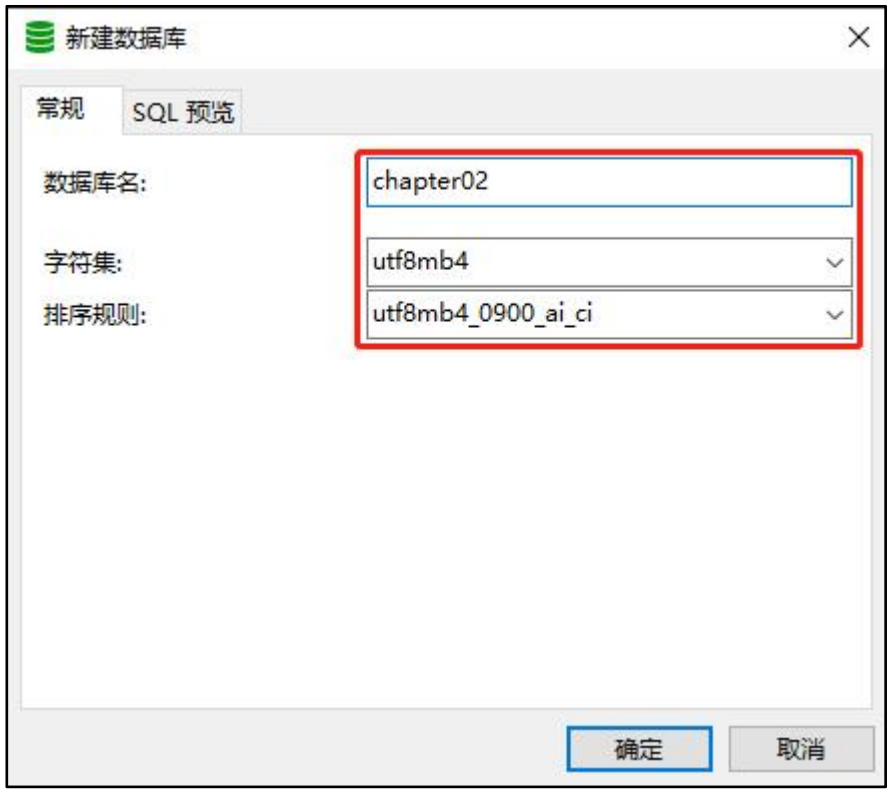
```
SELECT DATABASE();
```

- 查看当前数据库所有的表

```
SHOW TABLES;
```

- 删除数据库

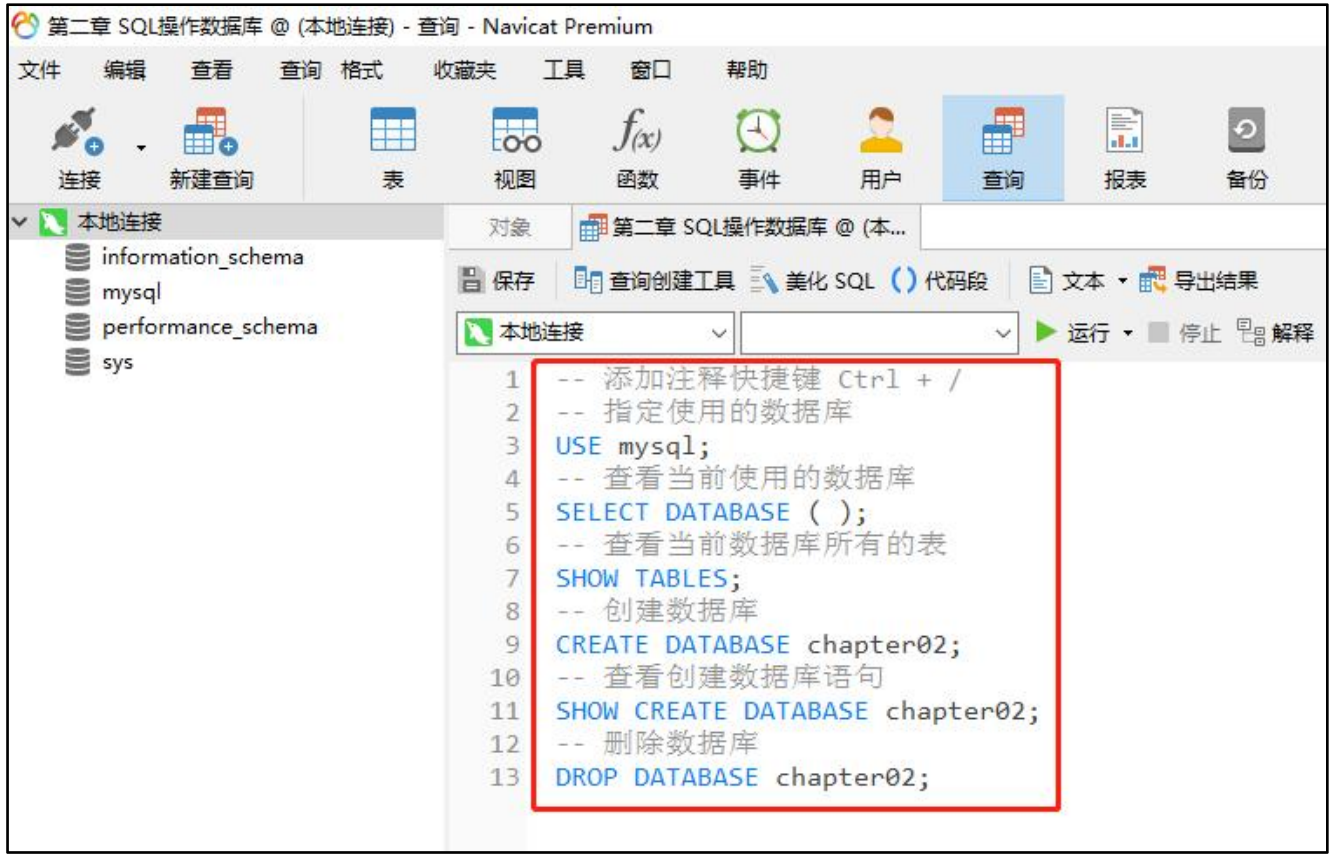
```
DROP DATABASE chapter02;
```



添加SQL语句注释

- #注释内容
- -- 注释内容

  快捷键 Ctrl + /
- /* 注释内容 */



数据表相关概念

数据库（Database）：存储数据的仓库，按【特定格式】存储的【数据文件】集合。

数据表：若干字段和记录组成

字段：数据类型相同的列

记录：一条数据

主键：唯一标识一行记录的字段

数据库：若干数据表组成

MySQL数据类型

- 字段：【数据类型相同】的列。
- 数据类型分类：数值型、字符串型、日期时间型。

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市
EMP-00005	孙七	女	29	顾问	9000	2021-10-13 09:00:00	南京市

数值型

- 1 Bytes（字节）

  01011011

- 年龄字段：正整数

  Age INT -> Age INT SIGNED

  Age INT UNSIGNED

整数型				
类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 Bytes	(-128，127)	(0，255)	小整数值
SMALLINT	2 Bytes	(-32768，32767)	(0，65535)	大整数值
MEDIUMINT	3 Bytes	(-8388608，8388607)	(0，16777215)	大整数值
INT或INTEGER	4 Bytes	(-2147483648，2147483647)	(0，4294967295)	大整数值
BIGINT	8 Bytes	(-9223372036854775808，9223372036854775807)	(0，18446744073709551615)	极大整数值

数值型

- FLOAT、DOUBLE超出精度，会自动进行四舍五入处理。

  例：Salary FLOAT (10, 2)，如存入数据为10000.005，则获取数据为10000.01。

- DECIMAL精度可达28位，用于应对科学及金融等数据精度要求高的场景。

  例：Salary Decimal (10, 2)

小数型				
类型	大小	范围（有符号）	范围（无符号）	精度
FLOAT	4 Bytes	(-3.402823466 E+38，-1.175494351 E-38)，0，(1.175 494351 E-38，3.402823466351 E+38)	0，(1.175494351 E-38，3.402823466 E+38)	7位
DOUBLE	8 Bytes	(-1.7976931348623157 E+308，-2.2250738585072014 E-308)，0，(2.225073858507 2014 E-308，1.7976931348623157 E+308)	0，(2.2250738585072014 E-308，1.7976931348623157 E+308)	15位

MySQL数据类型

- 字段：【数据类型相同】的列。
- 数据类型分类：数值型、字符串型、日期时间型。

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市
EMP-00005	孙七	女	29	顾问	9000	2021-10-13 09:00:00	南京市

字符串型

- CHAR是定长字符串、VARCHAR是变长字符串。

例：EmployeeID CHAR（ 20 ），存入字符串内容：’EMP-00001’，依然会占据20个字符的空间。

EmployeeID VARCHAR（ 20 ），存入字符串内容：’EMP-00001’，则只会占据9个字符的空间。

字符串型		
类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串

字符串型

- TEXT无法设置长度，占据空间固定。

  例：EmployeeID TEXT，存入字符串内容：’EMP-00001’，会占据65535个字符的空间。

文本字符串型		
类型	大小	用途
TINYTEXT	0-255 bytes	短文本字符串
TEXT	0-65535 bytes	长文本数据
MEDIUMTEXT	0-16777215 bytes	中等长度文本数据
LONGTEXT	0-4294 967295 bytes	极大文本数据

MySQL数据类型

- 字段：【数据类型相同】的列。
- 数据类型分类：数值型、字符串型、日期时间型。

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市
EMP-00005	孙七	女	29	顾问	9000	2021-10-13 09:00:00	南京市

日期时间型

- DATE类型字段，只能存储日期数据。DATETIME类型字段，则可以存储日期和时间。

  例：JoinedAt DATE，只能存入日期：’2022-01-05’。

  JoinedAt DATETIME，则可以存入日期和时间，’2022-01-05 10:10:00’。

- DATETIME和TIMSTAMP类型字段，都可以存储日期和时间，主要区别在于存储的范围不同。

日期时间型				
类型	大小	范围	格式	用途
DATE	3 bytes	1000-01-01-9999-12-31	YYYY-MM-DD	日期值
DATETIME	8 bytes	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 bytes	1970-01-01 00:00:00-2038 北京时间 2038-01-19 11:14:07AM 格林尼治时间 2038-01-19 03:14:07AM	YYYY-MM-DD hh:mm:ss	混合日期和时间值，时间戳

枚举类型

- 枚举类型作用：限制存入内容只能是固定值选一。

  例：Gender ENUM（'男','女'）

  JobPosition ENUM（'业务经理','开发人员','顾问'）

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市
EMP-00005	孙七	女	29	顾问	9000	2021-10-13 09:00:00	南京市

员工表数据类型确定

- EmployeeID CHAR（9）
- Name VARCHAR（50）
- Gender ENUM（'男','女'）
- Age INT UNSIGNED
- JobPosition ENUM（'业务经理','开发人员','顾问'）
- Salary DECIMAL（10, 2）
- JoinedAt DATETIME
- Address TEXT

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市
EMP-00005	孙七	女	29	顾问	9000	2021-10-13 09:00:00	南京市

数据表相关操作

- 创建数据表

```
CREATE TABLE Employees (
        EmployeeID CHAR ( 9 ),
        Name VARCHAR ( 50 ),
        Gender ENUM('男','女'),
        Age INT UNSIGNED,
        JobPosition ENUM ( '业务经理', '开发人员', '顾
问' ),
        Salary DECIMAL ( 10, 2 ),
        JoinedAt DATETIME,
        Address TEXT
);
```

- 查看表创建语句

```
SHOW CREATE TABLE Employees;
```

- 查看表结构

```
DESC Employees;
```

- 删除表

```
DROP TABLE Employees;
```

为数据表插入数据

- 全字段插入（插入值必须与表字段顺序完全一致）

```
INSERT INTO Employees VALUES ('EMP-00001', '张三', '男', 34, '业务经理', 20000, '2000-01-15 15:00:00', '上海市');
```

- 部分字段插入（插入值必须与指定字段顺序完全一致）

```
INSERT INTO Employees ( EmployeeID, Name, Gender, Age, JobPosition, Salary ) VALUES
        ('EMP-00002', '李四', '女', 28, '开发人员', 15000 );
```

对象 | * 第二章 SQL操作数据库 @ (... | employees @chapter03 (本...

开始事务 | 文本 | 筛选 | 排序 | 导入 | 导出

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	(Null)	(Null)

为数据表插入数据

- 全字段一次插入多行记录

```
INSERT INTO Employees VALUES
        ('EMP-00003', '王五', '男', 25, '开发人员', 10000, '2015-07-28 13:00:00', '北京市'),
        ('EMP-00004', '赵六', '女', 32, '顾问', 8000, '2019-11-12 18:00:00', '杭州市'),
        ('EMP-00005', '孙七', '女', 29, '顾问', 9000, '2021-10-13 09:00:00', '南京市');
```

- 部分字段一次插入多行记录

```
INSERT INTO Employees ( EmployeeID, Name, Gender, Age, JobPosition, Salary ) VALUES
        ('EMP-00006', '周八', '男', 28, '开发人员', 20000 ),
        ('EMP-00007', '杨九', '女', 33, '顾问', 7500 );
```

为数据表插入数据

- INSERT INTO Employees VALUES

  ('EMP-00008', '郑十', '男', -34, '业务经理', 23000, '2012-01-18', '上海市');

- INSERT INTO Employees VALUES

  ('EMP-00008', '郑十', '男', 34, '经理', 23000, '2012-01-18', '上海市');

- INSERT INTO Employees VALUES

  ('EMP-000081', '郑十', '男', 34, '业务经理', 23000, '2012-01-18', '上海市');

- INSERT INTO Employees VALUES

  (NULL, '郑十', '男', 34, '业务经理', 23000, '2012-01-18', '上海市');

数据表约束规则

➢ 非空（ NOT NULL ）

```
DROP TABLE Employees;
CREATE TABLE Employees (
            EmployeeID CHAR ( 9 ) NOT NULL,
            Name VARCHAR ( 50 ) NOT NULL,
            Gender ENUM ( '男', '女' ),
            Age INT UNSIGNED,
            JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
            Salary DECIMAL ( 10, 2 ) NOT NULL,
            JoinedAt DATETIME,
            Address TEXT
);
INSERT INTO Employees VALUES ( 'EMP-00001', '张三', NULL, NULL, '业务经理', 20000, NULL, NULL );
INSERT INTO Employees ( EmployeeID, Name, JobPosition, Salary ) VALUES ( 'EMP-00002', '李四', '开发人员',
15000 );
INSERT INTO Employees ( EmployeeID, Name, JobPosition ) VALUES ( 'EMP-00003', '王五', '开发人员' );
```

数据表约束规则

➢ 默认值（ DEFAULT ）

```
DROP TABLE Employees;
CREATE TABLE Employees (
            EmployeeID CHAR ( 9 ) NOT NULL,
            Name VARCHAR ( 50 ) NOT NULL,
            Gender ENUM ( '男', '女' ),
            Age INT UNSIGNED,
            JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
            Salary DECIMAL ( 10, 2 ) NOT NULL DEFAULT 0,
            JoinedAt DATETIME,
            Address TEXT
);
INSERT INTO Employees( EmployeeID, Name, JobPosition ) VALUES ( 'EMP-00003', '王五', '开发人员' );
```

数据表约束规则

➢ 唯一性（ UNIQUE ）

```
DROP TABLE Employees;
CREATE TABLE Employees (
            EmployeeID CHAR ( 9 ) NOT NULL UNIQUE,
            Name VARCHAR ( 50 ) NOT NULL,
            Gender ENUM ( '男', '女' ),
            Age INT UNSIGNED,
            JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
            Salary DECIMAL ( 10, 2 ) NOT NULL DEFAULT 0,
            JoinedAt DATETIME,
            Address TEXT
);
INSERT INTO Employees( EmployeeID, Name, JobPosition ) VALUES ( 'EMP-00001', '张三', '业务经理' );
INSERT INTO Employees( EmployeeID, Name, JobPosition ) VALUES ( 'EMP-00001', '李四', '开发人员' );
```

数据表约束规则

➢ 自动增长（ AUTO_INCREMENT ）

```
DROP TABLE Employees;
CREATE TABLE Employees (
            EmployeeID INT NOT NULL UNIQUE AUTO_INCREMENT,
            Name VARCHAR ( 50 ) NOT NULL,
            Gender ENUM ( '男', '女' ),
            Age INT UNSIGNED,
            JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
            Salary DECIMAL ( 10, 2 ) NOT NULL DEFAULT 0,
            JoinedAt DATETIME,
            Address TEXT
);
INSERT INTO Employees( Name, JobPosition ) VALUES ( '张三', '业务经理' );
INSERT INTO Employees( Name, JobPosition ) VALUES ( '李四', '开发人员' );
INSERT INTO Employees( Name, JobPosition ) VALUES ( '王五', '开发人员' );
```

主键与外键

- 设置主键（ PRIMARY KEY，非空且唯一 ）

```
CREATE TABLE Employees (
        EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
        Name VARCHAR ( 50 ) NOT NULL,
        Gender ENUM ( '男', '女' ),
        Age INT UNSIGNED,
        JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
        Salary DECIMAL ( 10, 2 ) NOT NULL DEFAULT 0,
        JoinedAt DATETIME,
        Address TEXT
);
```

```
CREATE TABLE Employees (
        EmployeeID INT AUTO_INCREMENT,
        Name VARCHAR ( 50 ) NOT NULL,
        Gender ENUM ( '男', '女' ),
        Age INT UNSIGNED,
        JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
        Salary DECIMAL ( 10, 2 ) NOT NULL DEFAULT 0,
        JoinedAt DATETIME,
        Address TEXT,
        PRIMARY KEY ( EmployeeID )
);
```

- 插入主键字段值时，可使用NULL占位，或省略字段。

主键与外键

- 为什么要使用外键？

员工表（Employees）								
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address	DeptName
11	张三	男	34	业务经理	20000	2000-01-15	上海市	管理部
22	李四	女	28	开发人员	15000	2010-03-22	南京市	研发部
33	王五	男	25	开发人员	10000	2015-07-28	北京市	研发部
44	赵六	女	32	顾问	8000	2019-11-12	杭州市	咨询部
55	孙七	女	29	顾问	9000	2021-10-13	南京市	咨询部

主键与外键

- 增加、更新错误

员工表（Employees）								
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address	DeptName
1	张三	男	34	业务经理	20000	2000-01-15	上海市	管理部
2	李四	女	28	开发人员	15000	2010-03-22	南京市	开发部
3	王五	男	25	开发人员	10000	2015-07-28	北京市	开发
4	赵六	女	32	顾问	8000	2019-11-12	杭州市	咨询部
5	孙七	女	29	顾问	9000	2021-10-13	南京市	咨询部

主键与外键

- 删除丢失

员工表（Employees）								
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address	DeptName
2	李四	女	28	开发人员	15000	2010-03-22	南京市	开发部
3	王五	男	25	开发人员	10000	2015-07-28	北京市	开发
4	赵六	女	32	顾问	8000	2019-11-12	杭州市	咨询部
5	孙七	女	29	顾问	9000	2021-10-13	南京市	咨询部

主键与外键

- 使用外键进行解决

员工表（Employees）								
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address	DeptID
1	张三	男	34	业务经理	20000	2000-01-15	上海市	1
2	李四	女	28	开发人员	15000	2010-03-22	南京市	2
3	王五	男	25	开发人员	10000	2015-07-28	北京市	2
4	赵六	女	32	顾问	8000	2019-11-12	杭州市	3
5	孙七	女	29	顾问	9000	2021-10-13	南京市	3

部门表（Department）	
DeptID	DeptName
1	管理部
2	开发部
3	咨询部

主键与外键

```
DROP TABLE IF EXISTS Employees;
DROP TABLE IF EXISTS Department;

CREATE TABLE IF NOT EXISTS Department (
            DeptID INT PRIMARY KEY AUTO_INCREMENT,
            DeptName VARCHAR ( 10 ) NOT NULL UNIQUE
);

INSERT INTO Department( DeptName ) VALUES
( '管理部' ), ( '开发部' ), ( '咨询部' );
```

```
CREATE TABLE IF NOT EXISTS Employees (
            EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
            Name VARCHAR ( 50 ) NOT NULL,
            Gender ENUM ( '男', '女' ),
            Age INT UNSIGNED,
            JobPosition ENUM ( '业务经理', '开发人员', '顾问' ) NOT NULL,
            Salary DECIMAL ( 10, 2 ) NOT NULL DEFAULT 0,
            JoinedAt DATETIME,
            Address TEXT,
            DeptID INT NOT NULL,
            FOREIGN KEY ( DeptID ) REFERENCES Department ( DeptID )
);
```

主键与外键

```
-- 插入数据
INSERT INTO employees ( Name, JobPosition, DeptID )
VALUES
        ( '张三', '业务经理', 1 ),
        ( '李四', '开发人员', 2 ),
        ( '王五', '开发人员', 2 ),
        ( '赵六', '顾问', 3 ),
        ( '孙七', '顾问', 3 );
```

```
-- 测试数据
INSERT INTO employees ( Name, JobPosition ) VALUES ( '周八', '开发人员' );
INSERT INTO employees ( Name, JobPosition, DeptID ) VALUES
( '周八', '开发人员', 4 );
```

外键在实际业务中应用并不是很多，会降低新增、删除、修改数据的效率。

SQL修改表结构

- 修改表名称

```
ALTER TABLE Employees RENAME TO Employee;
```

- 修改字段名称

```
ALTER TABLE Employee CHANGE COLUMN JoinedAt HireDate DATETIME;
```

- 修改数据类型和约束

```
ALTER TABLE Employee MODIFY COLUMN Name VARCHAR( 30 );
```

- 添加字段

```
ALTER TABLE Employee ADD COLUMN Birthday DATE;
```

- 删除字段

```
ALTER TABLE Employee DROP COLUMN Birthday;
```

- 删除主键

```
ALTER TABLE Employee DROP PRIMARY KEY;
```

- 添加主键

```
ALTER TABLE Employee ADD PRIMARY KEY ( EmployeeID );
```

- 添加外键

```
ALTER TABLE Employee ADD
FOREIGN KEY ( DeptID ) REFERENCES Department ( DeptID );
```

- 删除外键

```
ALTER TABLE Employee DROP FOREIGN KEY employee_ibfk_1;
```

查询数据

- 查询全字段数据

```
SELECT * FROM chapter4.Employee;

USE chapter04;

SELECT * FROM Employee;
```

- 查询指定字段数据

```
SELECT Name, Salary FROM Employee;

SELECT Employee.Name, Employee.Salary FROM Employee;
```

- 为字段取别名

```
SELECT Name AS "姓名", Salary AS "月薪" FROM Employee;

SELECT Employees.Name AS "姓名",  Employees.Salary AS "月薪" FROM Employee AS Employees;
```

- 根据主键获取某一条记录

```
SELECT Name AS "姓名", Salary AS "月薪" FROM Employee WHERE EmployeeID = 10;
```

修改、删除数据

- 修改全部数据

```
UPDATE Employee SET Salary = Salary + 1000;
```

```
UPDATE Employee SET Salary = Salary + 1000, age = age + 10;
```

- 修改指定数据

```
UPDATE Employee SET Salary = Salary + 1000 WHERE EmployeeID = 10;
```

- 删除全部数据

```
DELETE FROM Employee;
```

- 删除指定数据

```
DELETE FROM Employee WHERE EmployeeID = 10;
```

SQL基础查询

- 算术运算（+、-、*、/）

```
SELECT Name,Salary AS '涨薪前',Salary + 1000 AS '涨薪后' FROM Employee;

SELECT Name,Salary*13 AS '年薪' FROM Employee;
```

- 数据拼接（CONCAT、CONCAT_WS）

```
SELECT CONCAT(Name,Gender,Age) FROM Employee;

SELECT CONCAT(Name,',',Gender,',',Age) FROM Employee;

SELECT CONCAT_WS(',',Name,Gender,Age) FROM Employee;
```

- 数据排序（ORDER BY）

```
SELECT Name, Salary FROM Employee ORDER BY Salary ASC;

SELECT Name, Salary FROM Employee ORDER BY Salary DESC;

SELECT Name, Salary, Age FROM Employee ORDER BY Salary, Age DESC;
```

SQL基础查询

- 限制行数（ LIMIT 起始行号,行数 ）

```
SELECT EmployeeID, Name, Salary FROM Employee LIMIT 10;

SELECT EmployeeID, Name, Salary FROM Employee LIMIT 5,10;

SELECT EmployeeID, Name, Salary FROM Employee ORDER BY Salary DESC LIMIT 10;
```

- 数据去重（DISTINCT）

```
SELECT DISTINCT Gender FROM Employee;
```

SQL条件查询

- 比较运算查询（ >、<、>=、<=、=、<>、!= ）

```
SELECT * FROM Employee WHERE Age = 33;

SELECT * FROM Employee WHERE Age <> 33;

SELECT * FROM Employee WHERE Age != 33;

SELECT * FROM Employee WHERE Age > 30;

SELECT * FROM Employee WHERE Age >= 30;

SELECT * FROM Employee WHERE Age < 30;

SELECT * FROM Employee WHERE Age <= 30;

SELECT * FROM Employee WHERE Salary > 20000;

SELECT * FROM Employee WHERE JobPosition = '开发人员';

SELECT * FROM Employee WHERE JoinedAt > '2020-01-01';
```

SQL条件查询

- 逻辑运算查询（ AND、OR、NOT ）

```
SELECT * FROM Employee WHERE Age >= 40 AND Salary > 20000;

SELECT * FROM Employee WHERE Age >= 40 AND Salary > 20000 AND Gender = '男';

SELECT * FROM Employee WHERE Age >= 40 OR Salary > 20000;

SELECT * FROM Employee WHERE NOT ( Age >= 40 AND Salary > 20000 );
```

- 范围查询（ IN、NOT IN ）（ BETWEEN AND ）

```
SELECT * FROM Employee WHERE Age IN ( 32, 33, 34 );

SELECT * FROM Employee WHERE JobPosition IN ( '开发人员', '顾问' );

SELECT * FROM Employee WHERE JobPosition NOT IN ( '开发人员', '顾问' );

SELECT * FROM Employee WHERE Age BETWEEN 32 AND 34;

SELECT * FROM Employee WHERE JoinedAt BETWEEN '2022-01-01' AND '2022-04-01';
```

SQL条件查询

- 空值查询（ IS NULL、IS NOT NULL ）

```
SELECT * FROM Employee WHERE JoinedAt IS NULL;

SELECT * FROM Employee WHERE JoinedAt IS NOT NULL;
```

- 模糊查询（ %，任意多个字符 ，0个、1个或多个 ）（ _，单个字符）

```
SELECT * FROM Employee WHERE Name LIKE '杨%';

SELECT * FROM Employee WHERE Name LIKE '杨_';

SELECT * FROM Employee WHERE Name LIKE '%国%';

SELECT * FROM Employee WHERE Name LIKE '__';
```

聚合函数统计查询

- 计数（ COUNT ）

```
SELECT COUNT(*) FROM Employee;

SELECT COUNT(*) FROM Employee WHERE Gender = '男' ;

SELECT COUNT(*) FROM Employee WHERE Age <= 30;
```

- 最大值（ MAX ）

```
SELECT MAX(Age) FROM Employee;

SELECT MAX(Salary) FROM Employee WHERE JobPosition = '业务经理' ;
```

- 最小值（ MIN ）

```
SELECT MIN(Age) FROM Employee;

SELECT MIN(Salary) FROM Employee WHERE Gender = '男';
```

聚合函数统计查询

- 求和（ SUM ）

```
SELECT SUM(Salary) FROM Employee;

SELECT SUM(Salary) / COUNT(*) FROM Employee;
```

- 平均值（ AVG ）

```
SELECT AVG(Salary) FROM Employee;

SELECT AVG(Age) FROM Employee;

SELECT AVG(Age), AVG(Salary) FROM Employee;
```

- 结合DISTINCT使用聚合函数

```
SELECT COUNT(DISTINCT JobPosition) FROM Employee;
```

- 注意事项：字段为NULL的数据，会被忽略。

分组查询的原理

```
SELECT JobPosition, AVG(Salary) FROM Employee WHERE EmployeeID <= 10 GROUP BY JobPosition;
```

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
2	林国	女	29	顾问	14400	2015/6/15
3	林玟	女	26	业务经理	12200	2019/12/8
4	林雅	女	30	顾问	28400	2021/6/12
5	江奕	女	33	业务经理	26500	2018/3/23
6	刘柏	男	26	顾问	13300	2018/12/26
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20
10	吉茹	男	30	业务经理	19800	2021/1/18

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
3	林玟	女	26	业务经理	12200	2019/12/8
5	江奕	女	33	业务经理	26500	2018/3/23
10	吉茹	男	30	业务经理	19800	2021/1/18
				业务经理	19500	

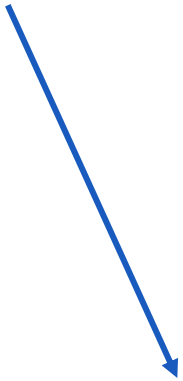
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20
				开发人员	21250	

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
2	林国	女	29	顾问	14400	2015/6/15
4	林雅	女	30	顾问	28400	2021/6/12
6	刘柏	男	26	顾问	13300	2018/12/26
				顾问	18700	

分组查询的原理

```
SELECT JobPosition, AVG(Salary) FROM Employee WHERE EmployeeID <= 10 GROUP BY JobPosition;
```

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
2	林国	女	29	顾问	14400	2015/6/15
3	林玟	女	26	业务经理	12200	2019/12/8
4	林雅	女	30	顾问	28400	2021/6/12
5	江奕	女	33	业务经理	26500	2018/3/23
6	刘柏	男	26	顾问	13300	2018/12/26
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20
10	吉茹	男	30	业务经理	19800	2021/1/18



JobPosition	Salary
开发人员	21250
顾问	18700
业务经理	19500

GROUP BY分组

```
SELECT JobPosition, AVG(Salary) FROM Employee GROUP BY JobPosition;

SELECT JobPosition, AVG(Age) FROM Employee GROUP BY JobPosition;

SELECT Gender, COUNT(*) FROM Employee GROUP BY Gender;


SELECT JobPosition, Gender, COUNT(*) FROM Employee

WHERE EmployeeID <= 10

GROUP BY JobPosition, Gender

ORDER BY JobPosition, Gender;
```

GROUP BY分组

```
SELECT JobPosition, Gender, COUNT(*) FROM Employee
WHERE EmployeeID <= 10
GROUP BY JobPosition, Gender
ORDER BY JobPosition, Gender;
```

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
2	林国	女	29	顾问	14400	2015/6/15
3	林玟	女	26	业务经理	12200	2019/12/8
4	林雅	女	30	顾问	28400	2021/6/12
5	江奕	女	33	业务经理	26500	2018/3/23
6	刘柏	男	26	顾问	13300	2018/12/26
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20
10	吉茹	男	30	业务经理	19800	2021/1/18

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
3	林玟	女	26	业务经理	12200	2019/12/8
5	江奕	女	33	业务经理	26500	2018/3/23
10	吉茹	男	30	业务经理	19800	2021/1/18

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
2	林国	女	29	顾问	14400	2015/6/15
4	林雅	女	30	顾问	28400	2021/6/12
6	刘柏	男	26	顾问	13300	2018/12/26

GROUP BY分组

```
SELECT JobPosition, Gender, COUNT(*) FROM Employee
WHERE EmployeeID <= 10
GROUP BY JobPosition, Gender
ORDER BY JobPosition, Gender;
```

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
2	林国	女	29	顾问	14400	2015/6/15
3	林玟	女	26	业务经理	12200	2019/12/8
4	林雅	女	30	顾问	28400	2021/6/12
5	江奕	女	33	业务经理	26500	2018/3/23
6	刘柏	男	26	顾问	13300	2018/12/26
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20
10	吉茹	男	30	业务经理	19800	2021/1/18

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
3	林玟	女	26	业务经理	12200	2019/12/8
5	江奕	女	33	业务经理	26500	2018/3/23

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
10	吉茹	男	30	业务经理	19800	2021/1/18

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20

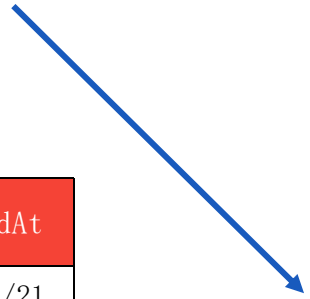
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
2	林国	女	29	顾问	14400	2015/6/15
4	林雅	女	30	顾问	28400	2021/6/12

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
6	刘柏	男	26	顾问	13300	2018/12/26

GROUP BY分组

```
SELECT JobPosition, Gender, COUNT(*) FROM Employee
WHERE EmployeeID <= 10
GROUP BY JobPosition, Gender
ORDER BY JobPosition, Gender;
```

EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt
1	张吉	男	29	开发人员	19000	2014/5/21
2	林国	女	29	顾问	14400	2015/6/15
3	林玟	女	26	业务经理	12200	2019/12/8
4	林雅	女	30	顾问	28400	2021/6/12
5	江奕	女	33	业务经理	26500	2018/3/23
6	刘柏	男	26	顾问	13300	2018/12/26
7	阮建安	男	22	开发人员	22500	2018/12/29
8	林子	男	35	开发人员	25700	2020/4/15
9	夏志豪	男	30	开发人员	17800	2019/3/20
10	吉茹	男	30	业务经理	19800	2021/1/18



JobPosition	Gender	COUNT(*)
业务经理	男	1
业务经理	女	2
开发人员	男	4
顾问	男	1
顾问	女	2

GROUP BY分组

```
SELECT JobPosition, Gender, COUNT(*), AVG(Salary)
FROM Employee
WHERE EmployeeID <= 10
GROUP BY JobPosition, Gender
ORDER BY JobPosition, Gender;
```

分组后过滤（HAVING）

```
SELECT Gender, COUNT(*) FROM Employee GROUP BY Gender HAVING Gender = '男';

SELECT Age, ROUND(AVG(Salary),0) FROM Employee GROUP BY Age HAVING Age >=30 ORDER BY Age;

SELECT Age, ROUND(AVG(Salary),0) AS AVG_Salary FROM Employee
GROUP BY Age HAVING AVG_Salary >= 20000 ORDER BY AVG_Salary;

SELECT Age, ROUND(AVG(Salary),0) AS AVG_Salary FROM Employee
WHERE JobPosition = '开发人员'
GROUP BY Age HAVING AVG_Salary >= 21000 ORDER BY Age;
```

- WHERE是针对表数据进行过滤
- HAVING是针对分组后数据进行过滤

查询语句执行顺序

```
SELECT
Age, ROUND(AVG(Salary),0)  AS AVG_Salary
FROM Employee
WHERE JobPosition = '开发人员'
GROUP BY Age
HAVING AVG_Salary >= 21000
ORDER BY Age DESC
LIMIT 3;
```

查询语句执行顺序

```
-- 获取表数据
SELECT * FROM Employee;
-- 通过WHERE条件进行过滤
SELECT * FROM Employee WHERE JobPosition = '开发人员';
-- 针对过滤后数据进行分组，执行聚合函数
SELECT ROUND(AVG(Salary),0) AS AVG_Salary FROM Employee WHERE JobPosition = '开发人员' GROUP BY Age;
-- 针对分组后数据，进行HAVING过滤
SELECT ROUND(AVG(Salary),0) AS AVG_Salary  FROM Employee WHERE JobPosition = '开发人员' GROUP BY Age HAVING AVG_Salary >= 21000;
-- 过滤分组数据后，进行查询，再排序
SELECT Age, ROUND(AVG(Salary),0) AS AVG_Salary FROM Employee WHERE JobPosition = '开发人员'
GROUP BY Age HAVING AVG_Salary >= 21000 ORDER BY Age;
-- 针对排序结果，限制展示行数
SELECT Age, ROUND(AVG(Salary),0) AS AVG_Salary FROM Employee WHERE JobPosition = '开发人员'
GROUP BY Age HAVING AVG_Salary >= 21000 ORDER BY Age DESC LIMIT 3;
```

查询语句执行顺序

```
SELECT
Age, ROUND(AVG(Salary),0)  AS AVG_Salary
FROM Employee
WHERE JobPosition = '开发人员'
GROUP BY Age
HAVING AVG_Salary >= 21000
ORDER BY Age DESC
LIMIT 3;
```

FROM > WHERE >【GROUP BY > 聚合函数 > HAVING】> SELECT > DISTINCT > ORDER BY > LIMIT

GROUP_CONCAT函数

```
SELECT Age, GROUP_CONCAT(Name) FROM Employee WHERE JobPosition = '开发人员' GROUP BY Age;
```

Age	GROUP_CONCAT(Name)
22	阮建安,王美,黄芸欢,周白,李小爱,赖怡,吴怡婷,钟庭,周筠,陈韦,许岳,陈怡,余仪礼,唐盛人,陈信,陈意婷,叶于,李文荣,王香君,宋廷
23	邱雅,张冠,许雅婷,蔡佳,杨淑,龚静,陈正
24	丁汉,林佩,黄育,潘欣臻,卢志,黄雅慧,何伶元,郑雅,侯文贤,周妤军,张伦启,张佩,王怡贵
25	李雅惠,陈信峰,徐紫,黄佩,邱宜,白凯修,王雅雯,刘小,涂武盛,陈政,连书,袁哲仪,林钰,王宗,陈婉,黄圣
26	沈俊君,陈建,陈仲,姜佩珊,秦欣瑜,詹允坚,谢姿君,陈佳
27	邓诗涵,吴思翰,陈枝,谢佳,胡东,陈雅,陈雅萍,陈俊,陈伟孝,陈宗馨
28	吴美,吴心真,韩健,查瑜,林孟富,蔡国伟,许承,唐欣,黄雅慧,阮肇宪,吴佩仁
29	张吉,王恩,林孟,王雅,林乐,谢晓,林佳蓉,林志嘉,林怡
30	夏志豪,方一,潘吉,李必,徐采伶,蔡玉婷,张琇纶,吴家,潘怡,张峻,徐景,黄菁坚,刘莹睿,吴承
31	蔡宜,黄柏仪,王怡,彭郁婷,韩宁,杨大雪,谢健铭,吴孟钰,陈秀琬,温燕,赖俊军,杜怡臻,陈铭,叶得梅,詹南,黄善,许淑玫,冯萱雨
32	张姿,陈淑妤,陈素达,柯乔,黄彦,张佳,张瑞,白怡均,吴乔茂,邱萱,周立,陈淑婷,刘怡,许宜
33	林婉,黄伟依,黄康,陈育,黄韦,傅予名,叶佩璇,林玉信,郑建泉,许金
34	李育泉,卢木仲,张虹,张政,张俞,郑士,潘右博,叶茜,谢怡君,谢懿富,叶惟,王雅,郑婉,邱承,蒋佳,张雅菱,林孟
35	林子,苏玮,赵吟琪,陈嘉,何珮甄,邓幸,叶怡财,冯雅筑,戎郁文,黄静,陈雅雯

WITH ROLLUP

```
SELECT JobPosition, Gender, COUNT(*), SUM(Salary) FROM Employee GROUP BY JobPosition, Gender WITH ROLLUP;

SELECT JobPosition, Gender, COUNT(*), AVG(Salary) FROM Employee GROUP BY JobPosition, Gender WITH ROLLUP;
```

JobPosition	Gender	COUNT(*)	SUM(Salary)
业务经理	男	94	2046100
业务经理	女	84	1733200
业务经理	(Null)	178	3779300
开发人员	男	98	2075400
开发人员	女	80	1669000
开发人员	(Null)	178	3744400
顾问	男	99	2012900
顾问	女	87	1814200
顾问	(Null)	186	3827100
(Null)	(Null)	542	11350800

JobPosition	Gender	COUNT(*)	AVG(Salary)
业务经理	男	94	21767.021277
业务经理	女	84	20633.333333
业务经理	(Null)	178	21232.022472
开发人员	男	98	21177.55102
开发人员	女	80	20862.5
开发人员	(Null)	178	21035.955056
顾问	男	99	20332.323232
顾问	女	87	20852.873563
顾问	(Null)	186	20575.806452
(Null)	(Null)	542	20942.435424

SQL常用数值函数

- ABS，返回绝对值
- POW，幂计算
- SIGN，判断正负数
- RAND，返回一个0到1的随机值
- CEIL，返回大于等于指定值的最小整数值
- FLOOR，返回小于等于指定值的最大整数值
- ROUND，返回指定小数点位数的四舍五入值
- TRUNCATE，返回指定小数点位数的截断值
- FORMAT，数值格式化

SQL常用文本函数

- LTRIM、RTRIM、TRIM，文本空格处理
- CONCAT、CONCAT_WS，文本连接
- REPLACE，文本替换
- LEFT、RIGHT、SUBSTRING，文本截取
- LENGTH，文本计数
- INSTR，文本查找
- UPPER、LOWER，文本大小写

SQL常用日期时间函数

- 获取当前日期和时间

  CURDATE、CURTIME、NOW

- 获得日期和时间组成部分

  DATE / YEAR / QUARTER / MONTH / DAY / WEEKOFYEAR

  TIME / HOUR / MINUTE / SECOND

  EXTRACT

- 日期和时间格式设置

  DATE_FORMAT

    %Y（年），%m（月），%d（日），%H（时），%i（分），%S（秒）

- 日期计算

  DATE_ADD

  DATEDIFF

SQL空值函数

- ISNULL，判断数据是否为空
- IFNULL，如果数据为空，返回替换值

CAST和COALESCE函数

- CAST，数据类型转换函数
- COALESCE，返回数据列表中的第一个非空值

Value	Description
DATETIME	转换为日期格式："YYYY-MM-DD HH:MM:SS"
DATE	转换为日期格式："YYYY-MM-DD"
TIME	转换为时间格式："HH:MM:SS"
DECIMAL	转换为小数格式
CHAR	转换为字符格式

IF函数

- IF(条件, 结果1, 结果2)

```
SELECT IF( 5 < 2, 1, 0);

SELECT
*, IF(Salary < 20000, 'LOW', 'HIGH') AS SAL_GRADE
FROM Employee ORDER BY SAL_GRADE;

SELECT
*,
IF(Salary < 20000, 'LOW' ,
          IF(Salary <= 25000, 'MEDIUM', 'HIGH')) AS SAL_GRADE
FROM Employee ORDER BY SAL_GRADE;

SELECT
COUNT(*),
IF(Salary < 20000, 'LOW', IF(Salary <= 25000, 'MEDIUM', 'HIGH')) AS SAL_GRADE
FROM Employee GROUP BY SAL_GRADE;
```

CASE函数

```
CASE
    WHEN 条件1 THEN 结果1
    WHEN 条件2 THEN 结果2
                ……
    WHEN 条件N THEN 结果N
    ELSE 结果
END;
```

```
SELECT Name, JobPosition,
(CASE
        WHEN Salary < 20000 THEN 'LOW'
        WHEN Salary BETWEEN 20000 AND 25000 THEN 'MEDIUM'
        ELSE 'HIGH'
END) AS SAL_GRADE
FROM Employee ORDER BY SAL_GRADE;

SELECT
COUNT(*),
(CASE
        WHEN Salary < 20000 THEN 'LOW'
        WHEN Salary BETWEEN 20000 AND 25000 THEN 'MEDIUM'
        ELSE 'HIGH'
END) AS SAL_GRADE
FROM Employee GROUP BY SAL_GRADE;
```

CASE函数

```
CASE 字段
    WHEN 值1 THEN 结果1
    WHEN 值2 THEN 结果2
                ……
    WHEN 值N THEN 结果N
    ELSE 结果
END;
```

```
SELECT
Name,
JobPosition,
(CASE JobPosition
        WHEN '开发人员' THEN '开发部'
        WHEN '顾问' THEN '咨询部'
        WHEN '业务经理' THEN '管理部'
        ELSE '其他'
END
) AS DepartmentName
FROM Employee
ORDER BY DepartmentName;
```

CASE转置应用

订单信息表（OrderInfo）		
CustomerName	Month	Amount
张三	1月	50
张三	2月	100
张三	3月	200
李四	1月	70
李四	2月	100
李四	3月	150
王五	1月	200
王五	2月	300
王五	3月	800

转置结果			
Customer	1月	2月	3月
张三	50	100	200
李四	70	100	150
王五	200	300	800

CASE转置应用

订单信息表（OrderInfo）		
CustomerName	Month	Amount
张三	1月	50
张三	2月	100
张三	3月	200
李四	1月	70
李四	2月	100
李四	3月	150
王五	1月	200
王五	2月	300
王五	3月	800

CASE过渡处理			
Customer	1月	2月	3月
张三	50	0	0
张三	0	100	0
张三	0	0	200
李四	70	0	0
李四	0	100	0
李四	0	0	150
王五	200	0	0
王五	0	300	0
王五	0	0	800

GROUP BY汇总			
Customer	1月	2月	3月
张三	50	100	200
李四	70	100	150
王五	200	300	800

为什么要使用多表连接？

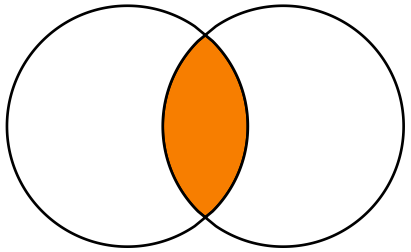
员工表（Employee）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	3
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	4
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	5
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		

部门表（Department）	
DeptID	DeptName
1	总部
2	开发部
3	咨询部
4	财务部
5	行政部
6	人力部

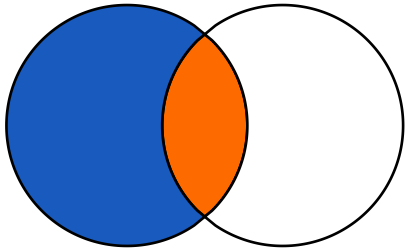
经理表（Manager）		
ManagerID	ManagerName	DeptID
1	黄文隆	1
2	张吉	2
3	林玟	3
4	林雅	3
5	江奕	3
6	刘柏	4
7	吉茹	4

关联查询类型

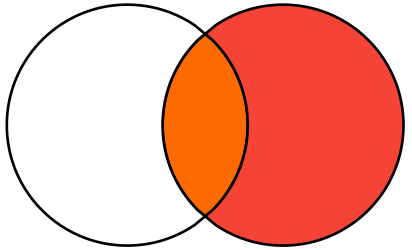
INNER JOIN（内连接）



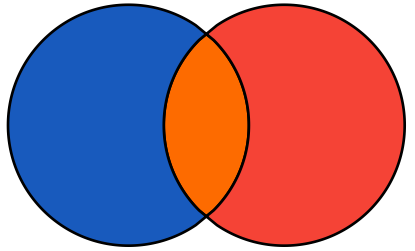
LEFT JOIN（左连接）



RIGHT JOIN（右连接）



FULL JOIN（全连接）



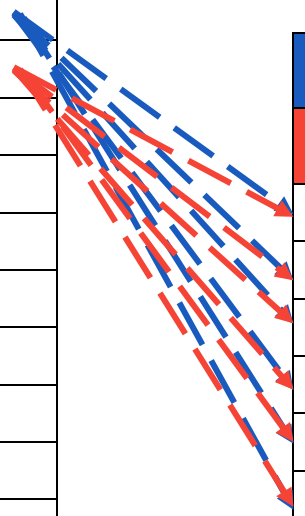
CROSS JOIN（交叉连接）

NATURAL JOIN（自然连接）

SELF JOIN（自连接）

什么是笛卡尔积？

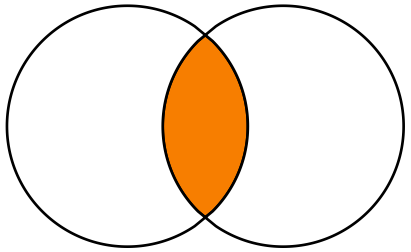
员工表（Employee）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	3
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	4
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	5
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		



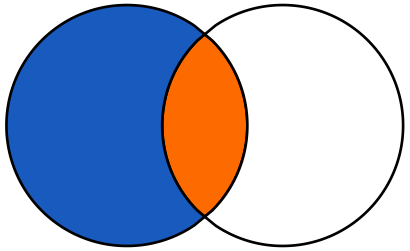
部门表（Department）	
DeptID	DeptName
1	总部
2	开发部
3	咨询部
4	财务部
5	行政部
6	人力部

关联查询类型

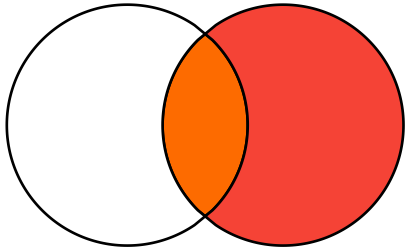
INNER JOIN（内连接）



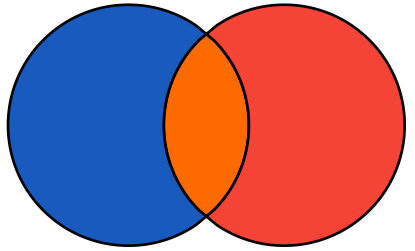
LEFT JOIN（左连接）



RIGHT JOIN（右连接）



FULL JOIN（全连接）



CROSS JOIN（交叉连接）

NATURAL JOIN（自然连接）

SELF JOIN（自连接）

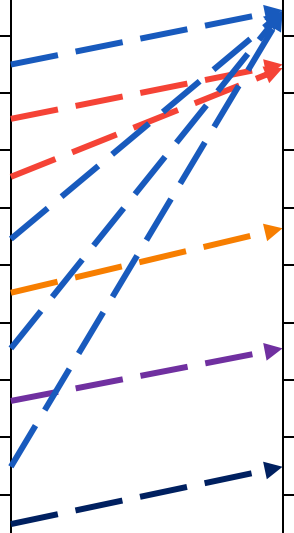
NATURAL JOIN（自然连接）

员工表（EmployeeVar）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	3
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	4
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	5
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		

部门表（Department）	
DeptID	DeptName
1	总部
2	开发部
3	咨询部
4	财务部
5	行政部
6	人力部

SELF JOIN（自连接）

员工表（EmployeeVar）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	5
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	7
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	9
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		



员工表（EmployeeVar）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	5
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	7
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	9
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		

为什么要使用子查询？

员工表（Employee）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	3
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	4
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	5
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		

1. 查询平均工资
2. 查询高于此平均工资的员工信息

子查询类型

1. *单行单列*
2. *多行多列*
3. *多行单列*

关联子查询

查询各部门高于部门平均薪资的员工信息

```
SELECT e1.*
FROM Employee AS e1
WHERE e1.Salary > (
          SELECT AVG(Salary)
          FROM Employee AS e2
          WHERE e2.DeptID = e1.DeptID
);
```

EXISTS与NOT EXISTS

查询不存在员工的部门名称

```
SELECT * FROM Department AS d
WHERE NOT EXISTS(
          SELECT * FROM Employee AS e
          WHERE e.DeptID = d.DeptID
);

SELECT * FROM Department AS d
WHERE NOT EXISTS(
          SELECT 1 FROM Employee AS e
          WHERE e.DeptID = d.DeptID
);
```

子查询应用场景

WHERE中使用子查询

FROM中使用子查询

SELECT中使用子查询

HAVING中使用子查询

CREATE中使用子查询

INSERT中使用子查询

UPDATE中使用子查询

DELETE中使用子查询

UNION与UNION ALL

差异点：UNION会去重，UNION ALL则不会

Tips 1：列数量需保持一致

Tips 2：列顺序需保持一致

WITH语句

- 增强SQL可读性
- 维护便利

WITH循环递归

- 获取1-10的数字

```
WITH RECURSIVE temp AS
(
        SELECT 1 AS n          -- 起始条件
        union
        SELECT n + 1           -- 基于起始条件，加1
        FROM temp  -- 从temp表中获取数据
        WHERE n < 10           -- 终止条件判断
)
SELECT * FROM temp;
```

执行顺序：1 2 3 4 2 3 4 2 3 4…

temp

n
1
2
3
4
5
6
7
8
9
10

WITH循环递归

- 自上而下，获取员工等级结构

员工表（EmployeeVar）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	5
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	7
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	9
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		

员工工号	员工姓名	经理姓名	Level
1	黄文隆	(Null)	1
2	张吉	黄文隆	2
5	林玟	黄文隆	2
7	林雅	黄文隆	2
9	江奕	黄文隆	2
3	谢彦	张吉	3
4	傅智翔	张吉	3
6	荣姿康	林玟	3
8	雷进宝	林雅	3
10	李雅惠	江奕	3

WITH循环递归

- 自上而下，获取员工等级结构

员工表（EmployeeVar）					
EmpID	EmpName	JobPosition	Salary	DeptID	ManagerID
1	黄文隆	CEO	100000	1	
2	张吉	经理	30000	2	1
3	谢彦	开发人员	15000	2	2
4	傅智翔	开发人员	12000	2	2
5	林玟	经理	35000	3	1
6	荣姿康	顾问	12000	3	5
7	林雅	经理	32500	3	1
8	雷进宝	顾问	12500	3	7
9	江奕	经理	31000	3	1
10	李雅惠	顾问	26000	3	9
11	吴佳瑞	HR	12000		
12	周琼玟	会计	14000		
13	黄文	行政	19000		

```
WITH RECURSIVE EmpHierarchy AS
(
        SELECT EmpID, EmpName, ManagerID
        FROM EmployeeVar AS e1 WHERE JobPosition = 'CEO'
        UNION
        SELECT e2.EmpID, e2.EmpName, e2.ManagerID
        FROM EmployeeVar AS e2
        JOIN EmpHierarchy AS temp1
        ON e2.ManagerID = temp1.EmpID
)
```

EmpHierarchy

EmpID	EmpName	ManagerID
1	黄文隆	
2	张吉	1
5	林玟	1
7	林雅	1
9	江奕	1
3	谢彦	2
4	傅智翔	2
6	荣姿康	5
8	雷进宝	7
10	李雅惠	9

窗口函数简介

```
SELECT *, MAX(UnitPrice) OVER() AS MaxUnitPrice FROM OrderInfo;

SELECT *, AVG(UnitPrice) OVER() AS AvgUnitPrice FROM OrderInfo;

SELECT *, MAX(UnitPrice) OVER() AS MaxUnitPrice, AVG(UnitPrice) OVER() AS AvgUnitPrice
FROM OrderInfo;
```

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	MaxUnitPrice	AvgUnitPrice
1	苹果	手机	9999	100	20	10999	5099
2	苹果	耳机	599	500	50	10999	5099
3	苹果	平板	5999	300	100	10999	5099
4	苹果	笔记本	7999	800	100	10999	5099
5	华为	手机	10999	200	100	10999	5099
6	华为	耳机	399	700	200	10999	5099
7	华为	平板	3999	500	100	10999	5099
8	华为	笔记本	4999	600	300	10999	5099
9	小米	手机	7999	300	220	10999	5099
10	小米	耳机	299	700	150	10999	5099
11	小米	平板	3999	400	50	10999	5099
12	小米	笔记本	5999	700	450	10999	5099
13	荣耀	手机	7999	300	120	10999	5099
14	荣耀	耳机	299	200	100	10999	5099
15	荣耀	平板	3999	300	30	10999	5099
16	荣耀	笔记本	5999	500	220	10999	5099

窗口函数简介

```
SELECT *, MAX(UnitPrice) OVER(PARTITION BY ProductName) AS MaxUnitPrice FROM OrderInfo;

SELECT *, AVG(UnitPrice) OVER(PARTITION BY ProductName) AS AvgUnitPrice FROM OrderInfo;

SELECT *, MAX(UnitPrice) OVER(PARTITION BY ProductName) AS MaxUnitPrice,
                    AVG(UnitPrice) OVER(PARTITION BY ProductName) AS AvgUnitPrice FROM
OrderInfo;
```

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	MaxUnitPrice	AvgUnitPrice
3	苹果	平板	5999	300	100	5999	4499
7	华为	平板	3999	500	100	5999	4499
11	小米	平板	3999	400	50	5999	4499
15	荣耀	平板	3999	300	30	5999	4499
1	苹果	手机	9999	100	20	10999	9249
5	华为	手机	10999	200	100	10999	9249
9	小米	手机	7999	300	220	10999	9249
13	荣耀	手机	7999	300	120	10999	9249
4	苹果	笔记本	7999	800	100	7999	6249
8	华为	笔记本	4999	600	300	7999	6249
12	小米	笔记本	5999	700	450	7999	6249
16	荣耀	笔记本	5999	500	220	7999	6249
2	苹果	耳机	599	500	50	599	399
6	华为	耳机	399	700	200	599	399
10	小米	耳机	299	700	150	599	399
14	荣耀	耳机	299	200	100	599	399

ROW_NUMBER函数

```
SELECT *, ROW_NUMBER() OVER() AS RN FROM OrderInfo;

SELECT *, ROW_NUMBER() OVER(PARTITION BY ProductName) AS RN FROM OrderInfo;

SELECT *, ROW_NUMBER() OVER(PARTITION BY ProductName ORDER BY UnitPrice DESC) AS RN
FROM OrderInfo;


-- 查询价格排名最高的商品信息
SELECT * FROM (
        SELECT *, ROW_NUMBER() OVER(PARTITION BY ProductName ORDER BY UnitPrice DESC) AS RN
FROM OrderInfo
) AS Temp
WHERE Temp.RN = 1;
```

ROW_NUMBER、RANK与DENSE_RANK函数

```
SELECT
        *,
        ROW_NUMBER() OVER(PARTITION BY ProductName ORDER BY UnitsOnOrder DESC) AS RN,
        RANK() OVER(PARTITION BY ProductName ORDER BY UnitsOnOrder DESC) AS Rnk,
        DENSE_RANK() OVER(PARTITION BY ProductName ORDER BY UnitsOnOrder DESC) AS DR
FROM OrderInfo
WHERE ProductName = '平板';
```

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	RN	Rnk	DR
3	苹果	平板	5999	300	100	1	1	1
7	华为	平板	3999	500	100	2	1	1
11	小米	平板	3999	400	50	3	3	2
15	荣耀	平板	3999	300	30	4	4	3

窗口命名之WINDOW子句

```
SELECT
        *,
        ROW_NUMBER() OVER w AS RN,
        RANK() OVER w AS Rnk,
        DENSE_RANK() OVER w AS DR
FROM OrderInfo
WHERE ProductName = '平板'
WINDOW w AS (PARTITION BY ProductName ORDER BY UnitsOnOrder DESC);
```

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	RN	Rnk	DR
3	苹果	平板	5999	300	100	1	1	1
7	华为	平板	3999	500	100	2	1	1
11	小米	平板	3999	400	50	3	3	2
15	荣耀	平板	3999	300	30	4	4	3

FRAME子句设置

```
SELECT *, SUM(UnitsOnOrder) OVER(
            PARTITION BY ProductName ORDER BY UnitsOnOrder ) AS SUM
FROM OrderInfo;
```

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
15	荣耀	平板	3999	300	30	280
11	小米	平板	3999	400	50	280
3	苹果	平板	5999	300	100	280
7	华为	平板	3999	500	100	280
1	苹果	手机	9999	100	20	460
5	华为	手机	10999	200	100	460
13	荣耀	手机	7999	300	120	460
9	小米	手机	7999	300	220	460
4	苹果	笔记本	7999	800	100	1070
16	荣耀	笔记本	5999	500	220	1070
8	华为	笔记本	4999	600	300	1070
12	小米	笔记本	5999	700	450	1070
2	苹果	耳机	599	500	50	500
14	荣耀	耳机	299	200	100	500
10	小米	耳机	299	700	150	500
6	华为	耳机	399	700	200	500

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
15	荣耀	平板	3999	300	30	30
11	小米	平板	3999	400	50	80
3	苹果	平板	5999	300	100	280
7	华为	平板	3999	500	100	280
1	苹果	手机	9999	100	20	20
5	华为	手机	10999	200	100	120
13	荣耀	手机	7999	300	120	240
9	小米	手机	7999	300	220	460
4	苹果	笔记本	7999	800	100	100
16	荣耀	笔记本	5999	500	220	320
8	华为	笔记本	4999	600	300	620
12	小米	笔记本	5999	700	450	1070
2	苹果	耳机	599	500	50	50
14	荣耀	耳机	299	200	100	150
10	小米	耳机	299	700	150	300
6	华为	耳机	399	700	200	500

FRAME子句设置

```
SELECT *, SUM(UnitsOnOrder) OVER(
        PARTITION BY ProductName ORDER BY UnitsOnOrder
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;
```

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
15	荣耀	平板	3999	300	30	280
11	小米	平板	3999	400	50	280
3	苹果	平板	5999	300	100	280
7	华为	平板	3999	500	100	280
1	苹果	手机	9999	100	20	460
5	华为	手机	10999	200	100	460
13	荣耀	手机	7999	300	120	460
9	小米	手机	7999	300	220	460
4	苹果	笔记本	7999	800	100	1070
16	荣耀	笔记本	5999	500	220	1070
8	华为	笔记本	4999	600	300	1070
12	小米	笔记本	5999	700	450	1070
2	苹果	耳机	599	500	50	500
14	荣耀	耳机	299	200	100	500
10	小米	耳机	299	700	150	500
6	华为	耳机	399	700	200	500

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
15	荣耀	平板	3999	300	30	30
11	小米	平板	3999	400	50	80
3	苹果	平板	5999	300	100	280
7	华为	平板	3999	500	100	280
1	苹果	手机	9999	100	20	20
5	华为	手机	10999	200	100	120
13	荣耀	手机	7999	300	120	240
9	小米	手机	7999	300	220	460
4	苹果	笔记本	7999	800	100	100
16	荣耀	笔记本	5999	500	220	320
8	华为	笔记本	4999	600	300	620
12	小米	笔记本	5999	700	450	1070
2	苹果	耳机	599	500	50	50
14	荣耀	耳机	299	200	100	150
10	小米	耳机	299	700	150	300
6	华为	耳机	399	700	200	500

FRAME子句设置

Frame单位（默认为RANGE）

1. RANGE（无排序）：各个分区为一个单位
2. RANGE（有排序）：非连续值为一个单位，连续相同值为一个单位

FRAME子句设置

```
-- 所有行为一个分区，整个分区为一个单位
SELECT *, SUM(UnitsOnOrder) OVER() AS SUM FROM OrderInfo;

-- CURRENT ROW：当前单位的末尾行
SELECT *, SUM(UnitsOnOrder) OVER(
          RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;
```

FRAME子句设置

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
1	苹果	手机	9999	100	20	2310
2	苹果	耳机	599	500	50	2310
3	苹果	平板	5999	300	100	2310
4	苹果	笔记本	7999	800	100	2310
5	华为	手机	10999	200	100	2310
6	华为	耳机	399	700	200	2310
7	华为	平板	3999	500	100	2310
8	华为	笔记本	4999	600	300	2310
9	小米	手机	7999	300	220	2310
10	小米	耳机	299	700	150	2310
11	小米	平板	3999	400	50	2310
12	小米	笔记本	5999	700	450	2310
13	荣耀	手机	7999	300	120	2310
14	荣耀	耳机	299	200	100	2310
15	荣耀	平板	3999	300	30	2310
16	荣耀	笔记本	5999	500	220	2310

CURRENT ROW

FRAME子句设置

```
-- 划分分区，各个分区为一个单位
SELECT *, SUM(UnitsOnOrder) OVER(PARTITION BY ProductName) AS SUM FROM OrderInfo;


SELECT *, SUM(UnitsOnOrder) OVER(
          PARTITION BY ProductName
          RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;
```

FRAME子句设置

CURRENT ROW

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
3	苹果	平板	5999	300	100	280
7	华为	平板	3999	500	100	280
11	小米	平板	3999	400	50	280
15	荣耀	平板	3999	300	30	280
1	苹果	手机	9999	100	20	460
5	华为	手机	10999	200	100	460
9	小米	手机	7999	300	220	460
13	荣耀	手机	7999	300	120	460
4	苹果	笔记本	7999	800	100	1070
8	华为	笔记本	4999	600	300	1070
12	小米	笔记本	5999	700	450	1070
16	荣耀	笔记本	5999	500	220	1070
2	苹果	耳机	599	500	50	500
6	华为	耳机	399	700	200	500
10	小米	耳机	299	700	150	500
14	荣耀	耳机	299	200	100	500

FRAME子句设置

```
-- 所有行为一个分区，非连续行为一个单位，连续相同行为一个单位
SELECT *, SUM(UnitsOnOrder) OVER(ORDER BY UnitsOnOrder) AS SUM FROM OrderInfo;


SELECT *, SUM(UnitsOnOrder) OVER(
          ORDER BY UnitsOnOrder
          RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;
```

FRAME子句设置

CURRENT ROW

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
1	苹果	手机	9999	100	20	20
15	荣耀	平板	3999	300	30	50
2	苹果	耳机	599	500	50	150
11	小米	平板	3999	400	50	150
3	苹果	平板	5999	300	100	650
4	苹果	笔记本	7999	800	100	650
5	华为	手机	10999	200	100	650
7	华为	平板	3999	500	100	650
14	荣耀	耳机	299	200	100	650
13	荣耀	手机	7999	300	120	770
10	小米	耳机	299	700	150	920
6	华为	耳机	399	700	200	1120
9	小米	手机	7999	300	220	1560
16	荣耀	笔记本	5999	500	220	1560
8	华为	笔记本	4999	600	300	1860
12	小米	笔记本	5999	700	450	2310

FRAME子句设置

```
-- 划分分区，非连续值为一个单位，连续相同值为一个单位
SELECT *, SUM(UnitsOnOrder) OVER(PARTITION BY ProductName ORDER BY UnitsOnOrder) AS SUM
FROM OrderInfo;


SELECT *, SUM(UnitsOnOrder) OVER(
          PARTITION BY ProductName ORDER BY UnitsOnOrder
          RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;
```

FRAME子句设置

CURRENT ROW

OrderID	ProductBrand	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder	SUM
15	荣耀	平板	3999	300	30	30
11	小米	平板	3999	400	50	80
3	苹果	平板	5999	300	100	280
7	华为	平板	3999	500	100	280
1	苹果	手机	9999	100	20	20
5	华为	手机	10999	200	100	120
13	荣耀	手机	7999	300	120	240
9	小米	手机	7999	300	220	460
4	苹果	笔记本	7999	800	100	100
16	荣耀	笔记本	5999	500	220	320
8	华为	笔记本	4999	600	300	620
12	小米	笔记本	5999	700	450	1070
2	苹果	耳机	599	500	50	50
14	荣耀	耳机	299	200	100	150
10	小米	耳机	299	700	150	300
6	华为	耳机	399	700	200	500

FRAME子句设置

```
-- ROWS：一行为一个单位
SELECT *, SUM(UnitsOnOrder) OVER(
          PARTITION BY ProductName ORDER BY UnitsOnOrder
          ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;
```

FRAME子句设置

-- CURRENT ROW 当前单位的末尾行

-- UNBOUNDED PRECEDING 当前单位的末尾行上方所有行

-- UNBOUNDED FOLLOWING 当前单位的末尾行下方所有行

-- n PRECEDING 当前单位的末尾行上方n行

-- n FOLLOWING 当前单位的末尾行下方n行

FRAME子句设置

```
SELECT *, SUM(UnitsOnOrder) OVER(
          PARTITION BY ProductName ORDER BY UnitsOnOrder) AS SUM
FROM OrderInfo;


SELECT *, SUM(UnitsOnOrder) OVER(
          PARTITION BY ProductName ORDER BY UnitsOnOrder
          RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS SUM
FROM OrderInfo;


SELECT *, SUM(UnitsOnOrder) OVER(
          PARTITION BY ProductName ORDER BY UnitsOnOrder
          RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS SUM
FROM OrderInfo;
```

LAG、LEAD函数

```
SELECT
        *,
        LAG(UnitsOnOrder) OVER w AS Lg,
        Lead(UnitsOnOrder) OVER w AS Ld
FROM OrderInfo
WINDOW w AS (PARTITION BY ProductName ORDER BY UnitsOnOrder DESC);
```

FIRST_VALUE、LAST_VALUE、 NTH_VALUE函数

```
SELECT
        *,
        FIRST_VALUE(ProductBrand) OVER w AS BestSellerBrand,
        LAST_VALUE(ProductBrand) OVER w AS WorstSellerBrand,
        NTH_VALUE(ProductBrand, 2) OVER w AS SecondSellerBrand
FROM OrderInfo
WINDOW w AS (PARTITION BY ProductName ORDER BY UnitPrice DESC
                RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING);
```

NTILE函数

```
SELECT
        *, UnitPrice * UnitsOnOrder,
        NTILE(3) OVER (ORDER BY UnitPrice * UnitsOnOrder DESC) AS Nt
FROM OrderInfo;
```

CUME_DIST、PERCENT_RANK函数

```
SELECT
        *,
        CUME_DIST() OVER (PARTITION BY ProductName ORDER BY UnitPrice * UnitsOnOrder DESC) AS Cd
FROM OrderInfo;

SELECT
        *,
        PERCENT_RANK() OVER (PARTITION BY ProductName ORDER BY UnitPrice * UnitsOnOrder DESC) AS Cd
FROM OrderInfo;
```

视图

视图，是基于SELECT创建的虚拟表，不存储任何数据，但可动态查询最新数据。

- 创建视图

```
CREATE VIEW 视图名称 AS
```

- 查看视图

```
SHOW TABLES;
```

- 修改查询

```
CREATE OR REPLACE VIEW 视图名称 AS
```

- 删除查询

```
DROP VIEW 视图名称;
```

索引

- 数据量大，高频查询，使用索引，可加快查询速度。
- 数据量小，没有必要使用索引。
- 构建索引，会占用磁盘空间，会降低增删改的速度。
- 主键，会自动被设置索引。

➢ 创建索引

```
CREATE INDEX 索引名 ON 表名(字段名);
```

➢ 显示索引信息

```
SHOW INDEX FROM 表名;
```

➢ 删除索引

```
DROP INDEX 索引名 ON 表名;
```

事务

新增操作

删除操作

更新操作

事务

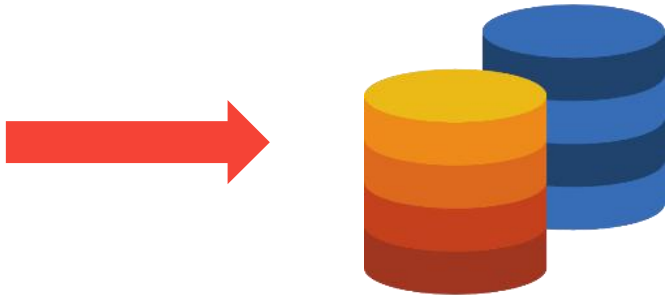
开启事务

新增操作

删除操作

更新操作

提交 / 回滚



- 设置提交方式

```
SET AUTOCOMMIT = 0;
```

- 开启事务

```
START TRANSACTION;
```

- 提交

```
COMMIT;
```

- 回滚

```
ROLLBACK;
```

- 设置保存点

```
SAVEPOINT 保存点名称
```

数据库设计规范化

目的：解决表数据冗余问题。

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

数据冗余

- 存储空间浪费
- 插入异常
- 删除异常
- 更新异常

插入异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

插入异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市
EMP-00005	孙七	女	29	**咨询顾问**	9000	2021-10-13 09:00:00	南京市

插入岗位名称错误！！！

删除异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

删除异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

顾问这个岗位，丢失了！！！

修改异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	开发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

修改异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	研发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

修改异常

员工表（Employees）							
EmployeeID	Name	Gender	Age	JobPosition	Salary	JoinedAt	Address
EMP-00001	张三	男	34	业务经理	20000	2000-01-15 15:00:00	上海市
EMP-00002	李四	女	28	研发人员	15000	2010-03-22 14:00:00	南京市
EMP-00003	王五	男	25	开发人员	10000	2015-07-28 13:00:00	北京市
EMP-00004	赵六	女	32	顾问	8000	2019-11-12 18:00:00	杭州市

部分数据未修改！！！

解决方案

Employees表

Employees表  JobPosition表

解决方案

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	开发人员
3	顾问

通过外键，构建表关联

插入问题解决

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	开发人员
3	顾问

插入问题解决

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3
EMP-00005	孙七	女	29	9000	2021-10-13 09:00:00	南京市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	开发人员
3	顾问

插入岗位名称不会出现错误。

删除问题解决

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	开发人员
3	顾问

删除问题解决

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	开发人员
3	顾问

岗位名称不会丢失。

修改问题解决

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	开发人员
3	顾问

修改问题解决

员工表（Employees）							
EmployeeID	Name	Gender	Age	Salary	JoinedAt	Address	JobPosition
EMP-00001	张三	男	34	20000	2000-01-15 15:00:00	上海市	1
EMP-00002	李四	女	28	15000	2010-03-22 14:00:00	南京市	2
EMP-00003	王五	男	25	10000	2015-07-28 13:00:00	北京市	2
EMP-00004	赵六	女	32	8000	2019-11-12 18:00:00	杭州市	3

岗位表（JobPosition）	
JobID	JobPosition
1	业务经理
2	研发人员
3	顾问

数据一改全改！！！

数据库规范化

- 第一范式（1NF）
- 第二范式（2NF）
- 第三范式（3NF）
- 巴斯范式（ BCNF ）
- 第四范式（4NF）
- 第五范式（5NF）

第一范式（1NF）

字段值：保持原子性（不可拆分）。

学生表（Student）				
StuID	Name	Gender	Age	Subject
1001	张三	男	20	MySQL
1002	李四	女	21	MySQL，SQL Server
1003	王五	男	22	Oracle
1004	赵六	女	20	Oracle，MySQL

违反1NF！！！

第一范式（1NF）

字段值：保持原子性（不可拆分）。

学生表（Student）				
StuID	Name	Gender	Age	Subject
1001	张三	男	20	MySQL
1002	李四	女	21	MySQL
1002	李四	女	21	SQL Server
1003	王五	男	22	Oracle
1004	赵六	女	20	Oracle
1004	赵六	女	20	MySQL

符合1NF

第二范式（2NF）

符合第一范式，并且不存在部分依赖。

什么是依赖？

依赖

主键

学生表（Student）

StuID	Name	Gender	Age	Subject
1001	张三	男	20	MySQL
1002	李四	女	21	MySQL
1003	王五	男	22	Oracle
1004	赵六	女	20	Oracle

什么是依赖？

依赖

学生表（Student）

复合主键

StuID	Name	Gender	Age	Subject
1001	张三	男	20	MySQL
1002	李四	女	21	MySQL
1003	王五	男	22	Oracle
1004	赵六	女	20	Oracle

```
CREATE TABLE Student (
        StuID INT AUTO_INCREMENT,
        Name VARCHAR ( 50 ) NOT NULL,
        Gender ENUM ( '男', '女' ),
        Age INT UNSIGNED,
        Subject VARCHAR ( 50 )
        PRIMARY KEY ( StuID, Name )
);
```

什么是部分依赖？

依赖

复合主键

成绩表（Score）			
StudentID	SubjectID	Marks	Teacher
1001	2001	100	张吉
1001	2002	95	谢彦
1002	2002	88	谢彦
1003	2003	97	傅智翔

数据冗余

部分依赖

学生表（Student）	
StudentID	SutdentName
1001	张三
1002	李四
1003	王五

学科表（Subject）	
SubjectID	SubjectName
2001	MySQL
2002	SQL Server
2003	Oracle

什么是部分依赖？

复合主键

成绩表（Score）

StudentID	SubjectID	Marks
1001	2001	100
1001	2002	95
1002	2002	88
1003	2003	97

学生表（Student）

StudentID	SutdentName
1001	张三
1002	李四
1003	王五

学科表（Subject）

SubjectID	SubjectName
2001	MySQL
2002	SQL Server
2003	Oracle

教师表（Teacher）

TeacherID	TeacherName	SubjectID
3001	张吉	2001
3002	谢彦	2002
3003	傅智翔	2003

第三范式（3NF）

符合第二范式，并且不存在传递依赖。

第三范式（3NF）

符合第二范式，并且不存在传递依赖。

成绩表（Score）		
StudentID	SubjectID	Marks
1001	2001	100
1001	2002	95
1002	2002	88
1003	2003	97

学生表（Student）	
StudentID	SutdentName
1001	张三
1002	李四
1003	王五

学科表（Subject）	
SubjectID	SubjectName
2001	MySQL
2002	SQL Server
2003	Oracle

教师表（Teacher）		
TeacherID	TeacherName	SubjectID
3001	张吉	2001
3002	谢彦	2002
3003	傅智翔	2003

第三范式（3NF）

符合第二范式，并且不存在传递依赖。

依赖　　依赖

复合主键

成绩表（Score）				
StudentID	SubjectID	Marks	ExamName	TotalMark
1001	2001	100	MySQL 试卷I	100
1001	2002	95	SQL Server 试卷I	120
1002	2002	88	SQL Server 试卷II	150
1003	2003	97	Oracle 试卷I	120

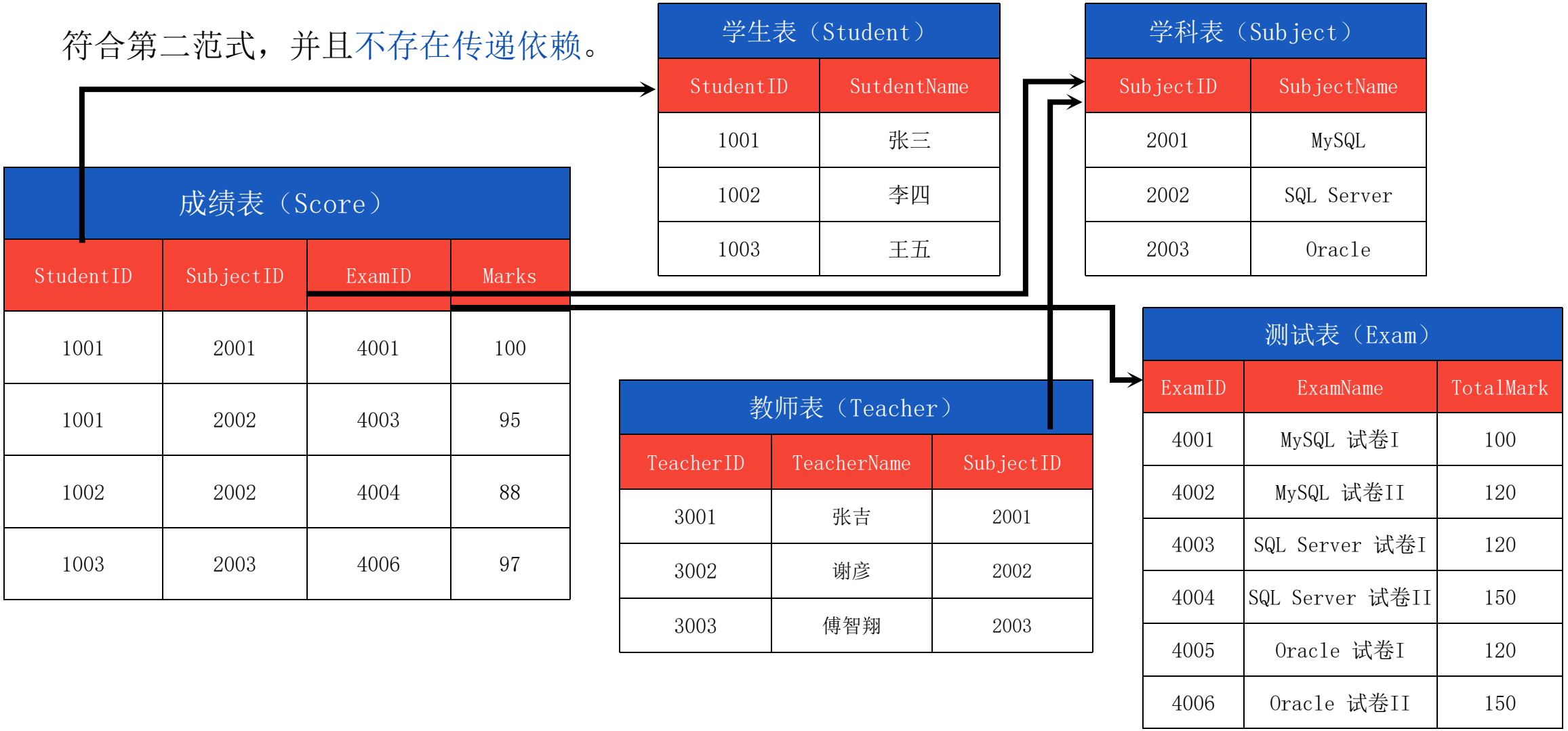
学生表（Student）	
StudentID	SutdentName
1001	张三
1002	李四
1003	王五

学科表（Subject）	
SubjectID	SubjectName
2001	MySQL
2002	SQL Server
2003	Oracle

教师表（Teacher）		
TeacherID	TeacherName	SubjectID
3001	张吉	2001
3002	谢彦	2002
3003	傅智翔	2003

第三范式（3NF）

符合第二范式，并且不存在传递依赖。



成绩表（Score）

StudentID	SubjectID	ExamID	Marks
1001	2001	4001	100
1001	2002	4003	95
1002	2002	4004	88
1003	2003	4006	97

学生表（Student）

StudentID	SutdentName
1001	张三
1002	李四
1003	王五

学科表（Subject）

SubjectID	SubjectName
2001	MySQL
2002	SQL Server
2003	Oracle

教师表（Teacher）

TeacherID	TeacherName	SubjectID
3001	张吉	2001
3002	谢彦	2002
3003	傅智翔	2003

测试表（Exam）

ExamID	ExamName	TotalMark
4001	MySQL 试卷I	100
4002	MySQL 试卷II	120
4003	SQL Server 试卷I	120
4004	SQL Server 试卷II	150
4005	Oracle 试卷I	120
4006	Oracle 试卷II	150

三范式总结

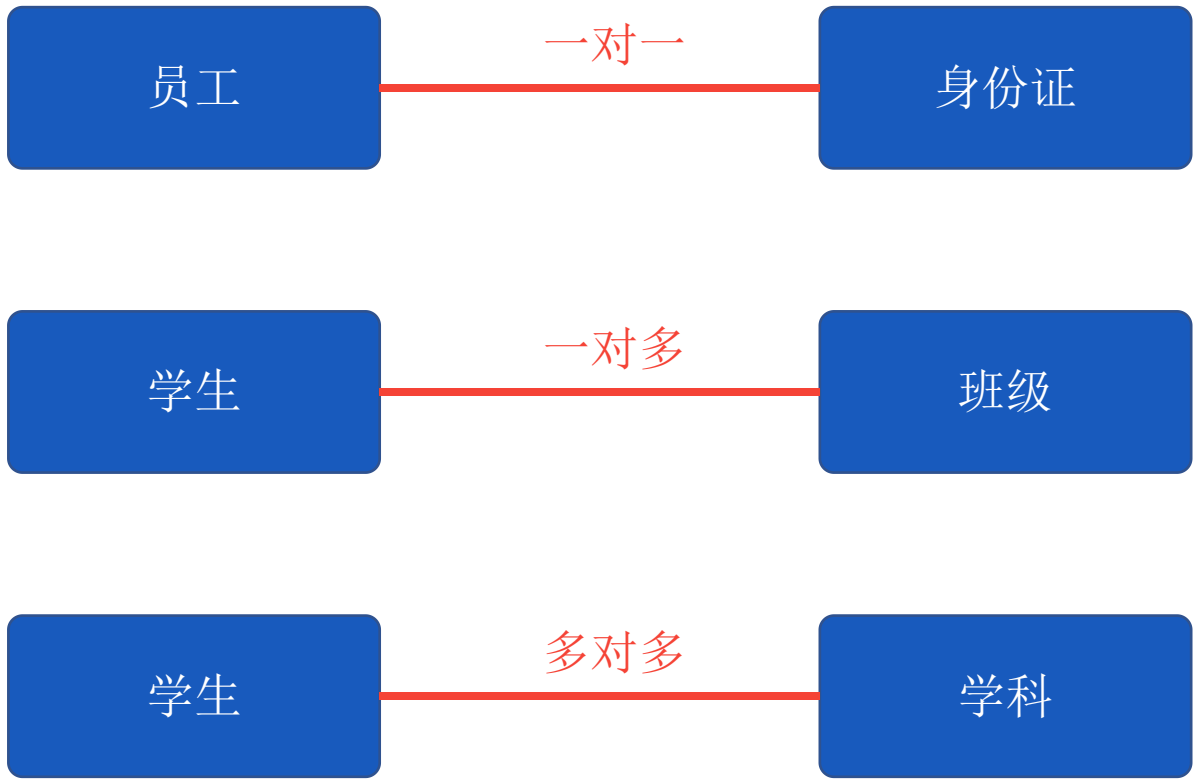
目的：解决多表数据冗余问题。

1. 拆字段

2. 拆表

3. 构建表关系

表关系

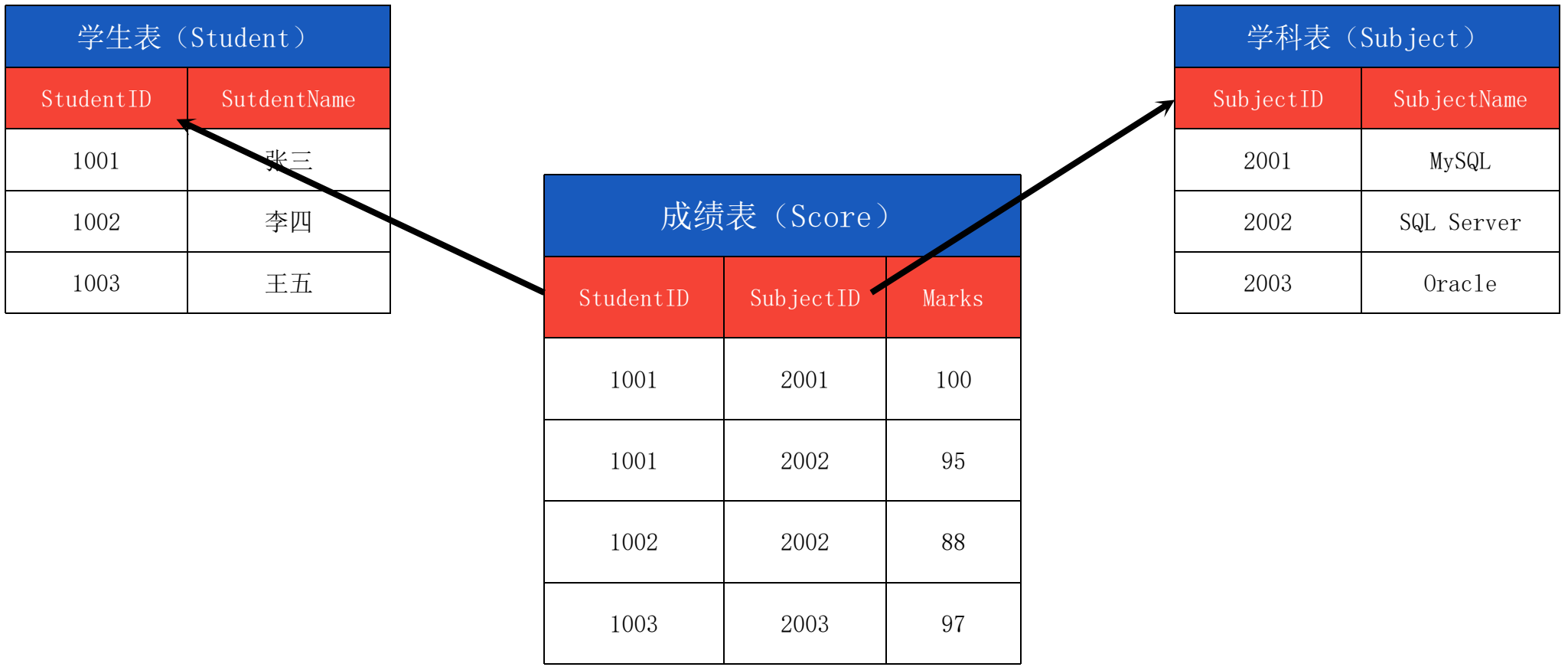


一对多

学生表（Student）		
StudentID	SutdentName	ClassID
1001	张三	5001
1002	李四	5001
1003	王五	5002
1004	赵六	5003
1005	孙七	5002

班级表（Class）	
ClassID	ClassName
5001	计算机一班
5002	计算机二班
5003	计算机三班

多对多


学生表（Student）
StudentID
SutdentName
1001
张三
1002
李四
1003
王五
成绩表（Score）
StudentID
SubjectID
Marks
1001
2001
100
1001
2002
95
1002
2002
88
1003
2003
97
学科表（Subject）
SubjectID
SubjectName
2001
MySQL
2002
SQL Server
2003
Oracle

数据库设计实操

学生表（Student）		
StudentID	SutdentName	ClassID
1001	张三	5001
1002	李四	5001
1003	王五	5002

学科表（Subject）	
SubjectID	SubjectName
2001	MySQL
2002	SQL Server
2003	Oracle

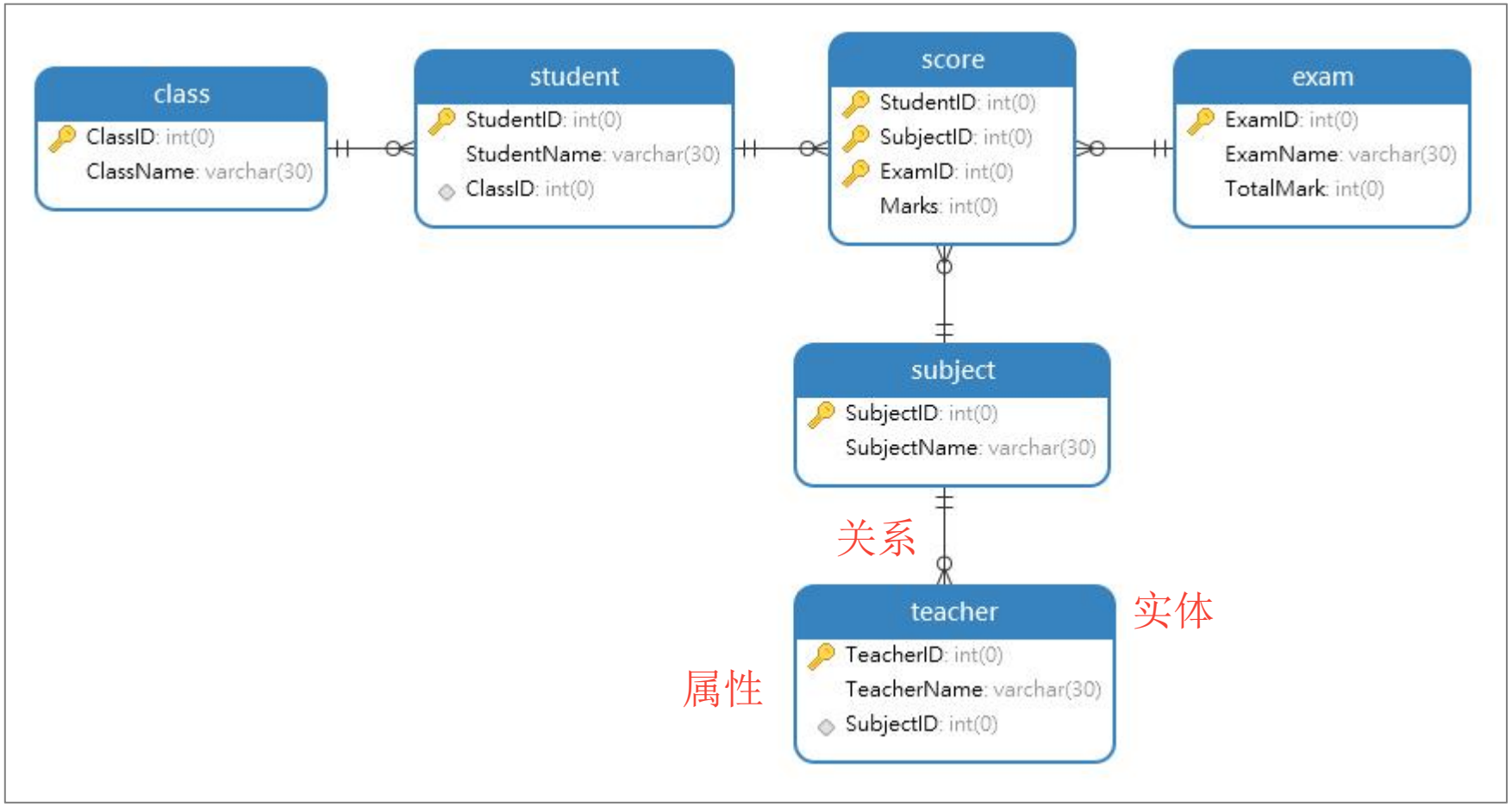
测试表（Exam）		
ExamID	ExamName	TotalMark
4001	MySQL 试卷I	100
4002	MySQL 试卷II	120
4003	SQL Server 试卷I	120
4004	SQL Server 试卷II	150
4005	Oracle 试卷I	120
4006	Oracle 试卷II	150

成绩表（Score）			
StudentID	SubjectID	ExamID	Marks
1001	2001	4001	100
1001	2002	4003	95
1002	2002	4004	88
1003	2003	4006	97

班级表（Class）	
ClassID	ClassName
5001	计算机一班
5002	计算机二班
5003	计算机三班

教师表（Teacher）		
TeacherID	TeacherName	SubjectID
3001	张吉	2001
3002	谢彦	2002
3003	傅智翔	2003

ER图（Entity Relationship Diagram）



EXCEL VS MySQL