Hive 第 3 章 综合实践工作手册

课程信息

- **实践时间**: 4小时
- **数据库名称**: unit3db
- **数据目录**: /usr/datas/unit3
- **MySQL信息**: 用户名: hiveuser, 密码: 123456
- **Beeline连接**: 使用 `hive-bee` 命令

目录

第一部分:环境准备与基础操作 (30 分钟)

1.1 数据库创建与管理

1.2 准备实践数据集

1.3 数据导入验证

第二部分:数据类型与表操作实践 (60 分钟)

2.1 复杂数据类型实践

2.2 内部表与外部表对比

2.3 分区表实践

第三部分:运算符与函数综合实践 (90 分钟)

3.1 算术与比较运算符

3.2 逻辑运算符应用

3.3 字符串函数实践

3.4 日期时间函数实践

3.5 条件函数实践

3.6 聚合函数应用

第四部分:查询与分析综合实践 (60 分钟)

4.1 复杂WHERE条件查询

4.2 GROUP BY与聚合分析

4.3 ORDER BY与排序

4.4 JOIN多表关联

4.5 子查询应用

第五部分:数据导入导出实践 (30 分钟)

5.1 多种数据导入方式

5.2 数据导出实践

5.3 综合案例

第一部分:环境准备与基础操作 (30 分钟)

1.1 数据库创建与管理

步骤1: 连接Beeline

```BASH
hive-bee
```

步骤2: 创建实践数据库

```SQL
-- 创建数据库
CREATE DATABASE IF NOT EXISTS unit3db
COMMENT '第3章综合实践数据库'
LOCATION '/user/hive/unit3db';

-- 查看数据库
SHOW DATABASES;

-- 使用数据库
USE unit3db;

-- 查看当前数据库
SELECT current_database();
```

验证: 确认数据库创建成功,当前使用的是unit3db

1.2 准备实践数据集

创建数据目录

```bash
# 在新终端窗口执行
sudo mkdir -p /usr/datas/unit3
sudo chmod 777 /usr/datas/unit3
cd /usr/datas/unit3
```

数据集1: 学生信息表 (students.txt)

```bash
cat > /usr/datas/unit3/students.txt << 'EOF'
1001    张三    男    20    计算机学院    2023-09-01    18612345601    beijing
1002    李四    女    19    数学学院    2023-09-01    18612345602    shanghai
1003    王五    男    21    物理学院    2022-09-01    18612345603    guangzhou
1004    赵六    女    20    化学学院    2023-09-01    18612345604    shenzhen
1005    钱七    男    22    计算机学院    2021-09-01    18612345605    hangzhou
1006    孙八    女    19    数学学院    2023-09-01    18612345606    nanjing
1007    周九    男    20    计算机学院    2023-09-01    18612345607    wuhan
1008    吴十    女    21    物理学院    2022-09-01    18612345608    chengdu
1009    郑一    男    19    计算机学院    2023-09-01    18612345609    xian
1010    王二    女    20    数学学院    2023-09-01    18612345610    tianjin
EOF
```

数据集2: 课程成绩表 (scores.txt)

```bash
cat > /usr/datas/unit3/scores.txt << 'EOF'
1001    C001    85    2024-01-15
1001    C002    90    2024-01-16
1001    C003    78    2024-01-17
1002    C001    92    2024-01-15
1002    C002    88    2024-01-16
1003    C001    76    2024-01-15
1003    C003    95    2024-01-17
1004    C002    89    2024-01-16
1005    C001    94    2024-01-15
1005    C002    87    2024-01-16
1005    C003    91    2024-01-17
1006    C001    88    2024-01-15
1007    C002    92    2024-01-16
1008    C003    85    2024-01-17
1009    C001    90    2024-01-15
1009    C002    86    2024-01-16
1010    C001    93    2024-01-15
1010    C003    89    2024-01-17
EOF
```

数据集3: 课程信息表 (courses.txt)

```bash
cat > /usr/datas/unit3/courses.txt << 'EOF'
C001    大数据技术    4    张教授
C002    数据库原理    3    李教授
C003    Python编程    4    王教授
C004    机器学习    3    赵教授
C005    数据结构    4    钱教授
EOF
```

数据集4: 电商订单表 (orders.txt)

```bash
cat > /usr/datas/unit3/orders.txt << 'EOF'
O001    1001    2024-01-01 10:30:00    1580.50    已支付
O002    1002    2024-01-02 14:20:00    2399.00    已支付
O003    1003    2024-01-02 16:45:00    599.00    已发货
O004    1001    2024-01-03 09:15:00    3200.00    已支付
O005    1004    2024-01-03 11:30:00    899.00    待支付
O006    1002    2024-01-04 15:20:00    1299.00    已支付
O007    1005    2024-01-05 10:10:00    4599.00    已发货
O008    1003    2024-01-05 13:30:00    799.00    已完成
O009    1006    2024-01-06 16:00:00    1899.00    已支付
O010    1007    2024-01-07 09:45:00    2599.00    已发货
EOF
```

数据集5: 订单明细表 (order_items.txt)

```bash
cat > /usr/datas/unit3/order_items.txt << 'EOF'
O001    P001    笔记本电脑    1    4999.00
O001    P002    鼠标    2    99.00
O002    P003    手机    1    2399.00
O003    P004    键盘    1    299.00
O003    P005    耳机    1    199.00
O004    P001    笔记本电脑    1    4999.00
O004    P006    显示器    1    1599.00
O005    P007    平板电脑    1    2899.00
O006    P003    手机    1    2399.00
O007    P008    相机    1    4599.00
O008    P009    音箱    1    799.00
O009    P010    路由器    2    299.00
O010    P003    手机    1    2599.00
EOF
```

数据集6: 员工信息表 (employees.txt)

```bash
cat > /usr/datas/unit3/employees.txt << 'EOF'
E001    张伟    男    28    研发部    15000    2020-03-15    北京
E002    李娜    女    26    市场部    12000    2021-06-01    上海
E003    王强    男    32    研发部    25000    2018-01-10    深圳
E004    刘芳    女    29    人事部    13000    2019-09-20    广州
E005    陈杰    男    35    研发部    28000    2017-05-15    北京
E006    赵敏    女    27    财务部    14000    2020-11-01    杭州
E007    孙涛    男    30    市场部    16000    2019-03-25    上海
E008    周丽    女    25    人事部    11000    2022-01-10    南京
E009    吴刚    男    33    研发部    26000    2018-07-20    深圳
E010    郑美    女    28    市场部    15000    2020-08-15    武汉
EOF
```

数据集7: 网站访问日志 (weblogs.txt)

```bash
cat > /usr/datas/unit3/weblogs.txt << 'EOF'
192.168.1.100    2024-01-15 08:30:45    /index.html    GET    200    1523    Chrome    Windows
192.168.1.101    2024-01-15 08:31:20    /products.html    GET    200    3456    Firefox    MacOS
192.168.1.102    2024-01-15 08:32:10    /login    POST    302    456    Safari    iOS
192.168.1.100    2024-01-15 08:35:30    /cart    GET    200    2345    Chrome    Windows
192.168.1.103    2024-01-15 08:40:15    /index.html    GET    200    1523    Edge    Windows
192.168.1.101    2024-01-15 08:45:20    /checkout    POST    200    678    Firefox    MacOS
192.168.1.104    2024-01-15 09:00:00    /products.html    GET    404    234    Chrome    Android
192.168.1.102    2024-01-15 09:10:30    /order    POST    200    890    Safari    iOS
192.168.1.105    2024-01-15 09:20:45    /index.html    GET    200    1523    Chrome    Linux
192.168.1.100    2024-01-15 09:30:00    /profile    GET    200    2100    Chrome    Windows
EOF
```

1.3 数据导入验证

回到Beeline创建基础表

```sql
USE unit3db;
```

```
-- 创建学生表
CREATE TABLE IF NOT EXISTS students (
    stu_id INT COMMENT '学生ID',
    stu_name STRING COMMENT '学生姓名',
    gender STRING COMMENT '性别',
    age INT COMMENT '年龄',
    college STRING COMMENT '学院',
    enroll_date DATE COMMENT '入学日期',
    phone STRING COMMENT '电话',
    city STRING COMMENT '城市'
)
COMMENT '学生信息表'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

-- 加载数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/students.txt'
OVERWRITE INTO TABLE students;

-- 验证数据
SELECT * FROM students LIMIT 5;
SELECT COUNT(*) as total_students FROM students;
```

期望结果: 应该看到10条学生记录

第二部分:数据类型与表操作实践 (60 分钟)

2.1 复杂数据类型实践

实践目标: 掌握ARRAY、MAP、STRUCT等复杂数据类型的使用

步骤1: 创建包含复杂类型的表

```SQL
-- 创建学生详细信息表(包含复杂类型)
CREATE TABLE student_detail (
    stu_id INT COMMENT '学生ID',
    stu_name STRING COMMENT '姓名',
    hobbies ARRAY<STRING> COMMENT '爱好列表',
    scores MAP<STRING, INT> COMMENT '各科成绩',
    address STRUCT<province:STRING, city:STRING, district:STRING> COMMENT '地址信息',
    contact_info MAP<STRING, STRING> COMMENT '联系方式'
)
```

```
COMMENT '学生详细信息表'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;
```

步骤2: 准备复杂类型数据

```bash
cat > /usr/datas/unit3/student_detail.txt << 'EOF'
1001|张三|篮球,编程,阅读|数学:85,英语:90,物理:78|北京,海淀区,中关村
|email:zhangsan@example.com,qq:123456789
1002|李四|游泳,音乐|数学:92,英语:88,化学:85|上海,浦东新区,陆家嘴
|email:lisi@example.com,wechat:lisi123
1003|王五|足球,旅游,摄影|数学:76,物理:95,生物:80|广州,天河区,珠江新城
|email:wangwu@example.com,qq:987654321
1004|赵六|绘画,舞蹈|英语:89,化学:91,生物:87|深圳,南山区,科技园
|email:zhaoliu@example.com,phone:13800138000
1005|钱七|编程,电竞|数学:94,英语:87,物理:91|杭州,西湖区,文三路
|email:qianqi@example.com,wechat:qianqi456
EOF
```

步骤3: 加载并查询复杂类型数据

```sql
-- 加载数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/student_detail.txt'
OVERWRITE INTO TABLE student_detail;


-- 查询ARRAY类型
SELECT stu_name, hobbies FROM student_detail;


-- 访问ARRAY元素(第一个爱好)
SELECT stu_name, hobbies[0] as first_hobby FROM student_detail;


-- 查询ARRAY大小
SELECT stu_name, SIZE(hobbies) as hobby_count FROM student_detail;


-- 查询MAP类型
SELECT stu_name, scores FROM student_detail;


-- 访问MAP元素(数学成绩)
SELECT stu_name, scores['数学'] as math_score FROM student_detail;
```

```
-- 获取MAP的所有键
SELECT stu_name, MAP_KEYS(scores) as subjects FROM student_detail;


-- 获取MAP的所有值
SELECT stu_name, MAP_VALUES(scores) as all_scores FROM student_detail;


-- 查询STRUCT类型
SELECT stu_name, address FROM student_detail;


-- 访问STRUCT字段
SELECT stu_name,
       address.province as province,
       address.city as city,
       address.district as district
FROM student_detail;


-- 复杂查询:找出数学成绩大于85且爱好中包含编程的学生
SELECT stu_name, scores['数学'] as math_score, hobbies
FROM student_detail
WHERE scores['数学'] > 85 AND ARRAY_CONTAINS(hobbies, '编程');
```

练习题:

查询每个学生的爱好数量,并按数量降序排列

查询所有学生的平均数学成绩

查询地址在北京的学生姓名和联系邮箱

2.2 内部表与外部表对比

实践目标: 理解内部表和外部表的区别

步骤1: 创建内部表

```
SQL
-- 创建内部表(已创建)
-- students表就是内部表


-- 查看表信息
DESC FORMATTED students;
```

步骤2: 创建外部表

```
SQL
-- 创建外部表
```

```
CREATE EXTERNAL TABLE students_external (
    stu_id INT COMMENT '学生ID',
    stu_name STRING COMMENT '学生姓名',
    gender STRING COMMENT '性别',
    age INT COMMENT '年龄',
    college STRING COMMENT '学院',
    enroll_date DATE COMMENT '入学日期',
    phone STRING COMMENT '电话',
    city STRING COMMENT '城市'
)
COMMENT '学生信息外部表'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/external/students';


-- 查看表信息
DESC FORMATTED students_external;
```

步骤3: 对比测试

```
SQL
-- 在Linux终端创建外部表数据目录
-- hadoop fs -mkdir -p /user/hive/external/students
-- hadoop fs -put /usr/datas/unit3/students.txt /user/hive/external/students/


-- 查询外部表数据
SELECT * FROM students_external LIMIT 5;


-- 删除内部表(数据也会被删除)
DROP TABLE IF EXISTS students_temp_internal;
CREATE TABLE students_temp_internal AS SELECT * FROM students WHERE age > 20;
SELECT COUNT(*) FROM students_temp_internal;
DROP TABLE students_temp_internal;


-- 删除外部表(数据不会被删除)
-- 测试略过,避免误删除
```

知识点总结:

- 内部表(MANAGED TABLE): DROP TABLE时,元数据和数据都会被删除
- 外部表(EXTERNAL TABLE): DROP TABLE时,只删除元数据,数据文件保留

2.3 分区表实践

实践目标: 掌握分区表的创建和使用

步骤1: 创建分区表

```sql
-- 创建按日期分区的订单表
└
```

步骤2: 准备分区数据

```bash
# 创建不同日期的订单数据
cat > /usr/datas/unit3/orders_20240101.txt << 'EOF'
O001    1001    2024-01-01 10:30:00    1580.50    已支付
O002    1002    2024-01-01 14:20:00    2399.00    已支付
EOF

cat > /usr/datas/unit3/orders_20240102.txt << 'EOF'
O003    1003    2024-01-02 16:45:00    599.00    已发货
O004    1001    2024-01-02 09:15:00    3200.00    已支付
EOF

cat > /usr/datas/unit3/orders_20240103.txt << 'EOF'
O005    1004    2024-01-03 11:30:00    899.00    待支付
O006    1002    2024-01-03 15:20:00    1299.00    已支付
EOF
```

步骤3: 加载分区数据

```sql
-- 加载2024-01-01分区数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/orders_20240101.txt'
INTO TABLE orders_partitioned
PARTITION (order_date='2024-01-01');

-- 加载2024-01-02分区数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/orders_20240102.txt'
INTO TABLE orders_partitioned
PARTITION (order_date='2024-01-02');

-- 加载2024-01-03分区数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/orders_20240103.txt'
INTO TABLE orders_partitioned
PARTITION (order_date='2024-01-03');
```

```
-- 查看所有分区
SHOW PARTITIONS orders_partitioned;


-- 查询特定分区数据
SELECT * FROM orders_partitioned WHERE order_date = '2024-01-01';


-- 查询多个分区数据
SELECT order_date, COUNT(*) as order_count, SUM(amount) as total_amount
FROM orders_partitioned
WHERE order_date IN ('2024-01-01', '2024-01-02')
GROUP BY order_date;


-- 查询所有分区数据
SELECT * FROM orders_partitioned;
```

步骤4: 动态分区插入

```
SQL
-- 启用动态分区
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;


-- 创建临时订单表
CREATE TABLE orders_temp (
    order_id STRING,
    customer_id INT,
    order_date STRING,
    order_time TIMESTAMP,
    amount DECIMAL(10,2),
    status STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';


-- 插入测试数据
LOAD DATA LOCAL
INPATH '/usr/datas/unit3/orders_temp.txt' OVERWRITE INTO TABLE orders_temp;
SELECT * FROM orders_temp;




-- 验证动态分区
SHOW PARTITIONS orders_partitioned;
SELECT * FROM orders_partitioned WHERE order_date IN ('2024-01-04', '2024-01-05');
```

练习题:

统计每个分区的订单总金额

查询金额超过2000的订单所在的分区

删除order_date='2024-01-05'的分区

第三部分:运算符与函数综合实践 (90 分钟)

3.1 算术与比较运算符

创建基础表

```sql
-- 创建成绩表
CREATE TABLE scores (
    stu_id INT COMMENT '学生ID',
    course_id STRING COMMENT '课程ID',
    score INT COMMENT '成绩',
    exam_date DATE COMMENT '考试日期'
)
COMMENT '课程成绩表'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;


-- 加载数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/scores.txt'
OVERWRITE INTO TABLE scores;


-- 验证数据
SELECT * FROM scores LIMIT 5;
```

算术运算实践

```sql
-- 基本算术运算
SELECT
    stu_id,
    course_id,
    score,
    score + 5 as bonus_score,        -- 加分
    score - 5 as penalty_score,      -- 减分
    score * 1.1 as scaled_score,     -- 按比例加分
    score / 10 as score_level,       -- 分数等级
    score % 10 as score_remainder    -- 取余
FROM scores
LIMIT 5;
```

```
-- 成绩统计
SELECT
    stu_id,
    COUNT(*) as course_count,              -- 课程数量
    SUM(score) as total_score,             -- 总分
    AVG(score) as avg_score,               -- 平均分
    MAX(score) as max_score,               -- 最高分
    MIN(score) as min_score,               -- 最低分
    MAX(score) - MIN(score) as score_diff  -- 分数差距
FROM scores
GROUP BY stu_id
ORDER BY avg_score DESC;


-- 计算及格率
SELECT
    course_id,
    COUNT(*) as total_students,
    SUM(CASE WHEN score >= 60 THEN 1 ELSE 0 END) as pass_count,
    SUM(CASE WHEN score >= 60 THEN 1 ELSE 0 END) / COUNT(*) as pass_rate,
    ROUND(SUM(CASE WHEN score >= 60 THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) as
pass_percentage
FROM scores
GROUP BY course_id;
```

比较运算实践

```
SQL
-- 基本比较运算
SELECT stu_id, course_id, score
FROM scores
WHERE score > 85;  -- 大于85分


SELECT stu_id, course_id, score
FROM scores
WHERE score >= 90;  -- 大于等于90分(优秀)


SELECT stu_id, course_id, score
FROM scores
WHERE score < 60;  -- 小于60分(不及格)


SELECT stu_id, course_id, score
FROM scores
WHERE score <= 80;  -- 小于等于80分


SELECT stu_id, course_id, score
FROM scores
WHERE score = 90;  -- 等于90分


SELECT stu_id, course_id, score
FROM scores
WHERE score != 85;  -- 不等于85分(也可以用<>)
```

```
SELECT stu_id, course_id, score
FROM scores
WHERE score <> 85;  -- 不等于85分

-- BETWEEN范围查询
SELECT stu_id, course_id, score
FROM scores
WHERE score BETWEEN 80 AND 90;  -- 80到90分之间

-- IN查询
SELECT stu_id, course_id, score
FROM scores
WHERE stu_id IN (1001, 1002, 1003);

-- NOT IN查询
SELECT stu_id, course_id, score
FROM scores
WHERE course_id NOT IN ('C001');

-- IS NULL查询
SELECT stu_id, course_id, score
FROM scores
WHERE score IS NULL;

-- IS NOT NULL查询
SELECT stu_id, course_id, score
FROM scores
WHERE score IS NOT NULL;
```

练习题:

查询平均分在85-95之间的学生ID

计算每门课程的最高分、最低分和平均分

查询有不及格科目的学生ID

3.2 逻辑运算符应用

```
SQL
-- AND运算符(与)
SELECT stu_id, course_id, score
FROM scores
WHERE score > 80 AND course_id = 'C001';

SELECT stu_id, course_id, score
FROM scores
WHERE score >= 85 AND score <= 95;
```

```
-- OR运算符(或)
SELECT stu_id, course_id, score
FROM scores
WHERE score > 90 OR course_id = 'C003';


SELECT stu_id, course_id, score
FROM scores
WHERE stu_id = 1001 OR stu_id = 1002;


-- NOT运算符(非)
SELECT stu_id, course_id, score
FROM scores
WHERE NOT score < 60;  -- 等同于score >= 60


SELECT stu_id, course_id, score
FROM scores
WHERE course_id NOT IN ('C001', 'C002');


-- 复合逻辑运算
SELECT stu_id, course_id, score
FROM scores
WHERE (score > 90 OR score < 70) AND course_id = 'C001';


SELECT stu_id, course_id, score
FROM scores
WHERE score BETWEEN 80 AND 90 AND course_id IN ('C001', 'C002');


-- 多条件组合查询
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    s.score,
    CASE
        WHEN s.score >= 90 THEN '优秀'
        WHEN s.score >= 80 AND s.score < 90 THEN '良好'
        WHEN s.score >= 70 AND s.score < 80 THEN '中等'
        WHEN s.score >= 60 AND s.score < 70 THEN '及格'
        ELSE '不及格'
    END as grade_level
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
WHERE (s.score >= 85 OR st.college = '计算机学院')
    AND s.exam_date >= '2024-01-15'
ORDER BY s.score DESC;
```

练习题:

查询分数在60-80之间或90以上的记录

查询计算机学院且成绩优秀(>=90)的学生

查询非C001课程且成绩不及格的记录

3.3 字符串函数实践

创建课程表

```SQL
-- 创建课程表
CREATE TABLE courses (
    course_id STRING COMMENT '课程ID',
    course_name STRING COMMENT '课程名称',
    credits INT COMMENT '学分',
    teacher STRING COMMENT '授课教师'
)
COMMENT '课程信息表'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;


-- 加载数据
LOAD DATA LOCAL INPATH '/usr/datas/unit3/courses.txt'
OVERWRITE INTO TABLE courses;
```

字符串函数综合应用

```SQL
-- LENGTH/CHAR_LENGTH: 字符串长度
SELECT
    stu_name,
    LENGTH(stu_name) as name_byte_length,
    CHAR_LENGTH(stu_name) as name_char_length
FROM students;


-- CONCAT: 字符串连接
SELECT
    CONCAT(stu_id, '-', stu_name) as stu_info,
    CONCAT(stu_name, '(', gender, ')') as name_with_gender,
    CONCAT(college, '-', stu_name) as college_stu
FROM students;


-- CONCAT_WS: 使用分隔符连接字符串
SELECT
    CONCAT_WS('-', stu_id, stu_name, college) as stu_full_info,
    CONCAT_WS('/', college, stu_name, phone) as contact_info
FROM students;


-- SUBSTR/SUBSTRING: 截取字符串
SELECT
```

```
    stu_name,
    SUBSTR(stu_name, 1, 1) as surname,              -- 姓
    SUBSTRING(phone, 1, 3) as phone_prefix,         -- 手机号前3位
    SUBSTRING(phone, -4) as phone_suffix            -- 手机号后4位
FROM students;


-- UPPER/LOWER: 大小写转换
SELECT
    course_id,
    UPPER(course_id) as upper_id,
    LOWER(course_id) as lower_id,
    city,
    UPPER(city) as upper_city
FROM students s
JOIN scores sc ON s.stu_id = sc.stu_id
LIMIT 5;


-- TRIM/LTRIM/RTRIM: 去除空格
SELECT
    TRIM('  Hello World  ') as trimmed,
    LTRIM('  Hello World  ') as left_trimmed,
    RTRIM('  Hello World  ') as right_trimmed;


-- LPAD/RPAD: 填充字符串
SELECT
    stu_id,
    LPAD(stu_id, 10, '0') as padded_id,             -- 左填充
    RPAD(stu_name, 10, '*') as padded_name          -- 右填充
FROM students
LIMIT 5;


-- INSTR: 查找子串位置
SELECT
    stu_name,
    INSTR(stu_name, '张') as position,
    INSTR(phone, '186') as has_186
FROM students;


-- LOCATE: 查找子串位置(同INSTR,参数顺序不同)
SELECT
    stu_name,
    LOCATE('张', stu_name) as position
FROM students;


-- REPLACE: 替换字符串
SELECT
    college,
    REPLACE(college, '学院', 'College') as college_en,
    phone,
    REPLACE(phone, '186', '138') as new_phone
FROM students;
```

```
-- SPLIT: 分割字符串
SELECT
    stu_name,
    SPLIT(phone, '123')[0] as phone_part1
FROM students
LIMIT 3;


-- REVERSE: 反转字符串
SELECT
    stu_name,
    REVERSE(stu_name) as reversed_name
FROM students
LIMIT 5;


-- REPEAT: 重复字符串
SELECT
    stu_name,
    REPEAT('*', 3) as stars,
    CONCAT(REPEAT('=', 10), stu_name, REPEAT('=', 10)) as decorated_name
FROM students
LIMIT 5;


-- 综合案例:格式化学生信息
SELECT
    CONCAT('[', LPAD(stu_id, 6, '0'), ']') as formatted_id,
    CONCAT(UPPER(SUBSTR(stu_name, 1, 1)), LOWER(SUBSTR(stu_name, 2))) as
formatted_name,
    CONCAT(SUBSTR(phone, 1, 3), '****', SUBSTR(phone, -4)) as masked_phone,
    REPLACE(college, '学院', '') as short_college
FROM students;
```

练习题:

提取所有学生的姓氏,并统计每个姓氏的人数

将所有课程名称转换为"[课程ID]-课程名称"格式

对手机号进行脱敏处理,只显示前3位和后4位

3.4 日期时间函数实践

```
SQL
-- 当前日期时间
SELECT
    CURRENT_DATE() as current_date,              -- 当前日期
    CURRENT_TIMESTAMP() as current_timestamp,    -- 当前时间戳
    NOW() as now_time,                           -- 当前时间
    UNIX_TIMESTAMP() as unix_ts;                 -- Unix时间戳
```

```
-- 日期提取
SELECT
    enroll_date,
    YEAR(enroll_date) as enroll_year,           -- 年
    MONTH(enroll_date) as enroll_month,         -- 月
    DAY(enroll_date) as enroll_day,             -- 日
    DAYOFMONTH(enroll_date) as day_of_month,    -- 月中的第几天
    DAYOFWEEK(enroll_date) as day_of_week,      -- 周中的第几天(1=周日)
    DAYOFYEAR(enroll_date) as day_of_year,      -- 年中的第几天
    WEEKOFYEAR(enroll_date) as week_of_year,    -- 年中的第几周
    QUARTER(enroll_date) as quarter             -- 季度
FROM students;

-- 日期格式化
SELECT
    enroll_date,
    DATE_FORMAT(enroll_date, 'yyyy-MM-dd') as formatted_date,
    DATE_FORMAT(enroll_date, 'yyyy年MM月dd日') as chinese_date,
    DATE_FORMAT(enroll_date, 'yyyy-MM') as year_month
FROM students;

-- 日期转换
SELECT
    '2024-01-15' as date_string,
    TO_DATE('2024-01-15 10:30:00') as to_date_result,
    CAST('2024-01-15' AS DATE) as cast_date,
    FROM_UNIXTIME(1705287000) as from_unix;

-- 日期计算
SELECT
    enroll_date,
    DATE_ADD(enroll_date, 7) as add_7days,          -- 加7天
    DATE_ADD(enroll_date, 1) as tomorrow,           -- 明天
    DATE_SUB(enroll_date, 7) as sub_7days,          -- 减7天
    ADD_MONTHS(enroll_date, 6) as add_6months,      -- 加6个月
    ADD_MONTHS(enroll_date, -3) as sub_3months      -- 减3个月
FROM students;

-- 日期差值计算
SELECT
    stu_name,
    enroll_date,
    CURRENT_DATE() as today,
    DATEDIFF(CURRENT_DATE(), enroll_date) as days_since_enroll,
    DATEDIFF(CURRENT_DATE(), enroll_date) / 365 as years_since_enroll,
    MONTHS_BETWEEN(CURRENT_DATE(), enroll_date) as months_since_enroll
FROM students;

-- 获取月份的最后一天
SELECT
    enroll_date,
    LAST_DAY(enroll_date) as last_day_of_month
FROM students;
```

```
-- 获取下一个星期几
SELECT
    CURRENT_DATE() as today,
    NEXT_DAY(CURRENT_DATE(), 'MON') as next_monday,
    NEXT_DAY(CURRENT_DATE(), 'FRI') as next_friday;


-- 时间戳转换
SELECT
    exam_date,
    UNIX_TIMESTAMP(exam_date) as timestamp,
    FROM_UNIXTIME(UNIX_TIMESTAMP(exam_date), 'yyyyMMdd') as formatted_ts
FROM scores
LIMIT 5;


-- 综合案例:学生入学分析
SELECT
    YEAR(enroll_date) as enroll_year,
    QUARTER(enroll_date) as enroll_quarter,
    COUNT(*) as student_count,
    ROUND(DATEDIFF(CURRENT_DATE(), MIN(enroll_date)) / 365, 1) as max_years_enrolled
FROM students
GROUP BY YEAR(enroll_date), QUARTER(enroll_date)
ORDER BY enroll_year, enroll_quarter;


-- 计算考试日期的星期几
SELECT
    exam_date,
    CASE DAYOFWEEK(exam_date)
        WHEN 1 THEN '星期日'
        WHEN 2 THEN '星期一'
        WHEN 3 THEN '星期二'
        WHEN 4 THEN '星期三'
        WHEN 5 THEN '星期四'
        WHEN 6 THEN '星期五'
        WHEN 7 THEN '星期六'
    END as weekday_name,
    COUNT(*) as exam_count
FROM scores
GROUP BY exam_date, DAYOFWEEK(exam_date);
```

练习题:

计算每个学生从入学到现在的天数

统计每个月入学的学生数量

查询入学超过1年的学生信息

3.5 条件函数实践

```sql
-- IF函数
SELECT
    stu_id,
    course_id,
    score,
    IF(score >= 60, '及格', '不及格') as result,
    IF(score >= 90, '优秀', '非优秀') as level
FROM scores;


-- CASE WHEN - 简单形式
SELECT
    stu_id,
    course_id,
    score,
    CASE score >= 60
        WHEN true THEN '及格'
        ELSE '不及格'
    END as pass_status
FROM scores;


-- CASE WHEN - 搜索形式(推荐)
SELECT
    stu_id,
    course_id,
    score,
    CASE
        WHEN score >= 90 THEN 'A-优秀'
        WHEN score >= 80 THEN 'B-良好'
        WHEN score >= 70 THEN 'C-中等'
        WHEN score >= 60 THEN 'D-及格'
        ELSE 'F-不及格'
    END as grade,
    CASE
        WHEN score >= 90 THEN 4.0
        WHEN score >= 80 THEN 3.0
        WHEN score >= 70 THEN 2.0
        WHEN score >= 60 THEN 1.0
        ELSE 0.0
    END as gpa
FROM scores;


-- COALESCE: 返回第一个非NULL值
SELECT
    stu_id,
    COALESCE(NULL, NULL, score, 0) as score_or_default,
    COALESCE(NULL, score) as score_coalesce
FROM scores
LIMIT 5;


-- NVL: 如果第一个参数为NULL,返回第二个参数
SELECT
    stu_id,
    score,
    NVL(score, 0) as score_with_default,
    NVL(NULL, 60) as default_pass_score
FROM scores
```

```
LIMIT 5;


-- NULLIF: 如果两个参数相等,返回NULL,否则返回第一个参数
SELECT
    score,
    NULLIF(score, 85) as nullif_85,  -- 如果score=85则返回NULL
    NULLIF(score, 0) as nullif_0      -- 如果score=0则返回NULL
FROM scores
LIMIT 10;


-- 综合案例1:成绩分析报告
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score,
    CASE
        WHEN s.score >= 90 THEN '优秀'
        WHEN s.score >= 80 THEN '良好'
        WHEN s.score >= 70 THEN '中等'
        WHEN s.score >= 60 THEN '及格'
        ELSE '不及格'
    END as level,
    IF(s.score >= AVG(s.score) OVER(), '高于平均分', '低于平均分') as vs_avg,
    CASE
        WHEN s.score >= 85 THEN '免考下学期'
        WHEN s.score < 60 THEN '需要补考'
        ELSE '正常'
    END as status
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
JOIN courses c ON s.course_id = c.course_id
ORDER BY s.stu_id, s.score DESC;


-- 综合案例2:学生档案
SELECT
    stu_id,
    stu_name,
    age,
    CASE
        WHEN age < 20 THEN '19岁及以下'
        WHEN age >= 20 AND age < 22 THEN '20-21岁'
        ELSE '22岁及以上'
    END as age_group,
    college,
    CASE college
        WHEN '计算机学院' THEN '工科'
        WHEN '数学学院' THEN '理科'
        WHEN '物理学院' THEN '理科'
        WHEN '化学学院' THEN '理科'
        ELSE '其他'
    END as college_type,
    DATEDIFF(CURRENT_DATE(), enroll_date) as days_enrolled,
```

```
    CASE
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date) < 365 THEN '一年级'
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date) < 730 THEN '二年级'
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date) < 1095 THEN '三年级'
        ELSE '四年级及以上'
    END as grade_level
FROM students;
```

练习题:

创建一个查询,将分数转换为五级制(优秀/良好/中等/及格/不及格)

统计每个成绩等级的学生人数

查询成绩在各科平均分以上的学生

3.6 聚合函数应用

```SQL
-- 基本聚合函数
SELECT
    COUNT(*) as total_records,                    -- 总记录数
    COUNT(DISTINCT stu_id) as unique_students,    -- 不同学生数
    COUNT(DISTINCT course_id) as unique_courses,  -- 不同课程数
    SUM(score) as total_score,                    -- 总分
    AVG(score) as avg_score,                      -- 平均分
    MAX(score) as max_score,                      -- 最高分
    MIN(score) as min_score,                      -- 最低分
    VARIANCE(score) as variance,                  -- 方差
    VAR_POP(score) as var_population,             -- 总体方差
    STDDEV(score) as stddev,                      -- 标准差
    STDDEV_POP(score) as stddev_population        -- 总体标准差
FROM scores;


-- 按学生分组聚合
SELECT
    stu_id,
    COUNT(*) as course_count,
    ROUND(AVG(score), 2) as avg_score,
    SUM(score) as total_score,
    MAX(score) as max_score,
    MIN(score) as min_score,
    MAX(score) - MIN(score) as score_range
FROM scores
GROUP BY stu_id
ORDER BY avg_score DESC;


-- 按课程分组聚合
SELECT
    course_id,
    COUNT(*) as student_count,
```

```
    ROUND(AVG(score), 2) as avg_score,
    MAX(score) as max_score,
    MIN(score) as min_score,
    SUM(CASE WHEN score >= 60 THEN 1 ELSE 0 END) as pass_count,
    ROUND(SUM(CASE WHEN score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate
FROM scores
GROUP BY course_id
ORDER BY avg_score DESC;


-- HAVING子句过滤
SELECT
    stu_id,
    COUNT(*) as course_count,
    ROUND(AVG(score), 2) as avg_score
FROM scores
GROUP BY stu_id
HAVING AVG(score) > 85;


-- 多列分组
SELECT
    YEAR(exam_date) as exam_year,
    MONTH(exam_date) as exam_month,
    course_id,
    COUNT(*) as exam_count,
    ROUND(AVG(score), 2) as avg_score
FROM scores
GROUP BY YEAR(exam_date), MONTH(exam_date), course_id
ORDER BY exam_year, exam_month, course_id;


-- COLLECT_SET和COLLECT_LIST
SELECT
    stu_id,
    COLLECT_SET(course_id) as courses_taken,      -- 去重课程列表
    COLLECT_LIST(score) as all_scores,             -- 所有分数列表
    COLLECT_LIST(course_id) as course_list         -- 课程列表(不去重)
FROM scores
GROUP BY stu_id;


-- 窗口聚合函数
SELECT
    stu_id,
    course_id,
    score,
    AVG(score) OVER() as overall_avg,                    -- 全局平均分
    AVG(score) OVER(PARTITION BY course_id) as course_avg,  -- 课程平均分
    AVG(score) OVER(PARTITION BY stu_id) as student_avg,    -- 学生平均分
    MAX(score) OVER(PARTITION BY course_id) as course_max,  -- 课程最高分
    MIN(score) OVER(PARTITION BY course_id) as course_min   -- 课程最低分
FROM scores
ORDER BY stu_id, course_id;


-- 排名函数
SELECT
    stu_id,
    course_id,
```

```
    score,
    ROW_NUMBER() OVER(PARTITION BY course_id ORDER BY score DESC) as row_num,
    RANK() OVER(PARTITION BY course_id ORDER BY score DESC) as rank,
    DENSE_RANK() OVER(PARTITION BY course_id ORDER BY score DESC) as dense_rank,
    PERCENT_RANK() OVER(PARTITION BY course_id ORDER BY score DESC) as percent_rank
FROM scores
ORDER BY course_id, score DESC;


-- 综合案例:成绩排名报表
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score,
    ROUND(AVG(s.score) OVER(PARTITION BY s.course_id), 2) as course_avg,
    s.score - AVG(s.score) OVER(PARTITION BY s.course_id) as vs_avg_diff,
    RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) as course_rank,
    CASE
        WHEN RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) <= 3 THEN '前
三名'
        WHEN s.score >= AVG(s.score) OVER(PARTITION BY s.course_id) THEN '中上'
        ELSE '中下'
    END as performance
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
JOIN courses c ON s.course_id = c.course_id
ORDER BY s.course_id, s.score DESC;
```

练习题:

统计每个学院的学生人数和平均年龄

查询平均分排名前3的学生信息

统计每门课程的及格人数和及格率

第四部分:查询与分析综合实践 (60 分钟)

4.1 复杂 WHERE 条件查询

```SQL
-- 准备订单相关表
CREATE TABLE orders (
    order_id STRING,
    customer_id INT,
    order_time TIMESTAMP,
    amount DECIMAL(10,2),
    status STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
LOAD DATA LOCAL INPATH '/usr/datas/unit3/orders.txt'
OVERWRITE INTO TABLE orders;


CREATE TABLE order_items (
    order_id STRING,
    product_id STRING,
    product_name STRING,
    quantity INT,
    price DECIMAL(10,2)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';


LOAD DATA LOCAL INPATH '/usr/datas/unit3/order_items.txt'
OVERWRITE INTO TABLE order_items;


-- 复杂条件查询示例
-- 查询1:多条件组合
SELECT *
FROM orders
WHERE amount > 1000
    AND status IN ('已支付', '已发货')
    AND DATE(order_time) >= '2024-01-02';


-- 查询2:子查询作为条件
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT stu_id
    FROM students
    WHERE college = '计算机学院'
);


-- 查询3:相关子查询
SELECT o.*
FROM orders o
WHERE amount > (
    SELECT AVG(amount)
    FROM orders
    WHERE customer_id = o.customer_id
);


-- 查询4:EXISTS子查询
SELECT s.*
FROM students s
WHERE EXISTS (
    SELECT 1
    FROM scores sc
    WHERE sc.stu_id = s.stu_id AND sc.score > 90
);


-- 查询5:NOT EXISTS子查询
SELECT s.*
```

```
FROM students s
WHERE NOT EXISTS (
    SELECT 1
    FROM scores sc
    WHERE sc.stu_id = s.stu_id AND sc.score < 60
);


-- 查询6:ALL和ANY
SELECT *
FROM scores
WHERE score > ALL (
    SELECT score
    FROM scores
    WHERE course_id = 'C001'
);


SELECT *
FROM scores
WHERE score > ANY (
    SELECT score
    FROM scores
    WHERE course_id = 'C001' AND score < 80
);


-- 查询7:复杂模糊匹配
SELECT *
FROM students
WHERE stu_name LIKE '张%'      -- 姓张
   OR stu_name LIKE '%三%'     -- 名字含三
   OR college LIKE '%计算机%'; -- 学院名含计算机


-- 查询8:正则表达式匹配
SELECT *
FROM students
WHERE stu_name REGEXP '^[张李王]'  -- 姓张、李或王
   OR phone REGEXP '^186';         -- 手机号以186开头
```

练习题:

查询购买金额最高的客户的所有订单

查询至少选修了3门课程的学生信息

查询所有科目都及格的学生

4.2 GROUP BY 与聚合分析

```
SQL
-- 基础分组
SELECT
```

```
    college,
    COUNT(*) as student_count,
    ROUND(AVG(age), 1) as avg_age
FROM students
GROUP BY college
ORDER BY student_count DESC;


-- 多列分组
SELECT
    college,
    gender,
    COUNT(*) as count,
    ROUND(AVG(age), 1) as avg_age
FROM students
GROUP BY college, gender
ORDER BY college, gender;


-- GROUP BY with ROLLUP(汇总)
SELECT
    college,
    gender,
    COUNT(*) as count
FROM students
GROUP BY college, gender WITH ROLLUP;


-- GROUPING SETS(分组集)
SELECT
    college,
    gender,
    COUNT(*) as count
FROM students
GROUP BY college, gender
GROUPING SETS (
    (college, gender),  -- 按学院和性别分组
    (college),          -- 只按学院分组
    (gender),           -- 只按性别分组
    ()                  -- 总计
);


-- CUBE(多维分析)
SELECT
    college,
    gender,
    COUNT(*) as count
FROM students
GROUP BY college, gender WITH CUBE;


-- HAVING过滤分组结果
SELECT
    college,
    COUNT(*) as student_count,
    ROUND(AVG(age), 1) as avg_age
FROM students
GROUP BY college
HAVING COUNT(*) >= 2 AND AVG(age) > 19
ORDER BY student_count DESC;
```

```
-- 聚合后计算
SELECT
    course_id,
    COUNT(*) as student_count,
    ROUND(AVG(score), 2) as avg_score,
    ROUND(STDDEV(score), 2) as stddev_score,
    MAX(score) - MIN(score) as score_range,
    ROUND(SUM(CASE WHEN score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate
FROM scores
GROUP BY course_id
HAVING COUNT(*) >= 3
ORDER BY avg_score DESC;


-- 综合案例:学生成绩分析
SELECT
    st.college,
    COUNT(DISTINCT s.stu_id) as student_count,
    COUNT(*) as total_exams,
    ROUND(AVG(s.score), 2) as avg_score,
    MAX(s.score) as max_score,
    MIN(s.score) as min_score,
    SUM(CASE WHEN s.score >= 90 THEN 1 ELSE 0 END) as excellent_count,
    SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) as pass_count,
    ROUND(SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
GROUP BY st.college
ORDER BY avg_score DESC;
```

练习题:

统计每个城市的学生数量和平均年龄

查询每门课程的及格率,要求只显示及格率低于90%的课程

统计每个学生的选课数和平均分,并找出平均分前5名

4.3 ORDER BY 与排序

```
SQL
-- 单列排序
SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC;  -- 降序


SELECT stu_id, course_id, score
FROM scores
ORDER BY score ASC;   -- 升序(默认)
```

```
-- 多列排序
SELECT stu_id, course_id, score
FROM scores
ORDER BY stu_id ASC, score DESC;


-- NULL值排序
SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC NULLS LAST;   -- NULL值排在最后


SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC NULLS FIRST;  -- NULL值排在最前


-- 按表达式排序
SELECT
    stu_id,
    course_id,
    score,
    CASE
        WHEN score >= 90 THEN 1
        WHEN score >= 80 THEN 2
        WHEN score >= 70 THEN 3
        WHEN score >= 60 THEN 4
        ELSE 5
    END as level
FROM scores
ORDER BY level, score DESC;


-- 按聚合结果排序
SELECT
    stu_id,
    COUNT(*) as course_count,
    ROUND(AVG(score), 2) as avg_score
FROM scores
GROUP BY stu_id
ORDER BY avg_score DESC, course_count DESC;


-- LIMIT子句(分页)
SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC
LIMIT 10;  -- 前10名


SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC
LIMIT 5, 5;  -- 跳过前5条,取接下来的5条(第6-10名)


-- 综合案例:成绩排行榜
SELECT
    RANK() OVER(ORDER BY avg_score DESC) as ranking,
    t.*
```

```
FROM (
    SELECT
        s.stu_id,
        st.stu_name,
        st.college,
        COUNT(*) as course_count,
        ROUND(AVG(s.score), 2) as avg_score,
        MAX(s.score) as max_score,
        MIN(s.score) as min_score
    FROM scores s
    JOIN students st ON s.stu_id = st.stu_id
    GROUP BY s.stu_id, st.stu_name, st.college
) t
ORDER BY avg_score DESC
LIMIT 10;
```

练习题:

查询每门课程成绩最高的3名学生

按学院和平均分排序,显示前10名学生

查询每个学生的排名(按平均分)

4.4 JOIN 多表关联

```SQL
-- INNER JOIN(内连接)
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score
FROM scores s
INNER JOIN students st ON s.stu_id = st.stu_id
INNER JOIN courses c ON s.course_id = c.course_id
ORDER BY s.stu_id, s.course_id;


-- LEFT JOIN(左外连接)
SELECT
    st.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score
FROM students st
LEFT JOIN scores s ON st.stu_id = s.stu_id
LEFT JOIN courses c ON s.course_id = c.course_id
ORDER BY st.stu_id;


-- RIGHT JOIN(右外连接)
SELECT
    c.course_id,
```

```
    c.course_name,
    s.stu_id,
    st.stu_name,
    s.score
FROM scores s
RIGHT JOIN courses c ON s.course_id = c.course_id
LEFT JOIN students st ON s.stu_id = st.stu_id
ORDER BY c.course_id;


-- FULL OUTER JOIN(全外连接)
SELECT
    COALESCE(st.stu_id, s.stu_id) as stu_id,
    st.stu_name,
    s.course_id,
    s.score
FROM students st
FULL OUTER JOIN scores s ON st.stu_id = s.stu_id
ORDER BY stu_id;


-- CROSS JOIN(交叉连接/笛卡尔积)
SELECT
    st.stu_id,
    st.stu_name,
    c.course_id,
    c.course_name
FROM students st
CROSS JOIN courses c
WHERE st.college = '计算机学院'
ORDER BY st.stu_id, c.course_id
LIMIT 20;


-- SELF JOIN(自连接)
SELECT
    s1.stu_id as stu_id1,
    st1.stu_name as stu_name1,
    s2.stu_id as stu_id2,
    st2.stu_name as stu_name2,
    s1.course_id
FROM scores s1
JOIN scores s2 ON s1.course_id = s2.course_id AND s1.stu_id < s2.stu_id
JOIN students st1 ON s1.stu_id = st1.stu_id
JOIN students st2 ON s2.stu_id = st2.stu_id
WHERE ABS(s1.score - s2.score) <= 5  -- 分数差不超过5分
ORDER BY s1.course_id, s1.stu_id;


-- 多表复杂关联
SELECT
    st.college,
    c.course_name,
    COUNT(DISTINCT s.stu_id) as student_count,
    ROUND(AVG(s.score), 2) as avg_score,
    MAX(s.score) as max_score,
    MIN(s.score) as min_score
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
JOIN courses c ON s.course_id = c.course_id
GROUP BY st.college, c.course_name
```

```
ORDER BY st.college, avg_score DESC;


-- 带子查询的JOIN
SELECT
    st.*,
    avg_scores.avg_score,
    avg_scores.max_score
FROM students st
JOIN (
    SELECT
        stu_id,
        ROUND(AVG(score), 2) as avg_score,
        MAX(score) as max_score
    FROM scores
    GROUP BY stu_id
) avg_scores ON st.stu_id = avg_scores.stu_id
WHERE avg_scores.avg_score > 85
ORDER BY avg_scores.avg_score DESC;


-- 综合案例:学生成绩单
SELECT
    st.stu_id,
    st.stu_name,
    st.gender,
    st.college,
    c.course_name,
    c.credits,
    c.teacher,
    s.score,
    CASE
        WHEN s.score >= 90 THEN 'A'
        WHEN s.score >= 80 THEN 'B'
        WHEN s.score >= 70 THEN 'C'
        WHEN s.score >= 60 THEN 'D'
        ELSE 'F'
    END as grade,
    s.exam_date,
    RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) as course_rank
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
JOIN courses c ON s.course_id = c.course_id
ORDER BY st.stu_id, s.score DESC;
```

练习题:

查询选修了"大数据技术"课程的所有学生信息

查询没有考试记录的学生

查询每个学院各门课程的平均分

4.5 子查询应用

```sql
-- WHERE子查询
-- 查询成绩高于平均分的记录
SELECT stu_id, course_id, score
FROM scores
WHERE score > (SELECT AVG(score) FROM scores);


-- FROM子查询(内联视图)
SELECT
    t.college,
    t.student_count,
    t.avg_age
FROM (
    SELECT
        college,
        COUNT(*) as student_count,
        ROUND(AVG(age), 1) as avg_age
    FROM students
    GROUP BY college
) t
WHERE t.student_count >= 2
ORDER BY t.student_count DESC;


-- SELECT子查询(标量子查询)
SELECT
    stu_id,
    course_id,
    score,
    (SELECT AVG(score) FROM scores) as avg_score,
    score - (SELECT AVG(score) FROM scores) as vs_avg
FROM scores;


-- 相关子查询
SELECT
    s.*,
    (SELECT AVG(score)
     FROM scores
     WHERE course_id = s.course_id) as course_avg
FROM scores s
WHERE score > (
    SELECT AVG(score)
    FROM scores
    WHERE course_id = s.course_id
);


-- IN子查询
SELECT *
FROM students
WHERE stu_id IN (
    SELECT stu_id
    FROM scores
    WHERE score >= 90
);


-- NOT IN子查询
SELECT *
FROM students
```

```
WHERE stu_id NOT IN (
    SELECT stu_id
    FROM scores
    WHERE score < 60
);


-- EXISTS子查询
SELECT *
FROM courses c
WHERE EXISTS (
    SELECT 1
    FROM scores s
    WHERE s.course_id = c.course_id
);


-- NOT EXISTS子查询
SELECT *
FROM students st
WHERE NOT EXISTS (
    SELECT 1
    FROM scores s
    WHERE s.stu_id = st.stu_id AND s.score < 60
);


-- 多层嵌套子查询
SELECT *
FROM students
WHERE stu_id IN (
    SELECT stu_id
    FROM scores
    WHERE course_id IN (
        SELECT course_id
        FROM courses
        WHERE credits >= 4
    )
    GROUP BY stu_id
    HAVING AVG(score) > 85
);


-- 综合案例:复杂分析查询
SELECT
    st.stu_id,
    st.stu_name,
    st.college,
    overall.course_count,
    overall.avg_score,
    overall.max_score,
    overall.min_score,
    CASE
        WHEN overall.avg_score >= 90 THEN '优等生'
        WHEN overall.avg_score >= 80 THEN '良好学生'
        WHEN overall.avg_score >= 70 THEN '中等学生'
        ELSE '普通学生'
    END as student_level
FROM students st
JOIN (
```

```
    SELECT
        stu_id,
        COUNT(*) as course_count,
        ROUND(AVG(score), 2) as avg_score,
        MAX(score) as max_score,
        MIN(score) as min_score
    FROM scores
    GROUP BY stu_id
) overall ON st.stu_id = overall.stu_id
WHERE overall.avg_score > (
    SELECT AVG(score)
    FROM scores
    WHERE stu_id IN (
        SELECT stu_id
        FROM students
        WHERE college = st.college
    )
)
ORDER BY overall.avg_score DESC;
```

练习题:

查询平均分最高的学生的详细信息

查询每门课程中成绩排名前3的学生

查询同时选修了C001和C002课程的学生

第五部分:数据导入导出实践 (30 分钟)

5.1 多种数据导入方式

```SQL
-- 方式1: LOAD DATA LOCAL INPATH(从本地文件系统加载)
LOAD DATA LOCAL INPATH '/usr/datas/unit3/students.txt'
OVERWRITE INTO TABLE students;


-- 方式2: LOAD DATA INPATH(从HDFS加载)
-- 首先上传文件到HDFS
-- hadoop fs -put /usr/datas/unit3/students.txt /tmp/
-- LOAD DATA INPATH '/tmp/students.txt'
-- OVERWRITE INTO TABLE students;


-- 方式3: INSERT INTO SELECT(从其他表插入)
CREATE TABLE students_backup AS SELECT * FROM students;


INSERT INTO TABLE students_backup
SELECT * FROM students WHERE age > 20;
```

```
-- 方式4: INSERT INTO VALUES(直接插入数据)
CREATE TABLE test_insert (
    id INT,
    name STRING,
    age INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';


INSERT INTO TABLE test_insert VALUES
(1, '测试1', 20),
(2, '测试2', 21),
(3, '测试3', 22);


SELECT * FROM test_insert;


-- 方式5: CREATE TABLE AS SELECT(CTAS)
CREATE TABLE high_score_students AS
SELECT
    s.stu_id,
    st.stu_name,
    st.college,
    ROUND(AVG(s.score), 2) as avg_score
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
GROUP BY s.stu_id, st.stu_name, st.college
HAVING AVG(s.score) >= 85;


SELECT * FROM high_score_students;


-- 方式6: 从Hive表导入MySQL(使用Sqoop - 仅演示)
-- sqoop export \
-- --connect jdbc:mysql://localhost:3306/unit3db \
-- --username hiveuser \
-- --password 123456 \
-- --table students_mysql \
-- --export-dir /user/hive/warehouse/unit3db.db/students \
-- --input-fields-terminated-by ','


-- 方式7: 多文件批量导入
-- 假设有多个文件在同一目录下
-- LOAD DATA LOCAL INPATH '/usr/datas/unit3/*'
-- INTO TABLE target_table;
```

5.2 数据导出实践

```
SQL
-- 方式1: INSERT OVERWRITE LOCAL DIRECTORY(导出到本地文件系统)
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/students_export'
```

```
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM students;


-- 方式2: 导出查询结果
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/score_analysis'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT
    s.stu_id,
    st.stu_name,
    st.college,
    s.course_id,
    c.course_name,
    s.score
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
JOIN courses c ON s.course_id = c.course_id
WHERE s.score >= 85;


-- 方式3: 导出到HDFS
INSERT OVERWRITE DIRECTORY '/user/hive/export/students'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM students;


-- 方式4: 导出为多个文件(按分区)
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/by_college'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT college, stu_id, stu_name, age
FROM students
DISTRIBUTE BY college;  -- 按学院分发到不同文件


-- 方式5: 使用Beeline导出
-- beeline -u jdbc:hive2://localhost:10000 \
-- -e "SELECT * FROM unit3db.students" \
-- --outputformat=csv2 \
-- > /usr/datas/unit3/output/students.csv


-- 方式6: 导出到MySQL(使用Sqoop - 仅演示)
-- sqoop export \
-- --connect jdbc:mysql://localhost:3306/unit3db \
-- --username hiveuser \
-- --password 123456 \
-- --table students_mysql \
-- --export-dir /user/hive/warehouse/unit3db.db/students
```

5.3 综合案例

案例1: 电商订单分析系统

```sql
-- 创建分析结果表
CREATE TABLE order_analysis AS
SELECT
    o.customer_id,
    st.stu_name as customer_name,
    st.city,
    COUNT(DISTINCT o.order_id) as order_count,
    SUM(o.amount) as total_amount,
    ROUND(AVG(o.amount), 2) as avg_order_amount,
    MAX(o.amount) as max_order_amount,
    MIN(DATE(o.order_time)) as first_order_date,
    MAX(DATE(o.order_time)) as last_order_date,
    DATEDIFF(MAX(DATE(o.order_time)), MIN(DATE(o.order_time))) as
customer_lifetime_days
FROM orders o
JOIN students st ON o.customer_id = st.stu_id
GROUP BY o.customer_id, st.stu_name, st.city;


-- 查看分析结果
SELECT * FROM order_analysis ORDER BY total_amount DESC;


-- 导出分析结果
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/order_analysis'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM order_analysis;
```

案例2: 学生成绩综合分析

```sql
-- 创建成绩分析宽表
CREATE TABLE student_performance_analysis AS
SELECT
    st.stu_id,
    st.stu_name,
    st.gender,
    st.age,
    st.college,
    st.city,
    COUNT(DISTINCT s.course_id) as course_count,
    ROUND(AVG(s.score), 2) as avg_score,
    MAX(s.score) as max_score,
    MIN(s.score) as min_score,
    SUM(CASE WHEN s.score >= 90 THEN 1 ELSE 0 END) as excellent_count,
    SUM(CASE WHEN s.score >= 80 AND s.score < 90 THEN 1 ELSE 0 END) as good_count,
    SUM(CASE WHEN s.score >= 70 AND s.score < 80 THEN 1 ELSE 0 END) as medium_count,
    SUM(CASE WHEN s.score >= 60 AND s.score < 70 THEN 1 ELSE 0 END) as pass_count,
    SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) as fail_count,
    CASE
        WHEN AVG(s.score) >= 90 THEN 'A级学生'
        WHEN AVG(s.score) >= 80 THEN 'B级学生'
        WHEN AVG(s.score) >= 70 THEN 'C级学生'
        ELSE 'D级学生'
```

```
    END as student_grade
FROM students st
LEFT JOIN scores s ON st.stu_id = s.stu_id
GROUP BY st.stu_id, st.stu_name, st.gender, st.age, st.college, st.city;


-- 查看分析结果
SELECT * FROM student_performance_analysis
ORDER BY avg_score DESC;


-- 按学院统计
SELECT
    college,
    COUNT(*) as student_count,
    ROUND(AVG(avg_score), 2) as college_avg_score,
    SUM(excellent_count) as total_excellent,
    SUM(fail_count) as total_fail
FROM student_performance_analysis
GROUP BY college
ORDER BY college_avg_score DESC;


-- 导出结果
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/student_performance'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM student_performance_analysis;
```

案例3: 多维度数据透视分析

```
SQL
-- 学院-课程-成绩透视分析
CREATE TABLE pivot_analysis AS
SELECT
    st.college,
    s.course_id,
    c.course_name,
    COUNT(*) as student_count,
    ROUND(AVG(s.score), 2) as avg_score,
    MAX(s.score) as max_score,
    MIN(s.score) as min_score,
    SUM(CASE WHEN s.score >= 90 THEN 1 ELSE 0 END) as excellent_count,
    SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) as pass_count,
    ROUND(SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
JOIN courses c ON s.course_id = c.course_id
GROUP BY st.college, s.course_id, c.course_name
WITH CUBE;


-- 查看结果
SELECT * FROM pivot_analysis
WHERE college IS NOT NULL AND course_id IS NOT NULL
ORDER BY college, avg_score DESC;
```

```
-- 导出
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/pivot_analysis'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM pivot_analysis;
```

综合练习题

练习 1: 学生成绩管理系统

要求:

创建一个视图,显示每个学生的学号、姓名、选课数、平均分、最高分、最低分

查询平均分排名前5的学生详细信息

统计每个学院的优秀率(平均分>=90的比例)

导出成绩不及格学生的名单到本地文件

```
SQL
-- 在这里编写你的SQL语句
-- 提示:可以使用CREATE VIEW, JOIN, GROUP BY, HAVING, ORDER BY, LIMIT等
```

练习 2: 课程选修分析

要求:

查询每门课程的选修人数、平均分、及格率

找出最受欢迎的3门课程(选修人数最多)

找出最难的3门课程(平均分最低)

创建一个报表,显示每门课程各分数段的人数分布

```
SQL
-- 在这里编写你的SQL语句
```

练习 3: 数据质量检查

要求:

检查是否有学生没有考试记录

检查是否有课程没有学生选修

检查是否有异常成绩(如负分或超过100分)

统计每个表的记录数和主要字段的NULL值情况

```sql
-- 在这里编写你的SQL语句
```

实践总结

知识点回顾

数据类型: 基本类型、ARRAY、MAP、STRUCT

表类型: 内部表、外部表、分区表

运算符: 算术、比较、逻辑运算符

函数类型:

- 字符串函数
- 日期时间函数
- 条件函数
- 聚合函数
- 窗口函数

查询语句:

- WHERE条件过滤
- GROUP BY分组
- ORDER BY排序
- JOIN多表关联
- 子查询

数据导入导出:

- LOAD DATA
- INSERT INTO/OVERWRITE
- CREATE TABLE AS SELECT

常见问题解决

问题1: beeline连接失败

```bash
# 检查HiveServer2是否启动
ps aux | grep HiveServer2


# 重启HiveServer2
nohup hiveserver2 &
```

问题2: 数据加载失败

```bash
# 检查文件是否存在
ls -l /usr/datas/unit3/


# 检查文件权限
chmod 644 /usr/datas/unit3/*.txt
```

问题3: 中文乱码

```sql
-- 在创建表时指定编码
ALTER TABLE students SET SERDEPROPERTIES ('serialization.encoding'='UTF-8');
```

最佳实践建议

命名规范: 使用有意义的表名和列名

注释习惯: 为表和列添加详细注释

分区策略: 合理使用分区提高查询效率

数据备份: 重要数据操作前先备份

性能优化:

- 使用分区过滤
- 避免SELECT *
- 合理使用JOIN
- 适当使用WHERE过滤条件

扩展学习资源

Hive官方文档: https://hive.apache.org/

Hive SQL语法参考

大数据技术博客和社区

实际项目案例

实践结束

恭喜你完成了Hive第3章的综合实践!

通过本次实践,你应该已经掌握了:

- Hive的基本语法和数据类型
- 各种运算符和函数的使用
- 复杂查询和多表关联
- 数据的导入导出方法

下一步建议:

复习本次实践的重点内容

独立完成综合练习题

尝试在实际项目中应用所学知识

继续学习Hive的高级特性(后续章节)

记得保存你的实践代码和笔记!

教师评语区:

- 完成情况: □优秀 □良好 □及格 □待提高
- 评语: _____
- 日期: _____