Hive 第 3 章 综合实践工作手册 - 练习题答案

目录

- [第二部分 练习题答案](#第二部分-练习题答案)
  - [2.1 复杂数据类型实践](#21-复杂数据类型实践)
  - [2.3 分区表实践](#23-分区表实践)
- [第三部分 练习题答案](#第三部分-练习题答案)
  - [3.1 算术与比较运算符](#31-算术与比较运算符)
  - [3.2 逻辑运算符应用](#32-逻辑运算符应用)
  - [3.3 字符串函数实践](#33-字符串函数实践)
  - [3.4 日期时间函数实践](#34-日期时间函数实践)
  - [3.5 条件函数实践](#35-条件函数实践)
  - [3.6 聚合函数应用](#36-聚合函数应用)
- [第四部分 练习题答案](#第四部分-练习题答案)
  - [4.1 复杂WHERE条件查询](#41-复杂where条件查询)
  - [4.2 GROUP BY与聚合分析](#42-group-by与聚合分析)
  - [4.3 ORDER BY与排序](#43-order-by与排序)
  - [4.4 JOIN多表关联](#44-join多表关联)
  - [4.5 子查询应用](#45-子查询应用)
- [综合练习题答案](#综合练习题答案)

第二部分 练习题答案

2.1 复杂数据类型实践

练习题1: 查询每个学生的爱好数量,并按数量降序排列

答案:

```SQL
SELECT
    stu_name,
    SIZE(hobbies) as hobby_count
FROM student_detail
ORDER BY hobby_count DESC;
```

说明:

- 使用 `SIZE()` 函数获取ARRAY类型的元素个数
- 按爱好数量降序排列,数量多的排在前面

预期结果:

```
stu_name  hobby_count
王五      3
张三      3
李四      2
赵六      2
钱七      2
```

练习题2: 查询所有学生的平均数学成绩

答案:

```SQL
SELECT
    ROUND(AVG(scores['数学']), 2) as avg_math_score,
    COUNT(*) as student_count,
    MAX(scores['数学']) as max_math,
    MIN(scores['数学']) as min_math
FROM student_detail
WHERE scores['数学'] IS NOT NULL;
```

说明:

- 使用MAP类型访问语法 `scores['数学']` 获取数学成绩
- 使用 `AVG()` 计算平均值
- 添加 `WHERE` 条件过滤NULL值
- 额外提供最高分和最低分信息

预期结果:

```
avg_math_score  student_count  max_math  min_math
86.20          5              94        76
```

练习题3: 查询地址在北京的学生姓名和联系邮箱

答案:

```sql
SELECT
    stu_name,
    address.province as province,
    address.city as city,
    address.district as district,
    contact_info['email'] as email
FROM student_detail
WHERE address.province = '北京';
```

说明:

- 使用STRUCT类型访问语法 `address.province` 访问结构体字段
- 使用MAP类型访问语法 `contact_info['email']` 获取邮箱
- WHERE条件过滤出北京的学生

预期结果:

```
stu_name  province  city  district  email
张三       北京       海淀区   中关村     zhangsan@example.com
```

2.3 分区表实践

练习题1: 统计每个分区的订单总金额

答案:

```sql
SELECT
    order_date,
    COUNT(*) as order_count,
    SUM(amount) as total_amount,
    ROUND(AVG(amount), 2) as avg_amount,
    MAX(amount) as max_amount,
    MIN(amount) as min_amount
FROM orders_partitioned
GROUP BY order_date
ORDER BY order_date;
```

说明:

- 按分区字段 `order_date` 进行分组
- 统计每个分区的订单数量、总金额、平均金额、最大和最小金额
- 按日期排序

预期结果:

```
order_date   order_count  total_amount  avg_amount  max_amount  min_amount
2024-01-01   2           3979.50       1989.75     2399.00     1580.50
2024-01-02   2           3799.00       1899.50     3200.00     599.00
2024-01-03   2           2198.00       1099.00     1299.00     899.00
2024-01-04   2           5398.00       2699.00     4599.00     799.00
2024-01-05   1           1899.00       1899.00     1899.00     1899.00
```

练习题2: 查询金额超过2000的订单所在的分区

答案:

```sql
SELECT DISTINCT
    order_date,
    COUNT(*) OVER(PARTITION BY order_date) as orders_in_partition,
    SUM(amount) OVER(PARTITION BY order_date) as partition_total
FROM orders_partitioned
WHERE amount > 2000
ORDER BY order_date;
```

或者更简单的方式:

```sql
SELECT
    order_date,
    COUNT(*) as high_value_order_count,
    SUM(amount) as high_value_total
FROM orders_partitioned
WHERE amount > 2000
GROUP BY order_date
ORDER BY order_date;
```

说明:

- 使用WHERE条件过滤金额大于2000的订单
- 第一种方法使用窗口函数显示分区级别的统计
- 第二种方法使用GROUP BY进行聚合
- 两种方法都能找出包含高金额订单的分区

预期结果:

```
order_date   high_value_order_count  high_value_total
```

```
2024-01-01   1                    2399.00
2024-01-02   1                    3200.00
2024-01-04   1                    4599.00
```

练习题3: 删除order_date='2024-01-05'的分区

答案:

```sql
-- 查看删除前的分区列表
SHOW PARTITIONS orders_partitioned;

-- 删除指定分区
ALTER TABLE orders_partitioned DROP IF EXISTS PARTITION (order_date='2024-01-05');

-- 验证删除结果
SHOW PARTITIONS orders_partitioned;

-- 确认该分区数据已不存在
SELECT * FROM orders_partitioned WHERE order_date = '2024-01-05';
```

说明:

- 使用 `ALTER TABLE ... DROP PARTITION` 删除指定分区
- `IF EXISTS` 避免分区不存在时报错
- 删除分区会同时删除该分区的元数据和数据文件
- 验证时应该看不到2024-01-05分区,查询也应该返回空结果

预期结果:

删除后分区列表应该只包含:

```
order_date=2024-01-01
order_date=2024-01-02
order_date=2024-01-03
order_date=2024-01-04
```

第三部分 练习题答案

3.1 算术与比较运算符

练习题1: 查询平均分在85-95之间的学生ID

答案:

```sql
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
HAVING AVG(score) BETWEEN 85 AND 95
ORDER BY avg_score DESC;
```

说明:

- 先按学生分组,计算每个学生的平均分
- 使用HAVING子句(不是WHERE)过滤聚合后的结果
- BETWEEN包含边界值,即 >= 85 AND <= 95

预期结果:

```
stu_id  course_count  avg_score  max_score  min_score
1005    3            90.67      94         87
1001    3            84.33      90         78  (如果算上78可能不在范围)
1002    2            90.00      92         88
1007    1            92.00      92         92
1010    2            91.00      93         89
1009    2            88.00      90         86
1006    1            88.00      88         88
```

练习题2: 计算每门课程的最高分、最低分和平均分

答案:

```sql
SELECT
    course_id,
    COUNT(*) as student_count,
    MAX(score) as max_score,
    MIN(score) as min_score,
    ROUND(AVG(score), 2) as avg_score,
    MAX(score) - MIN(score) as score_range,
```

```
    ROUND(STDDEV(score), 2) as stddev_score
FROM scores
GROUP BY course_id
ORDER BY course_id;
```

说明:

- 按课程ID分组
- 使用MAX(), MIN(), AVG()计算统计值
- 额外计算分数范围和标准差

预期结果:

```
course_id  student_count  max_score  min_score  avg_score  score_range  stddev_score
C001       8              94         76         88.38      18           5.68
C002       6              92         86         88.67      6            2.16
C003       4              95         78         86.75      17           7.27
```

练习题3: 查询有不及格科目的学生ID

答案:

```SQL
-- 方法1: 使用子查询
SELECT DISTINCT stu_id
FROM scores
WHERE stu_id IN (
    SELECT stu_id
    FROM scores
    WHERE score < 60
)
ORDER BY stu_id;


-- 方法2: 直接查询
SELECT DISTINCT
    stu_id,
    COUNT(*) as fail_count,
    COLLECT_LIST(course_id) as failed_courses,
    COLLECT_LIST(score) as failed_scores
FROM scores
WHERE score < 60
GROUP BY stu_id
ORDER BY stu_id;


-- 方法3: 使用EXISTS
SELECT DISTINCT s1.stu_id
FROM scores s1
WHERE EXISTS (
```

```
    SELECT 1
    FROM scores s2
    WHERE s2.stu_id = s1.stu_id AND s2.score < 60
)
ORDER BY stu_id;
```

说明:

- 三种方法都可以找出有不及格科目的学生
- 方法2更详细,显示不及格的课程和分数
- 根据提供的数据,所有学生成绩都>=60,所以结果应该为空

预期结果:

根据提供的数据,没有学生不及格,结果为空集。

3.2 逻辑运算符应用

练习题1: 查询分数在60-80之间或90以上的记录

答案:

```
SQL
SELECT
    stu_id,
    course_id,
    score,
    CASE
        WHEN score >= 90 THEN '优秀(>=90)'
        WHEN score BETWEEN 60 AND 80 THEN '中等(60-80)'
        ELSE '其他'
    END as score_category
FROM scores
WHERE (score BETWEEN 60 AND 80) OR (score >= 90)
ORDER BY score DESC, stu_id, course_id;
```

说明:

- 使用OR连接两个条件
- BETWEEN包含边界值
- 可以添加分类标签便于理解
- 按分数降序排列

预期结果:

```
stu_id  course_id  score  score_category
1003    C003       95     优秀(>=90)
```

```
1005    C001        94      优秀(>=90)
1002    C001        92      优秀(>=90)
1007    C002        92      优秀(>=90)
1010    C001        93      优秀(>=90)
1005    C003        91      优秀(>=90)
1001    C002        90      优秀(>=90)
1009    C001        90      优秀(>=90)
1001    C003        78      中等(60-80)
1003    C001        76      中等(60-80)
```

练习题2: 查询计算机学院且成绩优秀(>=90)的学生

答案:

```sql
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 st.college = '计算机学院'
    AND s.score >= 90
ORDER BY s.score DESC, s.stu_id;
```

说明:

- 使用AND连接两个条件
- 需要JOIN students表获取学院信息
- 可以JOIN courses表显示课程名称
- 两个条件都必须满足

预期结果:

```
stu_id  stu_name  college      course_id  course_name      score
1005    钱七       计算机学院    C001       大数据技术        94
1005    钱七       计算机学院    C003       Python编程        91
1007    周九       计算机学院    C002       数据库原理        92
1009    郑一       计算机学院    C001       大数据技术        90
1001    张三       计算机学院    C002       数据库原理        90
```

练习题3: 查询非C001课程且成绩不及格的记录

答案:

```sql
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score,
    '不及格' 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
WHERE s.course_id != 'C001'
    AND s.score < 60
ORDER BY s.score, s.stu_id;


-- 或使用NOT IN
SELECT
    s.stu_id,
    st.stu_name,
    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.course_id NOT IN ('C001')
    AND s.score < 60
ORDER BY s.score, s.stu_id;
```

说明:

- 使用 `!=` 或 `<>` 或 `NOT IN` 排除C001课程
- 使用 `AND` 连接不及格条件
- 根据提供的数据,所有成绩都>=60,所以结果为空

预期结果:

根据提供的数据,没有不及格记录,结果为空集。

3.3 字符串函数实践

练习题1: 提取所有学生的姓氏,并统计每个姓氏的人数

答案:

```sql
SELECT
    SUBSTR(stu_name, 1, 1) as surname,
    COUNT(*) as student_count,
    COLLECT_LIST(stu_name) as students
FROM students
GROUP BY SUBSTR(stu_name, 1, 1)
ORDER BY student_count DESC, surname;
```

说明:

- 使用 `SUBSTR(stu_name, 1, 1)` 提取第一个字符作为姓氏
- 按姓氏分组统计人数
- 使用 `COLLECT_LIST` 列出该姓氏的所有学生名字
- 按人数降序,同人数按姓氏排序

预期结果:

```
surname  student_count  students
王       2              ["王五","王二"]
张       1              ["张三"]
李       1              ["李四"]
赵       1              ["赵六"]
钱       1              ["钱七"]
孙       1              ["孙八"]
周       1              ["周九"]
吴       1              ["吴十"]
郑       1              ["郑一"]
```

练习题2: 将所有课程名称转换为"[课程ID]-课程名称"格式

答案:

```sql
SELECT
    course_id,
    course_name,
    CONCAT('[', course_id, ']-', course_name) as formatted_course,
    CONCAT_WS('-', CONCAT('[', course_id, ']'), course_name) as formatted_course2,
    credits,
    teacher
FROM courses
ORDER BY course_id;
```

说明:

- 使用 `CONCAT()` 函数连接字符串
- 第一种方法直接拼接
- 第二种方法使用 `CONCAT_WS()` 和分隔符
- 保留原始字段便于对比

预期结果:

```
course_id  course_name      formatted_course          credits  teacher
C001       大数据技术       [C001]-大数据技术          4        张教授
C002       数据库原理       [C002]-数据库原理          3        李教授
C003       Python编程       [C003]-Python编程          4        王教授
C004       机器学习         [C004]-机器学习            3        赵教授
C005       数据结构         [C005]-数据结构            4        钱教授
```

练习题3: 对手机号进行脱敏处理,只显示前3位和后4位

答案:

```SQL
SELECT
    stu_id,
    stu_name,
    phone as original_phone,
    CONCAT(
        SUBSTRING(phone, 1, 3),
        '****',
        SUBSTRING(phone, -4)
    ) as masked_phone,
    -- 或使用SUBSTR
    CONCAT(
        SUBSTR(phone, 1, 3),
        '****',
        SUBSTR(phone, LENGTH(phone)-3, 4)
    ) as masked_phone2
FROM students
ORDER BY stu_id;
```

说明:

- 使用 `SUBSTRING(phone, 1, 3)` 提取前3位
- 使用 `SUBSTRING(phone, -4)` 提取后4位
- 中间用 `****` 连接
- 两种方法效果相同

预期结果:

```
stu_id  stu_name  original_phone  masked_phone
1001    张三      18612345601     186****5601
1002    李四      18612345602     186****5602
1003    王五      18612345603     186****5603
1004    赵六      18612345604     186****5604
1005    钱七      18612345605     186****5605
1006    孙八      18612345606     186****5606
1007    周九      18612345607     186****5607
1008    吴十      18612345608     186****5608
1009    郑一      18612345609     186****5609
1010    王二      18612345610     186****5610
```

3.4 日期时间函数实践

练习题1: 计算每个学生从入学到现在的天数

答案:

```sql
SELECT
    stu_id,
    stu_name,
    enroll_date,
    CURRENT_DATE() as today,
    DATEDIFF(CURRENT_DATE(), enroll_date) as days_since_enroll,
    ROUND(DATEDIFF(CURRENT_DATE(), enroll_date) / 365, 1) as years_since_enroll,
    ROUND(MONTHS_BETWEEN(CURRENT_DATE(), enroll_date), 1) as months_since_enroll,
    CASE
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date) < 365 THEN '不满1年'
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date) < 730 THEN '1-2年'
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date) < 1095 THEN '2-3年'
        ELSE '3年以上'
    END as enrollment_period
FROM students
ORDER BY days_since_enroll DESC;
```

说明:

- 使用 `DATEDIFF()` 计算日期差
- 使用 `CURRENT_DATE()` 获取当前日期
- 转换为年份和月份便于理解
- 添加分类标签

预期结果:

```
stu_id  stu_name  enroll_date  days_since_enroll  years_since_enroll
months_since_enroll  enrollment_period
1005    钱七      2021-09-01   约1263             约3.5              约41.5
  3年以上
1003    王五      2022-09-01   约898              约2.5              约29.5
  2-3年
1008    吴十      2022-09-01   约898              约2.5              约29.5
  2-3年
1001    张三      2023-09-01   约533              约1.5              约17.5
  1-2年
... (其他2023-09-01入学的学生)
```

练习题2: 统计每个月入学的学生数量

答案:

```sql
SELECT
    YEAR(enroll_date) as enroll_year,
    MONTH(enroll_date) as enroll_month,
    CONCAT(YEAR(enroll_date), '-', LPAD(MONTH(enroll_date), 2, '0')) as year_month,
    COUNT(*) as student_count,
    COLLECT_LIST(stu_name) as students
FROM students
GROUP BY YEAR(enroll_date), MONTH(enroll_date)
ORDER BY enroll_year DESC, enroll_month DESC;


-- 或者使用更详细的格式
SELECT
    DATE_FORMAT(enroll_date, 'yyyy-MM') as year_month,
    DATE_FORMAT(enroll_date, 'yyyy年MM月') as year_month_cn,
    COUNT(*) as student_count,
    COLLECT_LIST(stu_name) as students
FROM students
GROUP BY DATE_FORMAT(enroll_date, 'yyyy-MM')
ORDER BY year_month DESC;
```

说明:

- 使用 `YEAR()` 和 `MONTH()` 提取年月
- 或使用 `DATE_FORMAT()` 格式化日期
- 按年月分组统计
- 使用 `COLLECT_LIST()` 列出学生名单

预期结果:

```
enroll_year  enroll_month  year_month  student_count  students
```

```
2023        9           2023-09     7           ["张三","李四","赵六","孙八","周九
","郑一","王二"]
2022        9           2022-09     2           ["王五","吴十"]
2021        9           2021-09     1           ["钱七"]
```

练习题3: 查询入学超过1年的学生信息

答案:

```sql
SELECT
    stu_id,
    stu_name,
    gender,
    age,
    college,
    enroll_date,
    DATEDIFF(CURRENT_DATE(), enroll_date) as days_enrolled,
    ROUND(DATEDIFF(CURRENT_DATE(), enroll_date) / 365, 2) as years_enrolled
FROM students
WHERE DATEDIFF(CURRENT_DATE(), enroll_date) > 365
ORDER BY enroll_date;


-- 或使用日期计算
SELECT
    stu_id,
    stu_name,
    gender,
    age,
    college,
    enroll_date,
    CURRENT_DATE() as today,
    DATE_ADD(enroll_date, 365) as one_year_after_enroll
FROM students
WHERE enroll_date < DATE_SUB(CURRENT_DATE(), 365)
ORDER BY enroll_date;
```

说明:

- 计算入学天数是否超过365天
- 或者比较入学日期加1年是否早于当前日期
- 两种方法效果相同
- 按入学日期排序,最早的在前

预期结果:

```
stu_id  stu_name  gender  age  college      enroll_date  days_enrolled  years_enrolled
```

```
1005    钱七    男    22    计算机学院    2021-09-01    约1263    约3.46
1003    王五    男    21    物理学院    2022-09-01    约898    约2.46
1008    吴十    女    21    物理学院    2022-09-01    约898    约2.46
```

3.5 条件函数实践

练习题1: 创建一个查询,将分数转换为五级制

答案:

```sql
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 grade_level,
    CASE
        WHEN s.score >= 90 THEN 5
        WHEN s.score >= 80 THEN 4
        WHEN s.score >= 70 THEN 3
        WHEN s.score >= 60 THEN 2
        ELSE 1
    END as grade_point
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;
```

说明:

- 使用 `CASE WHEN` 实现多级分类
- 90-100: 优秀
- 80-89: 良好
- 70-79: 中等
- 60-69: 及格
- <60: 不及格
- 同时提供数值等级便于计算

预期结果:

```
stu_id  stu_name  course_id  course_name      score  grade_level  grade_point
1001    张三       C002       数据库原理       90     优秀         5
1001    张三       C001       大数据技术       85     良好         4
1001    张三       C003       Python编程       78     中等         3
1002    李四       C001       大数据技术       92     优秀         5
1002    李四       C002       数据库原理       88     良好         4
...
```

练习题2: 统计每个成绩等级的学生人数

答案:

```SQL
SELECT
    grade_level,
    COUNT(*) as record_count,
    COUNT(DISTINCT stu_id) as unique_students,
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(), 2) as percentage
FROM (
    SELECT
        stu_id,
        score,
        CASE
            WHEN score >= 90 THEN '优秀'
            WHEN score >= 80 THEN '良好'
            WHEN score >= 70 THEN '中等'
            WHEN score >= 60 THEN '及格'
            ELSE '不及格'
        END as grade_level
    FROM scores
) t
GROUP BY grade_level
ORDER BY
    CASE grade_level
        WHEN '优秀' THEN 1
        WHEN '良好' THEN 2
        WHEN '中等' THEN 3
        WHEN '及格' THEN 4
        ELSE 5
    END;


-- 或使用更简洁的方式
SELECT
    CASE
        WHEN score >= 90 THEN '优秀'
        WHEN score >= 80 THEN '良好'
        WHEN score >= 70 THEN '中等'
        WHEN score >= 60 THEN '及格'
```

```
        ELSE '不及格'
    END as grade_level,
    COUNT(*) as count,
    ROUND(AVG(score), 2) as avg_score_in_level
FROM scores
GROUP BY
    CASE
        WHEN score >= 90 THEN '优秀'
        WHEN score >= 80 THEN '良好'
        WHEN score >= 70 THEN '中等'
        WHEN score >= 60 THEN '及格'
        ELSE '不及格'
    END
ORDER BY avg_score_in_level DESC;
```

说明:

- 先使用子查询对每条记录进行等级分类
- 然后按等级分组统计
- 计算每个等级的比例
- 按等级从高到低排序

预期结果:

```
grade_level  record_count  unique_students  percentage
优秀         6             6                33.33
良好         8             7                44.44
中等         2             2                11.11
及格         2             2                11.11
不及格       0             0                0.00
```

练习题3: 查询成绩在各科平均分以上的学生

答案:

```
SQL
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 diff_from_avg,
    CASE
        WHEN s.score > AVG(s.score) OVER(PARTITION BY s.course_id) THEN '高于平均'
        WHEN s.score = AVG(s.score) OVER(PARTITION BY s.course_id) THEN '等于平均'
        ELSE '低于平均'
```

```
    END as vs_average
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 > (
    SELECT AVG(score)
    FROM scores
    WHERE course_id = s.course_id
)
ORDER BY s.course_id, s.score DESC;


-- 或使用窗口函数
SELECT *
FROM (
    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
    FROM scores s
    JOIN students st ON s.stu_id = st.stu_id
    JOIN courses c ON s.course_id = c.course_id
) t
WHERE score > course_avg
ORDER BY course_id, score DESC;
```

说明:

- 使用窗口函数计算每门课程的平均分
- 过滤出成绩高于课程平均分的记录
- 显示与平均分的差值
- 按课程和成绩排序

预期结果:

```
stu_id  stu_name  course_id  course_name    score  course_avg  diff_from_avg
vs_average
1005    钱七      C001       大数据技术     94     88.38        5.62           高于平均
1002    李四      C001       大数据技术     92     88.38        3.62           高于平均
1010    王二      C001       大数据技术     93     88.38        4.62           高于平均
1009    郑一      C001       大数据技术     90     88.38        1.62           高于平均
1007    周九      C002       数据库原理     92     88.67        3.33           高于平均
1001    张三      C002       数据库原理     90     88.67        1.33           高于平均
1004    赵六      C002       数据库原理     89     88.67        0.33           高于平均
1003    王五      C003       Python编程      95     86.75        8.25           高于平均
1005    钱七      C003       Python编程      91     86.75        4.25           高于平均
1010    王二      C003       Python编程      89     86.75        2.25           高于平均
```

3.6 聚合函数应用

练习题1: 统计每个学院的学生人数和平均年龄

答案:

```sql
SELECT
    college,
    COUNT(*) as student_count,
    ROUND(AVG(age), 1) as avg_age,
    MAX(age) as max_age,
    MIN(age) as min_age,
    COLLECT_LIST(stu_name) as students,
    COUNT(CASE WHEN gender = '男' THEN 1 END) as male_count,
    COUNT(CASE WHEN gender = '女' THEN 1 END) as female_count
FROM students
GROUP BY college
ORDER BY student_count DESC, college;
```

说明:

- 按学院分组
- 统计人数和平均年龄
- 额外提供最大最小年龄
- 统计男女比例
- 按人数降序排列

预期结果:

```
college      student_count  avg_age  max_age  min_age  students
male_count  female_count
计算机学院    4             20.3     22       19       ["张三","钱七","周九","郑一"]
   4           0
数学学院      3             19.3     20       19       ["李四","孙八","王二"]
    0           3
物理学院      2             20.5     21       20       ["王五","吴十"]
    1           1
化学学院      1             20.0     20       20       ["赵六"]
    0           1
```

练习题2: 查询平均分排名前3的学生信息

答案:

```sql
SELECT
    t.ranking,
    t.stu_id,
    st.stu_name,
    st.gender,
    st.college,
    t.course_count,
    t.avg_score,
    t.max_score,
    t.min_score,
    t.total_score
FROM (
    SELECT
        stu_id,
        COUNT(*) as course_count,
        ROUND(AVG(score), 2) as avg_score,
        MAX(score) as max_score,
        MIN(score) as min_score,
        SUM(score) as total_score,
        RANK() OVER(ORDER BY AVG(score) DESC) as ranking
    FROM scores
    GROUP BY stu_id
) t
JOIN students st ON t.stu_id = st.stu_id
WHERE t.ranking <= 3
ORDER BY t.ranking;


-- 或使用LIMIT
SELECT
    s.stu_id,
    st.stu_name,
    st.gender,
    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.gender, st.college
ORDER BY avg_score DESC
LIMIT 3;
```

说明:

- 计算每个学生的平均分
- 使用 `RANK()` 或 `ROW_NUMBER()` 进行排名
- 取前3名
- JOIN students表获取详细信息

预期结果:

```
ranking  stu_id  stu_name  gender  college      course_count  avg_score  max_score
min_score  total_score
```

```
1       1010    王二    女      数学学院    2       91.00   93      89      182
2       1005    钱七    男      计算机学院  3       90.67   94      87      272
3       1002    李四    女      数学学院    2       90.00   92      88      180
```

练习题3: 统计每门课程的及格人数和及格率

答案:

```sql
SELECT
    c.course_id,
    c.course_name,
    c.teacher,
    c.credits,
    COUNT(*) as total_students,
    SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) as pass_count,
    SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) as fail_count,
    ROUND(SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate,
    ROUND(AVG(s.score), 2) as avg_score,
    MAX(s.score) as max_score,
    MIN(s.score) as min_score,
    CASE
        WHEN SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*) = 100
THEN '全部及格'
        WHEN SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*) >= 90
THEN '及格率优秀'
        WHEN SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*) >= 80
THEN '及格率良好'
        ELSE '及格率一般'
    END as pass_rate_level
FROM courses c
JOIN scores s ON c.course_id = s.course_id
GROUP BY c.course_id, c.course_name, c.teacher, c.credits
ORDER BY pass_rate DESC, avg_score DESC;
```

说明:

- 使用 `CASE WHEN` 统计及格和不及格人数
- 计算及格率 = 及格人数 / 总人数 * 100
- 添加及格率等级分类
- 包含课程的其他统计信息
- 按及格率和平均分排序

预期结果:

```
course_id  course_name    teacher  credits  total_students  pass_count  fail_count
pass_rate  avg_score  max_score  min_score  pass_rate_level
C001       大数据技术     张教授   4        8               8           0           100.0
0     88.38      94         76         全部及格
C002       数据库原理     李教授   3        6               6           0           100.0
0     88.67      92         86         全部及格
C003       Python编程     王教授   4        4               4           0
100.00     86.75      95         78         全部及格
```

第四部分 练习题答案

4.1 复杂 WHERE 条件查询

练习题1: 查询购买金额最高的客户的所有订单

答案:

```sql
-- 方法1: 使用子查询
SELECT
    o.*,
    st.stu_name as customer_name,
    st.city
FROM orders o
JOIN students st ON o.customer_id = st.stu_id
WHERE o.customer_id = (
    SELECT customer_id
    FROM orders
    GROUP BY customer_id
    ORDER BY SUM(amount) DESC
    LIMIT 1
)
ORDER BY o.order_time;


-- 方法2: 使用窗口函数
SELECT *
FROM (
    SELECT
        o.*,
        st.stu_name as customer_name,
        SUM(o.amount) OVER(PARTITION BY o.customer_id) as total_amount,
        RANK() OVER(ORDER BY SUM(o.amount) OVER(PARTITION BY o.customer_id) DESC) as
customer_rank
    FROM orders o
    JOIN students st ON o.customer_id = st.stu_id
) t
WHERE customer_rank = 1
ORDER BY order_time;
```

```
-- 方法3: 明确查找最大金额
WITH customer_totals AS (
    SELECT
        customer_id,
        SUM(amount) as total_amount
    FROM orders
    GROUP BY customer_id
)
SELECT
    o.order_id,
    o.customer_id,
    st.stu_name as customer_name,
    o.order_time,
    o.amount,
    o.status,
    ct.total_amount as customer_total
FROM orders o
JOIN students st ON o.customer_id = st.stu_id
JOIN customer_totals ct ON o.customer_id = ct.customer_id
WHERE ct.total_amount = (SELECT MAX(total_amount) FROM customer_totals)
ORDER BY o.order_time;
```

说明:

- 三种方法都能找出消费总额最高的客户
- 方法1使用子查询找出客户ID
- 方法2使用窗口函数和排名
- 方法3使用CTE更清晰
- 显示该客户的所有订单

预期结果:

```
order_id  customer_id  customer_name  order_time           amount   status
customer_total
O001      1001         张三           2024-01-01 10:30:00  1580.50  结付完料     4780.50
O004      1001         张三           2024-01-03 09:15:00  3200.00  结付完料     4780.50
```

练习题2: 查询至少选修了3门课程的学生信息

答案:

```
SQL
SELECT
    st.*,
    course_stats.course_count,
    course_stats.avg_score,
    course_stats.courses_taken
```

```
FROM students st
JOIN (
    SELECT
        stu_id,
        COUNT(*) as course_count,
        ROUND(AVG(score), 2) as avg_score,
        COLLECT_LIST(course_id) as courses_taken,
        COLLECT_LIST(score) as scores
    FROM scores
    GROUP BY stu_id
    HAVING COUNT(*) >= 3
) course_stats ON st.stu_id = course_stats.stu_id
ORDER BY course_stats.course_count DESC, course_stats.avg_score DESC;


-- 或使用EXISTS
SELECT
    st.*,
    (SELECT COUNT(*) FROM scores WHERE stu_id = st.stu_id) as course_count,
    (SELECT ROUND(AVG(score), 2) FROM scores WHERE stu_id = st.stu_id) as avg_score
FROM students st
WHERE (
    SELECT COUNT(*)
    FROM scores s
    WHERE s.stu_id = st.stu_id
) >= 3
ORDER BY course_count DESC;
```

说明:

- 先按学生分组统计选课数
- 使用HAVING过滤出>=3门的学生
- JOIN回students表获取完整信息
- 显示选课统计信息

预期结果:

```
stu_id  stu_name  gender  age  college      enroll_date  course_count  avg_score
courses_taken
1005    钱七      男      22   计算机学院   2021-09-01   3             90.67
["C001","C002","C003"]
1001    张三      男      20   计算机学院   2023-09-01   3             84.33
["C001","C002","C003"]
```

练习题3: 查询所有科目都及格的学生

答案:

```sql
-- 方法1: 使用NOT EXISTS
SELECT
    st.*,
    course_stats.course_count,
    course_stats.avg_score,
    course_stats.min_score
FROM students st
JOIN (
    SELECT
        stu_id,
        COUNT(*) as course_count,
        ROUND(AVG(score), 2) as avg_score,
        MIN(score) as min_score
    FROM scores
    GROUP BY stu_id
) course_stats ON st.stu_id = course_stats.stu_id
WHERE NOT EXISTS (
    SELECT 1
    FROM scores s
    WHERE s.stu_id = st.stu_id AND s.score < 60
)
ORDER BY course_stats.avg_score DESC;


-- 方法2: 使用HAVING
SELECT
    st.*,
    COUNT(s.course_id) as course_count,
    ROUND(AVG(s.score), 2) as avg_score,
    MIN(s.score) as min_score
FROM students st
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.enroll_date,
st.phone, st.city
HAVING MIN(s.score) >= 60
ORDER BY avg_score DESC;


-- 方法3: 使用子查询
SELECT
    st.*,
    course_stats.*
FROM students st
JOIN (
    SELECT
        stu_id,
        COUNT(*) as course_count,
        ROUND(AVG(score), 2) as avg_score,
        MIN(score) as min_score,
        MAX(score) as max_score
    FROM scores
    GROUP BY stu_id
) course_stats ON st.stu_id = course_stats.stu_id
WHERE course_stats.min_score >= 60
ORDER BY course_stats.avg_score DESC;
```

说明:

- 三种方法都能找出没有不及格科目的学生

- 方法1使用NOT EXISTS检查是否存在不及格
- 方法2使用HAVING MIN(score) >= 60
- 方法3使用子查询过滤
- 根据数据,所有学生都及格

预期结果:

所有10个学生都符合条件(因为所有成绩都>=60),按平均分排序:

```
stu_id  stu_name  college      course_count  avg_score  min_score
1010    王二      数学学院     2             91.00      89
1005    钱七      计算机学院   3             90.67      87
1002    李四      数学学院     2             90.00      88
...
```

4.2 GROUP BY 与聚合分析

练习题1: 统计每个城市的学生数量和平均年龄

答案:

```sql
SELECT
    city,
    COUNT(*) as student_count,
    ROUND(AVG(age), 1) as avg_age,
    MAX(age) as max_age,
    MIN(age) as min_age,
    COLLECT_LIST(stu_name) as students,
    COUNT(CASE WHEN gender = '男' THEN 1 END) as male_count,
    COUNT(CASE WHEN gender = '女' THEN 1 END) as female_count,
    COLLECT_SET(college) as colleges
FROM students
GROUP BY city
ORDER BY student_count DESC, city;
```

说明:

- 按城市分组
- 统计人数、平均年龄、年龄范围
- 统计男女人数
- 列出该城市的学生和涉及的学院
- 按人数降序排列

预期结果:

```
city       student_count  avg_age  max_age  min_age  students           male_count
female_count  colleges
seoul      1              20       20       20       ["김철수"]         1           0
["컴퓨터학과"]
busan      1              19       19       19       ["이영희"]         0           1
["수학과"]
incheon    1              21       21       21       ["박민수"]         1           0
["물리학과"]
daegu      1              20       20       20       ["정수진"]         0           1
["화학과"]
gwangju    1              22       22       22       ["최동욱"]         1           0
["컴퓨터학과"]
daejeon    1              19       19       19       ["강지은"]         0           1
["수학과"]
ulsan      1              20       20       20       ["윤서준"]         1           0
["컴퓨터학과"]
suwon      1              21       21       21       ["한소희"]         0           1
["물리학과"]
changwon   1              19       19       19       ["임준영"]         1           0
["컴퓨터학과"]
jeonju     1              20       20       20       ["송미나"]         0           1
["수학과"]
```

练习题2: 查询每门课程的及格率,要求只显示及格率低于90%的课程

答案:

```SQL
SELECT
    c.course_id,
    c.course_name,
    c.teacher,
    COUNT(s.stu_id) as total_students,
    SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) as pass_count,
    SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) as fail_count,
    ROUND(SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate,
    ROUND(AVG(s.score), 2) as avg_score,
    COLLECT_LIST(
        CASE WHEN s.score < 60
        THEN CONCAT(st.stu_name, ':', s.score)
        END
    ) as failed_students
FROM courses c
JOIN scores s ON c.course_id = s.course_id
JOIN students st ON s.stu_id = st.stu_id
GROUP BY c.course_id, c.course_name, c.teacher
HAVING ROUND(SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) < 90
ORDER BY pass_rate ASC;
```

说明:

- 按课程分组统计及格率
- 使用HAVING过滤及格率<90%的课程
- 列出不及格学生名单
- 按及格率升序排列
- 根据数据,所有课程及格率都是100%,所以结果为空

预期结果:

根据提供的数据,所有课程的及格率都是100%,所以没有低于90%的课程,结果为空集。

补充查询(显示所有课程及格率):

```sql
SQL
SELECT
    c.course_id,
    c.course_name,
    COUNT(s.stu_id) as total_students,
    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,
    ROUND(AVG(s.score), 2) as avg_score
FROM courses c
JOIN scores s ON c.course_id = s.course_id
GROUP BY c.course_id, c.course_name
ORDER BY pass_rate ASC, avg_score ASC;
```

练习题3: 统计每个学生的选课数和平均分,并找出平均分前5名

答案:

```sql
SQL
-- 完整版本
SELECT
    RANK() OVER(ORDER BY avg_score DESC) as ranking,
    t.*
FROM (
    SELECT
        s.stu_id,
        st.stu_name,
        st.gender,
        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,
        SUM(s.score) as total_score,
        COLLECT_LIST(
            CONCAT(c.course_name, ':', s.score)
        ) as course_details
```

```
    FROM scores s
    JOIN students st ON s.stu_id = st.stu_id
    JOIN courses c ON s.course_id = c.course_id
    GROUP BY s.stu_id, st.stu_name, st.gender, st.college
) t
ORDER BY avg_score DESC, course_count DESC
LIMIT 5;


-- 简化版本
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
ORDER BY avg_score DESC, course_count DESC
LIMIT 5;
```

说明:

- 按学生分组统计选课数和平均分
- 使用RANK()进行排名
- 取前5名
- 显示详细的课程成绩信息
- 按平均分降序,选课数降序

预期结果:

```
ranking  stu_id  stu_name  gender  college      course_count  avg_score  max_score
min_score  total_score
1        1010    王二      女      数学学院     2             91.00      93         89
182
2        1005    钱七      男      计算机学院   3             90.67      94         87
272
3        1002    李四      女      数学学院     2             90.00      92         88
180
4        1007    周九      男      计算机学院   1             92.00      92         92
92
5        1004    赵六      女      化学学院     1             89.00      89         89
89
```

4.3 ORDER BY 与排序

练习题1: 查询每门课程成绩最高的3名学生

答案:

```sql
SELECT *
FROM (
    SELECT
        s.stu_id,
        st.stu_name,
        s.course_id,
        c.course_name,
        s.score,
        ROW_NUMBER() OVER(PARTITION BY s.course_id ORDER BY s.score DESC, s.stu_id) as
row_num,
        RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) as rank_num,
        DENSE_RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) as
dense_rank_num
    FROM scores s
    JOIN students st ON s.stu_id = st.stu_id
    JOIN courses c ON s.course_id = c.course_id
) t
WHERE row_num <= 3
ORDER BY course_id, score DESC;


-- 或者分别查询每门课程
SELECT
    s.course_id,
    c.course_name,
    s.stu_id,
    st.stu_name,
    s.score,
    RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) as ranking
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
JOIN courses c ON s.course_id = c.course_id
QUALIFY RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) <= 3
ORDER BY s.course_id, s.score DESC;
```

说明:

- 使用窗口函数按课程分区排名
- ROW_NUMBER()给每行唯一编号
- RANK()相同分数排名相同,下一名跳号
- DENSE_RANK()相同分数排名相同,下一名不跳号
- 取每门课程前3名

预期结果:

```
stu_id  stu_name  course_id  course_name    score  row_num  rank_num  dense_rank_num
1005    钱七       C001       大数据技术     94     1        1         1
1010    王二       C001       大数据技术     93     2        2         2
```

```
1002    李四    C001    大数据技术    92    3    3    3

1007    周九    C002    数据库原理    92    1    1    1
1001    张三    C002    数据库原理    90    2    2    2
1004    赵六    C002    数据库原理    89    3    3    3

1003    王五    C003    Python编程    95    1    1    1
1005    钱七    C003    Python编程    91    2    2    2
1010    王二    C003    Python编程    89    3    3    3
```

练习题2: 按学院和平均分排序,显示前10名学生

答案:

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

说明:

- 先按学院排序,再按平均分降序
- 计算总体排名和学院内排名
- 显示选课统计信息
- 取前10名

预期结果:

```
college      stu_id  stu_name  gender  age  course_count  avg_score  max_score
min_score  overall_rank  college_rank
계산机학과   1005    钱七      男      22   3             90.67      94         87
1            1
```

```
계산기학과   1001   张三    男    20   3          84.33    90        78
2            2
계산기학과   1007   周九    男    20   1          92.00    92        92
3            3
계산기학과   1009   郑一    男    19   2          88.00    90        86
4            4
수학학과     1010   王二    女    20   2          91.00    93        89
5            1
수학학과     1002   李四    女    19   2          90.00    92        88
6            2
수학학과     1006   孙八    女    19   1          88.00    88        88
7            3
물理학과     1003   王五    男    21   2          85.50    95        76
8            1
물리학과     1008   吴十    女    21   1          85.00    85        85
9            2
화학과       1004   赵六    女    20   1          89.00    89        89
10           1
```

练习题3: 查询每个学生的排名(按平均分)

答案:

```sql
SQL
SELECT
    ROW_NUMBER() OVER(ORDER BY avg_score DESC, stu_id) as row_number,
    RANK() OVER(ORDER BY avg_score DESC) as rank_num,
    DENSE_RANK() OVER(ORDER BY avg_score DESC) as dense_rank_num,
    NTILE(4) OVER(ORDER BY avg_score DESC) as quartile,
    t.*
FROM (
    SELECT
        s.stu_id,
        st.stu_name,
        st.gender,
        st.college,
        st.city,
        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.gender, st.college, st.city
) t
ORDER BY avg_score DESC, stu_id;


-- 添加百分位排名
SELECT
    stu_id,
    stu_name,
    college,
    avg_score,
```

```
    RANK() OVER(ORDER BY avg_score DESC) as rank_num,
    PERCENT_RANK() OVER(ORDER BY avg_score DESC) as percent_rank,
    ROUND(PERCENT_RANK() OVER(ORDER BY avg_score DESC) * 100, 2) as percentile
FROM (
    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
) t
ORDER BY avg_score DESC;
```

说明:

- ROW_NUMBER(): 连续唯一排名
- RANK(): 相同分数同名次,跳号
- DENSE_RANK(): 相同分数同名次,不跳号
- NTILE(4): 分成4个四分位组
- PERCENT_RANK(): 百分位排名

预期结果:

```
row_number  rank_num  dense_rank_num  quartile  stu_id  stu_name  college       avg_score
1           1         1               1         1007    周九      计算机学院    92.00
2           1         1               1         1010    王二      数学学院      91.00
3           3         2               1         1005    钱七      计算机学院    90.67
4           4         3               2         1002    李四      数学学院      90.00
5           5         4               2         1004    赵六      化学学院      89.00
6           6         5               2         1009    郑一      计算机学院    88.00
7           6         5               3         1006    孙八      数学学院      88.00
8           8         6               3         1003    王五      物理学院      85.50
9           9         7               4         1008    吴十      物理学院      85.00
10          10        8               4         1001    张三      계산기학과    84.33
```

4.4 JOIN 多表关联

练习题1: 查询选修了"大数据技术"课程的所有学生信息

答案:

```sql
SELECT DISTINCT
    st.*,
    s.score,
    s.exam_date,
    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 grade_level
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
JOIN courses c ON s.course_id = c.course_id
WHERE c.course_name = '大数据技术'
ORDER BY s.score DESC;


-- 或使用子查询
SELECT
    st.*,
    (SELECT score FROM scores WHERE stu_id = st.stu_id AND course_id = 'C001') as score
FROM students st
WHERE st.stu_id IN (
    SELECT DISTINCT stu_id
    FROM scores s
    JOIN courses c ON s.course_id = c.course_id
    WHERE c.course_name = '大数据技术'
)
ORDER BY score DESC;
```

说明:

- JOIN三个表关联查询
- 通过课程名称过滤
- 显示学生完整信息和成绩
- 添加成绩等级
- 按成绩降序排列

预期结果:

```
stu_id  stu_name  gender  age  college      enroll_date  phone        city     score
exam_date   grade_level
1005    钱七      男      22   计算机学院   2021-09-01   18612345605  杭州     94
2024-01-15  优秀
1010    王二      女      20   数学学院     2023-09-01   18612345610  天津     93
2024-01-15  优秀
1002    李四      女      19   数学学院     2023-09-01   18612345602  上海     92
2024-01-15  优秀
1009    郑一      男      19   计算机学院   2023-09-01   18612345609  西安     90
2024-01-15  优秀
1006    孙八      女      19   数学学院     2023-09-01   18612345606  南京     88
```

```
2024-01-15  良好
1001    张三      男      20   计算机学院   2023-09-01   18612345601  北京     85
2024-01-15  良好
1003    王五      男      21   物理学院     2022-09-01   18612345603  广州     76
2024-01-15  中等
```

练习题2: 查询没有考试记录的学生

答案:

```sql
-- 方法1: 使用LEFT JOIN和NULL检查
SELECT
    st.*
FROM students st
LEFT JOIN scores s ON st.stu_id = s.stu_id
WHERE s.stu_id IS NULL;


-- 方法2: 使用NOT IN
SELECT *
FROM students
WHERE stu_id NOT IN (
    SELECT DISTINCT stu_id
    FROM scores
);


-- 方法3: 使用NOT EXISTS
SELECT *
FROM students st
WHERE NOT EXISTS (
    SELECT 1
    FROM scores s
    WHERE s.stu_id = st.stu_id
);


-- 附加: 统计每个学生的考试记录数
SELECT
    st.stu_id,
    st.stu_name,
    st.college,
    COUNT(s.course_id) as exam_count,
    CASE
        WHEN COUNT(s.course_id) = 0 THEN '无考试记录'
        WHEN COUNT(s.course_id) < 3 THEN '考试较少'
        ELSE '正常'
    END as exam_status
FROM students st
LEFT JOIN scores s ON st.stu_id = s.stu_id
GROUP BY st.stu_id, st.stu_name, st.college
ORDER BY exam_count ASC, st.stu_id;
```

说明:

- 三种方法都能找出没有考试记录的学生
- 方法1使用LEFT JOIN保留所有学生,然后过滤NULL
- 方法2使用NOT IN排除有记录的学生
- 方法3使用NOT EXISTS检查不存在的记录
- 根据数据,所有学生都有考试记录

预期结果:

根据提供的数据,所有学生都至少有一门考试记录,所以结果为空集。

补充结果(考试统计):

```
stu_id  stu_name  college      exam_count  exam_status
1004    赵六      化学学院     1           考试较少
1006    孙八      数学学院     1           考试较少
1007    周九      计算机学院   1           考试较少
1008    吴十      物理学院     1           考试较少
1002    李四      数学学院     2           考试较少
1003    王五      物理学院     2           考试较少
1009    郑一      计算机学院   2           考试较少
1010    王二      数学学院     2           考试较少
1001    张三      计算机学院   3           正常
1005    钱七      计算机学院   3           正常
```

练习题3: 查询每个学院各门课程的平均分

答案:

```SQL
SELECT
    st.college,
    c.course_id,
    c.course_name,
    COUNT(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,
    ROUND(STDDEV(s.score), 2) as stddev_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, c.course_id, c.course_name
ORDER BY st.college, avg_score DESC;


-- 使用WITH CUBE进行多维分析
SELECT
    st.college,
    c.course_id,
    c.course_name,
    COUNT(s.stu_id) as student_count,
    ROUND(AVG(s.score), 2) as avg_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_id, c.course_name WITH CUBE
ORDER BY st.college NULLS LAST, c.course_id NULLS LAST;
```

说明:

- 按学院和课程进行二维分组
- 统计每个组合的平均分和其他指标
- 计算优秀率和及格率
- WITH CUBE可以生成所有维度的汇总

预期结果:

```
college      course_id  course_name    student_count  avg_score  max_score  min_score
excellent_count  pass_count  pass_rate
계산기학과   C001       大数据技术     4              89.75      94         85         3
4           100.00
계산기학과   C002       大数据技术     3              89.67      92         87         2
3           100.00
계산기학과   C003       Python编程     2              84.50      91         78         1
2           100.00
수학학과     C001       大数据技术     3              91.00      93         88         3
3           100.00
수학학과     C002       数据库原理     1              88.00      88         88         0
1           100.00
수학학과     C003       Python编程     1              89.00      89         89         0
1           100.00
물리학과     C001       大数据技术     1              76.00      76         76         0
1           100.00
물리학과     C003       Python编程     1              95.00      95         95         1
1           100.00
화학학과     C002       数据库原理     1              89.00      89         89         0
1           100.00
```

4.5 子查询应用

练习题1: 查询平均分最高的学生的详细信息

答案:

```sql
-- 方法1: 使用子查询
SELECT
    st.*,
    avg_scores.course_count,
    avg_scores.avg_score,
    avg_scores.max_score,
    avg_scores.min_score,
    avg_scores.total_score
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,
        SUM(score) as total_score
    FROM scores
    GROUP BY stu_id
) avg_scores ON st.stu_id = avg_scores.stu_id
WHERE avg_scores.avg_score = (
    SELECT MAX(avg_score)
    FROM (
        SELECT ROUND(AVG(score), 2) as avg_score
        FROM scores
        GROUP BY stu_id
    ) t
)
ORDER BY avg_scores.avg_score DESC;


-- 方法2: 使用窗口函数
SELECT *
FROM (
    SELECT
        st.*,
        avg_scores.course_count,
        avg_scores.avg_score,
        avg_scores.max_score,
        avg_scores.min_score,
        RANK() OVER(ORDER BY avg_scores.avg_score DESC) as ranking
    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
    ) avg_scores ON st.stu_id = avg_scores.stu_id
) t
WHERE ranking = 1;
```

```
-- 方法3: 使用WITH CTE
WITH student_averages AS (
    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
),
max_avg AS (
    SELECT MAX(avg_score) as max_avg_score
    FROM student_averages
)
SELECT
    st.*,
    sa.course_count,
    sa.avg_score,
    sa.max_score,
    sa.min_score
FROM students st
JOIN student_averages sa ON st.stu_id = sa.stu_id
CROSS JOIN max_avg
WHERE sa.avg_score = max_avg.max_avg_score;
```

说明:

- 三种方法都能找出平均分最高的学生
- 方法1使用嵌套子查询
- 方法2使用窗口函数排名
- 方法3使用CTE更清晰
- 可能有多个学生并列最高

预期结果:

```
stu_id  stu_name  gender  age  college      enroll_date  phone        city
course_count  avg_score  max_score  min_score
1007    周九      男      20   计算机学院   2023-09-01   18612345607  武汉   1
92.00      92         92
```

注意: 如果1007和1010平均分相同,两个都会返回

练习题2: 查询每门课程中成绩排名前3的学生

答案:

```sql
SELECT
    course_id,
    course_name,
    stu_id,
    stu_name,
    score,
    ranking
FROM (
    SELECT
        s.course_id,
        c.course_name,
        s.stu_id,
        st.stu_name,
        s.score,
        RANK() OVER(PARTITION BY s.course_id ORDER BY s.score DESC) as ranking,
        ROW_NUMBER() OVER(PARTITION BY s.course_id ORDER BY s.score DESC, s.stu_id) as
row_num
    FROM scores s
    JOIN students st ON s.stu_id = st.stu_id
    JOIN courses c ON s.course_id = c.course_id
) t
WHERE ranking <= 3
ORDER BY course_id, score DESC, stu_id;


-- 或使用相关子查询
SELECT
    s.course_id,
    c.course_name,
    s.stu_id,
    st.stu_name,
    s.score,
    (
        SELECT COUNT(DISTINCT score)
        FROM scores
        WHERE course_id = s.course_id AND score > s.score
    ) + 1 as ranking
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
JOIN courses c ON s.course_id = c.course_id
WHERE (
    SELECT COUNT(DISTINCT score)
    FROM scores
    WHERE course_id = s.course_id AND score > s.score
) < 3
ORDER BY s.course_id, s.score DESC;
```

说明:

- 使用窗口函数按课程分区排名
- RANK()处理并列排名
- ROW_NUMBER()给出唯一序号
- 取每门课程前3名
- 按课程和成绩排序

预期结果:

```
course_id  course_name    stu_id  stu_name  score  ranking
C001       大数据技术     1005    钱七      94     1
C001       大数据技术     1010    王二      93     2
C001       大数据技术     1002    李四      92     3
C002       数据库原理     1007    周九      92     1
C002       数据库原理     1001    张三      90     2
C002       数据库原理     1004    赵六      89     3
C003       Python编程     1003    王五      95     1
C003       Python编程     1005    钱七      91     2
C003       Python编程     1010    王二      89     3
```

练习题3: 查询同时选修了C001和C002课程的学生

答案:

```sql
-- 方法1: 使用INTERSECT(如果Hive支持)
SELECT stu_id
FROM scores
WHERE course_id = 'C001'
INTERSECT
SELECT stu_id
FROM scores
WHERE course_id = 'C002';


-- 方法2: 使用子查询和IN
SELECT DISTINCT
    st.*,
    s1.score as c001_score,
    s2.score as c002_score
FROM students st
JOIN scores s1 ON st.stu_id = s1.stu_id AND s1.course_id = 'C001'
JOIN scores s2 ON st.stu_id = s2.stu_id AND s2.course_id = 'C002'
ORDER BY st.stu_id;


-- 方法3: 使用HAVING COUNT
SELECT
    st.*,
    COUNT(DISTINCT s.course_id) as matched_courses,
    COLLECT_LIST(CONCAT(s.course_id, ':', s.score)) as course_scores
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
WHERE s.course_id IN ('C001', 'C002')
GROUP BY st.stu_id, st.stu_name, st.gender, st.age, st.college, st.enroll_date,
st.phone, st.city
HAVING COUNT(DISTINCT s.course_id) = 2
ORDER BY st.stu_id;


-- 方法4: 使用EXISTS
SELECT *
```

```
FROM students st
WHERE EXISTS (
    SELECT 1 FROM scores WHERE stu_id = st.stu_id AND course_id = 'C001'
)
AND EXISTS (
    SELECT 1 FROM scores WHERE stu_id = st.stu_id AND course_id = 'C002'
)
ORDER BY stu_id;
```

说明:

- 四种方法都能找出同时选修两门课的学生
- 方法2通过两次JOIN确保两门课都存在
- 方法3使用HAVING COUNT确认课程数量
- 方法4使用EXISTS检查两门课程都存在
- 显示两门课程的成绩

预期结果:

```
stu_id  stu_name  gender  age  college      enroll_date  phone        city  c001_score
c002_score
1001    张三      男      20   计算机学院   2023-09-01   18612345601  북경  85
90
1002    李四      女      19   数学学院     2023-09-01   18612345602  상해  92
88
1005    钱七      男      22   계산기학과   2021-09-01   18612345605  항주  94
87
1007    周九      男      20   계산기학과   2023-09-01   18612345607  무한  (NULL)
92
1009    郑一      男      19   계산기학과   2023-09-01   18612345609  서안  90
86
```

综合练习题答案

练习 1: 学생 성적 관리 시스템

요구사항 1: 뷰 생성

답안:

```SQL
-- 학생 성적 종합 뷰 생성
CREATE VIEW student_performance_view AS
SELECT
    st.stu_id,
```

```
    st.stu_name,
    st.gender,
    st.college,
    COUNT(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(s.score) as total_score,
    COLLECT_LIST(s.course_id) as courses_taken,
    COLLECT_LIST(s.score) as all_scores
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.college;


-- 뷰 조회
SELECT * FROM student_performance_view ORDER BY avg_score DESC;


-- 뷰 삭제 (필요시)
-- DROP VIEW IF EXISTS student_performance_view;
```

요구사항 2: 평균 점수 상위 5명 조회

답안:

```SQL
SELECT
    spv.*,
    RANK() OVER(ORDER BY spv.avg_score DESC) as ranking
FROM student_performance_view spv
WHERE spv.course_count > 0  -- 수강 기록이 있는 학생만
ORDER BY spv.avg_score DESC
LIMIT 5;


-- 또는 원본 테이블에서 직접 조회
SELECT
    st.stu_id,
    st.stu_name,
    st.gender,
    st.age,
    st.college,
    st.city,
    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 students st
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
ORDER BY avg_score DESC
LIMIT 5;
```

요구사항 3: 각 학과의 우수율 통계

답안:

```sql
SELECT
    st.college,
    COUNT(DISTINCT st.stu_id) as total_students,
    COUNT(DISTINCT CASE WHEN avg_scores.avg_score >= 90 THEN st.stu_id END) as
excellent_students,
    ROUND(
        COUNT(DISTINCT CASE WHEN avg_scores.avg_score >= 90 THEN st.stu_id END) * 100.0
/
        COUNT(DISTINCT st.stu_id),
        2
    ) as excellent_rate,
    ROUND(AVG(avg_scores.avg_score), 2) as college_avg_score,
    MAX(avg_scores.avg_score) as max_avg_score,
    MIN(avg_scores.avg_score) as min_avg_score
FROM students st
LEFT JOIN (
    SELECT
        stu_id,
        ROUND(AVG(score), 2) as avg_score
    FROM scores
    GROUP BY stu_id
) avg_scores ON st.stu_id = avg_scores.stu_id
GROUP BY st.college
ORDER BY excellent_rate DESC, college_avg_score DESC;
```

요구사항 4: 불합격 학생 명단 출력

답안:

```sql
-- 불합격 학생 조회
SELECT DISTINCT
    st.stu_id,
    st.stu_name,
    st.gender,
    st.college,
    st.phone,
    s.course_id,
    c.course_name,
    s.score,
    s.exam_date
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
JOIN courses c ON s.course_id = c.course_id
WHERE s.score < 60
```

```
ORDER BY st.stu_id, s.course_id;


-- 로컬 파일로 출력
INSERT OVERWRITE LOCAL DIRECTORY '/usr/datas/unit3/output/failed_students'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
SELECT
    st.stu_id,
    st.stu_name,
    st.gender,
    st.college,
    st.phone,
    s.course_id,
    c.course_name,
    s.score,
    '불합격' as status,
    CURRENT_DATE() as report_date
FROM students st
JOIN scores s ON st.stu_id = s.stu_id
JOIN courses c ON s.course_id = c.course_id
WHERE s.score < 60
ORDER BY st.stu_id, s.course_id;


-- 참고: 제공된 데이터에는 불합격 학생이 없으므로 결과는 빈 파일
```

练습 2: 과목 수강 분석

요구사항 1: 각 과목의 수강 인원, 평균 점수, 합격률

답안:

```
SQL
SELECT
    c.course_id,
    c.course_name,
    c.teacher,
    c.credits,
    COUNT(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,
    ROUND(STDDEV(s.score), 2) as stddev_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 >= 60 AND s.score < 80 THEN 1 ELSE 0 END) as pass_count,
    SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) as fail_count,
    ROUND(SUM(CASE WHEN s.score >= 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pass_rate,
    ROUND(SUM(CASE WHEN s.score >= 90 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
excellent_rate
FROM courses c
JOIN scores s ON c.course_id = s.course_id
```

```
GROUP BY c.course_id, c.course_name, c.teacher, c.credits
ORDER BY student_count DESC, avg_score DESC;
```

요구사항 2: 가장 인기 있는 3개 과목

답안:

```sql
SELECT
    c.course_id,
    c.course_name,
    c.teacher,
    c.credits,
    COUNT(s.stu_id) as student_count,
    ROUND(AVG(s.score), 2) as avg_score,
    RANK() OVER(ORDER BY COUNT(s.stu_id) DESC) as popularity_rank
FROM courses c
JOIN scores s ON c.course_id = s.course_id
GROUP BY c.course_id, c.course_name, c.teacher, c.credits
ORDER BY student_count DESC
LIMIT 3;
```

예상 결과:

```
course_id  course_name  teacher  credits  student_count  avg_score  popularity_rank
C001       빅데이터기술  김교수   4        8              88.38      1
C002       데이터베이스  이교수   3        6              88.67      2
C003       파이썬프로그래밍 박교수   4        4              86.75      3
```

요구사항 3: 가장 어려운 3개 과목

답안:

```sql
SELECT
    c.course_id,
    c.course_name,
    c.teacher,
    COUNT(s.stu_id) as student_count,
    ROUND(AVG(s.score), 2) as avg_score,
    ROUND(STDDEV(s.score), 2) as stddev_score,
    MIN(s.score) as min_score,
    ROUND(SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
fail_rate,
```

```
    RANK() OVER(ORDER BY AVG(s.score) ASC) as difficulty_rank
FROM courses c
JOIN scores s ON c.course_id = s.course_id
GROUP BY c.course_id, c.course_name, c.teacher
ORDER BY avg_score ASC
LIMIT 3;
```

예상 결과:

```
course_id  course_name      teacher  student_count  avg_score  stddev_score  min_score
fail_rate  difficulty_rank
C003       Python编程        王教授   4              86.75      7.27          78
0.00       1
C001       大数据技术        张教授   8              88.38      5.68          76
0.00       2
C002       数据库原理        李教授   6              88.67      2.16          86
0.00       3
```

요구사항 4: 각 과목의 점수 분포

답안:

```SQL
SELECT
    c.course_id,
    c.course_name,
    SUM(CASE WHEN s.score >= 90 THEN 1 ELSE 0 END) as score_90_100,
    SUM(CASE WHEN s.score >= 80 AND s.score < 90 THEN 1 ELSE 0 END) as score_80_89,
    SUM(CASE WHEN s.score >= 70 AND s.score < 80 THEN 1 ELSE 0 END) as score_70_79,
    SUM(CASE WHEN s.score >= 60 AND s.score < 70 THEN 1 ELSE 0 END) as score_60_69,
    SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) as score_below_60,
    COUNT(*) as total_students,
    ROUND(SUM(CASE WHEN s.score >= 90 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pct_90_100,
    ROUND(SUM(CASE WHEN s.score >= 80 AND s.score < 90 THEN 1 ELSE 0 END) * 100.0 /
COUNT(*), 2) as pct_80_89,
    ROUND(SUM(CASE WHEN s.score >= 70 AND s.score < 80 THEN 1 ELSE 0 END) * 100.0 /
COUNT(*), 2) as pct_70_79,
    ROUND(SUM(CASE WHEN s.score >= 60 AND s.score < 70 THEN 1 ELSE 0 END) * 100.0 /
COUNT(*), 2) as pct_60_69,
    ROUND(SUM(CASE WHEN s.score < 60 THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) as
pct_below_60
FROM courses c
JOIN scores s ON c.course_id = s.course_id
GROUP BY c.course_id, c.course_name
ORDER BY c.course_id;


-- 더 상세한 분포 (10점 단위)
SELECT
```

```
    c.course_id,
    c.course_name,
    CONCAT(FLOOR(s.score/10)*10, '-', FLOOR(s.score/10)*10+9) as score_range,
    COUNT(*) as student_count,
    COLLECT_LIST(CONCAT(st.stu_name, ':', s.score)) as students_in_range
FROM courses c
JOIN scores s ON c.course_id = s.course_id
JOIN students st ON s.stu_id = st.stu_id
GROUP BY c.course_id, c.course_name, FLOOR(s.score/10)
ORDER BY c.course_id, FLOOR(s.score/10) DESC;
```

练习 3: 데이터 품질 검사

요구사항 1: 시험 기록이 없는 학생 확인

답안:

```SQL
-- 시험 기록이 없는 학생 조회
SELECT
    st.*,
    '시험기록없음' as status
FROM students st
LEFT JOIN scores s ON st.stu_id = s.stu_id
WHERE s.stu_id IS NULL;


-- 모든 학생의 시험 기록 현황
SELECT
    st.stu_id,
    st.stu_name,
    st.college,
    COUNT(s.course_id) as exam_count,
    CASE
        WHEN COUNT(s.course_id) = 0 THEN '시험기록없음'
        WHEN COUNT(s.course_id) < 2 THEN '시험기록부족'
        WHEN COUNT(s.course_id) < 3 THEN '시험기록보통'
        ELSE '시험기록충분'
    END as exam_status,
    COLLECT_LIST(s.course_id) as courses_taken
FROM students st
LEFT JOIN scores s ON st.stu_id = s.stu_id
GROUP BY st.stu_id, st.stu_name, st.college
ORDER BY exam_count ASC, st.stu_id;
```

요구사항 2: 학생이 없는 과목 확인

답안:

```sql
-- 학생이 없는 과목 조회
SELECT
    c.*,
    '수강학생없음' as status
FROM courses c
LEFT JOIN scores s ON c.course_id = s.course_id
WHERE s.course_id IS NULL;


-- 모든 과목의 수강 현황
SELECT
    c.course_id,
    c.course_name,
    c.teacher,
    c.credits,
    COUNT(s.stu_id) as student_count,
    CASE
        WHEN COUNT(s.stu_id) = 0 THEN '수강학생없음'
        WHEN COUNT(s.stu_id) < 3 THEN '수강학생부족'
        WHEN COUNT(s.stu_id) < 6 THEN '수강학생보통'
        ELSE '수강학생충분'
    END as enrollment_status
FROM courses c
LEFT JOIN scores s ON c.course_id = s.course_id
GROUP BY c.course_id, c.course_name, c.teacher, c.credits
ORDER BY student_count ASC;
```

요구사항 3: 이상 성적 확인

답안:

```sql
-- 이상 성적 조회 (음수 또는 100점 초과)
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score,
    s.exam_date,
    CASE
        WHEN s.score < 0 THEN '음수점수'
        WHEN s.score > 100 THEN '100점초과'
        ELSE '정상'
    END as score_status,
    '이상데이터' as alert_level
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 < 0 OR s.score > 100;
```

```
-- 성적 데이터 품질 종합 리포트
SELECT
    '성적데이터품질검사' as check_type,
    COUNT(*) as total_records,
    SUM(CASE WHEN score IS NULL THEN 1 ELSE 0 END) as null_count,
    SUM(CASE WHEN score < 0 THEN 1 ELSE 0 END) as negative_count,
    SUM(CASE WHEN score > 100 THEN 1 ELSE 0 END) as over_100_count,
    SUM(CASE WHEN score >= 0 AND score <= 100 THEN 1 ELSE 0 END) as valid_count,
    MIN(score) as min_score,
    MAX(score) as max_score,
    ROUND(AVG(score), 2) as avg_score
FROM scores;
```

요구사항 4: 각 테이블의 레코드 수와 NULL 값 현황

답안:

```
SQL
-- 학생 테이블 통계
SELECT
    'students' as table_name,
    COUNT(*) as total_records,
    SUM(CASE WHEN stu_id IS NULL THEN 1 ELSE 0 END) as null_stu_id,
    SUM(CASE WHEN stu_name IS NULL THEN 1 ELSE 0 END) as null_stu_name,
    SUM(CASE WHEN gender IS NULL THEN 1 ELSE 0 END) as null_gender,
    SUM(CASE WHEN age IS NULL THEN 1 ELSE 0 END) as null_age,
    SUM(CASE WHEN college IS NULL THEN 1 ELSE 0 END) as null_college,
    SUM(CASE WHEN enroll_date IS NULL THEN 1 ELSE 0 END) as null_enroll_date,
    SUM(CASE WHEN phone IS NULL THEN 1 ELSE 0 END) as null_phone,
    SUM(CASE WHEN city IS NULL THEN 1 ELSE 0 END) as null_city
FROM students

UNION ALL

-- 성적 테이블 통계
SELECT
    'scores' as table_name,
    COUNT(*) as total_records,
    SUM(CASE WHEN stu_id IS NULL THEN 1 ELSE 0 END) as null_stu_id,
    SUM(CASE WHEN course_id IS NULL THEN 1 ELSE 0 END) as null_course_id,
    SUM(CASE WHEN score IS NULL THEN 1 ELSE 0 END) as null_score,
    SUM(CASE WHEN exam_date IS NULL THEN 1 ELSE 0 END) as null_exam_date,
    0 as null_col5,
    0 as null_col6,
    0 as null_col7,
    0 as null_col8
FROM scores

UNION ALL
```

```
-- 과목 테이블 통계
SELECT
    'courses' as table_name,
    COUNT(*) as total_records,
    SUM(CASE WHEN course_id IS NULL THEN 1 ELSE 0 END) as null_course_id,
    SUM(CASE WHEN course_name IS NULL THEN 1 ELSE 0 END) as null_course_name,
    SUM(CASE WHEN credits IS NULL THEN 1 ELSE 0 END) as null_credits,
    SUM(CASE WHEN teacher IS NULL THEN 1 ELSE 0 END) as null_teacher,
    0 as null_col5,
    0 as null_col6,
    0 as null_col7,
    0 as null_col8
FROM courses


UNION ALL


-- 주문 테이블 통계
SELECT
    'orders' as table_name,
    COUNT(*) as total_records,
    SUM(CASE WHEN order_id IS NULL THEN 1 ELSE 0 END) as null_order_id,
    SUM(CASE WHEN customer_id IS NULL THEN 1 ELSE 0 END) as null_customer_id,
    SUM(CASE WHEN order_time IS NULL THEN 1 ELSE 0 END) as null_order_time,
    SUM(CASE WHEN amount IS NULL THEN 1 ELSE 0 END) as null_amount,
    SUM(CASE WHEN status IS NULL THEN 1 ELSE 0 END) as null_status,
    0 as null_col6,
    0 as null_col7,
    0 as null_col8
FROM orders;


-- 종합 데이터 품질 리포트
SELECT
    'unit3db' as database_name,
    CURRENT_DATE() as report_date,
    (SELECT COUNT(*) FROM students) as students_count,
    (SELECT COUNT(*) FROM scores) as scores_count,
    (SELECT COUNT(*) FROM courses) as courses_count,
    (SELECT COUNT(*) FROM orders) as orders_count,
    (SELECT COUNT(DISTINCT stu_id) FROM students) as unique_students,
    (SELECT COUNT(DISTINCT course_id) FROM courses) as unique_courses;
```

总结

本答案文档包含了Hive第3章综合实践工作手册中所有练习题的详细解答。

特点:

- 每道题提供多种解法
- 包含详细的SQL语句和说明
- 提供预期结果示例
- 注重实践性和可操作性

使用建议:

学生应先独立完成练习题

遇到困难时再参考答案

理解每种解法的优缺点

在实际环境中验证结果

尝试优化和改进SQL语句

注意事项:

- 答案基于提供的示例数据
- 实际结果可能因数据不同而异
- 某些查询在不同Hive版本可能需要调整
- 建议在测试环境中先验证后再在生产环境使用

教师使用说明:

- 本答案可作为教学参考
- 建议根据实际教学情况调整
- 可以此为基础设计更多练习题
- 鼓励学生探索多种解法

祝学习顺利!