LEFT: KR RIGHT: CN

Hive 제 3 장 종합 실습 워크북

과정 정보

- 실습 시간: 4 시간
- 데이터베이스명: unit3db
- 데이터 디렉토리: /usr/datas/unit3
- MySQL 정보: 사용자명: hiveuser, 비밀번호: 123456
- Beeline 연결: `hive-bee` 명령어 사용

목차

제 1 부: 환경 준비 및 기초 작업 (30 분)

1.1 데이터베이스 생성 및 관리

1.2 실습 데이터셋 준비

1.3 데이터 로딩 검증

제 2 부: 데이터 타입 및 테이블 조작 실습 (60 분)

2.1 복합 데이터 타입 실습

2.2 내부 테이블과 외부 테이블 비교

2.3 파티션 테이블 실습

제 3 부: 연산자 및 함수 종합 실습 (90 분)

3.1 산술 및 비교 연산자

3.2 논리 연산자 응용

3.3 문자열 함수 실습

3.4 날짜/시간 함수 실습

3.5 조건 함수 실습

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

LEFT: KR RIGHT: CN

3.6 집계 함수 응용 3.6 聚合函数应用

제 4 부: 쿼리 및 분석 종합 실습 (60 분) 第四部分:查询与分析综合实践 (60 分钟)

4.1 복잡한 WHERE 조건 쿼리 4.1 复杂 WHERE 条件查询

4.2 GROUP BY 와 집계 분석 4.2 GROUP BY 与聚合分析

4.3 ORDER BY 와 정렬 4.3 ORDER BY 与排序

4.4 JOIN 다중 테이블 연관 4.4 JOIN 多表关联

4.5 서브쿼리 응용 4.5 子查询应用

제 5 부: 데이터 입출력 실습 (30 분) 第五部分:数据导入导出实践 (30 分钟)

5.1 다양한 데이터 입력 방식 5.1 多种数据导入方式

5.2 데이터 출력 실습 5.2 数据导出实践

5.3 종합 사례 5.3 综合案例

제 1 부: 환경 준비 및 기초 작업 (30 분) 第一部分:环境准备与基础操作 (30 分钟)

1.1 데이터베이스 생성 및 관리 1.1 数据库创建与管理

Step 1: Beeline 연결 步骤 1: 连接 Beeline

```bash
hive-bee
```

Step 2: 실습용 데이터베이스 생성 步骤 2: 创建实践数据库

```sql
-- 데이터베이스 생성
CREATE DATABASE IF NOT EXISTS unit3db
COMMENT '제 3 장 종합 실습 데이터베이스'
LOCATION '/user/hive/unit3db';

-- 데이터베이스 목록 조회
SHOW DATABASES;

-- 데이터베이스 사용
USE unit3db;
```

```sql
-- 创建数据库
CREATE DATABASE IF NOT EXISTS unit3db;

-- 查看数据库
SHOW DATABASES;

-- 使用数据库
USE unit3db;
```

LEFT: KR

```
-- 현재 데이터베이스 확인
SELECT current_database();
```

검증: 데이터베이스가 성공적으로 생성되었고, 현재 unit3db 를 사용 중인지 확인

1.2 실습 데이터셋 준비

데이터 디렉토리 생성

```bash
# 새 터미널 창에서 실행
sudo mkdir -p /usr/datas/unit3
sudo chmod 777 /usr/datas/unit3
cd /usr/datas/unit3
```

데이터셋 1: 학생 정보 테이블 (students.txt)

```bash
cat > /usr/datas/unit3/students.txt << 'EOF'
1001    김철수  남      20      컴퓨터학과      2023-09-01      18612345601     seoul
1002    이영희  여      19      수학과  2023-09-01      18612345602     busan
1003    박민수  남      21      물리학과        2022-09-01      18612345603     incheon
1004    정수진  여      20      화학과  2023-09-01      18612345604     daegu
1005    최동욱  남      22      컴퓨터학과      2021-09-01      18612345605     gwangju
1006    강지은  여      19      수학과  2023-09-01      18612345606     daejeon
1007    윤서준  남      20      컴퓨터학과      2023-09-01      18612345607     ulsan
1008    한소희  여      21      물리학과        2022-09-01      18612345608     suwon
1009    임준영  남      19      컴퓨터학과      2023-09-01      18612345609     changwon
1010    송미나  여      20      수학과  2023-09-01      18612345610     jeonju
EOF
```

RIGHT: CN

```
-- 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(모든 데이터는 미리 넣어둠(/usr/datas/unit3)
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
```

데이터셋 2: 과목 성적 테이블 (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 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
```

데이터셋 3: 과목 정보 테이블 (courses.txt)

```bash
cat > /usr/datas/unit3/courses.txt << 'EOF'
C001	빅데이터기술	4	김교수
C002	데이터베이스	3	이교수
C003	파이썬프로그래밍	4	박교수
C004	머신러닝	3	정교수
C005	자료구조	4	최교수
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
```

데이터셋 4: 전자상거래 주문 테이블 (orders.txt)

```bash
cat > /usr/datas/unit3/orders.txt << 'EOF'
O0001	1001	2024-01-01 10:30:00	1580.50	결제완료
O0002	1002	2024-01-02 14:20:00	2399.00	결제완료
O0003	1003	2024-01-02 16:45:00	599.00	배송중
O0004	1001	2024-01-03 09:15:00	3200.00	결제완료
O0005	1004	2024-01-03 11:30:00	899.00	결제대기
O0006	1002	2024-01-04 15:20:00	1299.00	결제완료
O0007	1005	2024-01-05 10:10:00	4599.00	배송중
O0008	1003	2024-01-05 13:30:00	799.00	배송완료
O0009	1006	2024-01-06 16:00:00	1899.00	결제완료
O0010	1007	2024-01-07 09:45:00	2599.00	배송중
EOF
```

数据集 4: 电商订单表 (orders.txt)

```bash
cat > /usr/datas/unit3/orders.txt << 'EOF'
O0001	1001	2024-01-01 10:30:00	1580.50	已支付
O0002	1002	2024-01-02 14:20:00	2399.00	已支付
O0003	1003	2024-01-02 16:45:00	599.00	已发货
O0004	1001	2024-01-03 09:15:00	3200.00	已支付
O0005	1004	2024-01-03 11:30:00	899.00	待支付
O0006	1002	2024-01-04 15:20:00	1299.00	已支付
O0007	1005	2024-01-05 10:10:00	4599.00	已发货
O0008	1003	2024-01-05 13:30:00	799.00	已完成
O0009	1006	2024-01-06 16:00:00	1899.00	已支付
O0010	1007	2024-01-07 09:45:00	2599.00	已发货
EOF
```

LEFT: KR

RIGHT: CN

```
```

데이터셋 5: 주문 상세 테이블 (order_items.txt)

```bash
cat > /usr/datas/unit3/order_items.txt << 'EOF'
O001    P001    노트북  1       4999.00
O001    P002    마우스  2       99.00
O002    P003    스마트폰        1       2399.00
O003    P004    키보드  1       299.00
O003    P005    이어폰  1       199.00
O004    P001    노트북  1       4999.00
O004    P006    모니터  1       1599.00
O005    P007    태블릿  1       2899.00
O006    P003    스마트폰        1       2399.00
O007    P008    카메라  1       4599.00
O008    P009    스피커  1       799.00
O009    P010    공유기  2       299.00
O010    P003    스마트폰        1       2599.00
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
```

데이터셋 6: 직원 정보 테이블 (employees.txt)

```bash
cat > /usr/datas/unit3/employees.txt << 'EOF'
E001    김민준  남      28      개발팀  15000   2020-03-15      서울
E002    이서연  여      26      마케팅팀        12000   2021-06-01      부산
E003    박지훈  남      32      개발팀  25000   2018-01-10      인천
E004    정예은  여      29      인사팀  13000   2019-09-20      대구
E005    최우진  남      35      개발팀  28000   2017-05-15      서울
E006    강서윤  여      27      재무팀  14000   2020-11-01      광주
E007    윤도현  남      30      마케팅팀        16000   2019-03-25      부산
E008    한지우  여      25      인사팀  11000   2022-01-10      대전
E009    임현우  남      33      개발팀  26000   2018-07-20      인천
E010    송채원  여      28      마케팅팀        15000   2020-08-15      울산
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
```

LEFT: KR

RIGHT: CN

```
```

데이터셋 7: 웹사이트 접속 로그 (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
```

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 데이터 로딩 검증

1.3 数据导入验证(yàn zhèng)

Beeline 에서 기본 테이블 생성

```sql
USE unit3db;

-- 학생 테이블 생성
CREATE TABLE IF NOT EXISTS students (
    stu_id INT COMMENT '학번',
    stu_name STRING COMMENT '학생이름',
    gender STRING COMMENT '성별',
    age INT COMMENT '나이',
    college STRING COMMENT '학과',
    enroll_date DATE COMMENT '입학일',
    phone STRING COMMENT '전화번호',
    city STRING COMMENT '도시'
)
COMMENT '학생정보테이블'
ROW FORMAT DELIMITED
```

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
```

LEFT: KR

```
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

-- 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/students.txt'
OVERWRITE INTO TABLE students;

-- 데이터 확인
SELECT * FROM students LIMIT 5;
SELECT COUNT(*) as total_students FROM students;
```

예상 결과: 10 건의 학생 레코드가 조회되어야 함

제 2 부: 데이터 타입 및 테이블 조작 실습 (60 분)

2.1 복합 데이터 타입 실습

실습 목표: ARRAY, MAP, STRUCT 등 복합 데이터 타입 사용법 숙지

Step 1: 복합 타입을 포함한 테이블 생성

```sql
-- 학생 상세정보 테이블 생성(복합 타입 포함)
CREATE TABLE student_detail (
    stu_id INT COMMENT '학번',
    stu_name STRING COMMENT '이름',
    hobbies ARRAY<STRING> COMMENT '취미목록',
    scores MAP<STRING, INT> COMMENT '과목별성적',
    address STRUCT<province:STRING, city:STRING, district:STRING> COMMENT '주소정보',
    contact_info MAP<STRING, STRING> COMMENT '연락처정보'
)
COMMENT '학생상세정보테이블'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;
```

RIGHT: CN

```
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;
```(실행시 약 40.833 초 걸림)

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(구분문자:delimiter: 分隔符)
FIELDS TERMINATED BY '|'
COLLECTION ITEMS TERMINATED BY ','
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;
```

LEFT: KR

```
```

Step 2: 복합 타입 데이터 준비

```bash
cat > /usr/datas/unit3/student_detail.txt << 'EOF'
1001|김철수|농구,코딩,독서|수학:85,영어:90,물리:78|서울,강남구,역삼동|email:kim@example.com,qq:123456789
1002|이영희|수영,음악|수학:92,영어:88,화학:85|부산,해운대구,중동|email:lee@example.com,wechat:lee123
1003|박민수|축구,여행,사진|수학:76,물리:95,생물:80|인천,남동구,구월동|email:park@example.com,qq:987654321
1004|정수진|그림,춤|영어:89,화학:91,생물:87|대구,수성구,범어동|email:jung@example.com,phone:01012345678
1005|최동욱|코딩,게임|수학:94,영어:87,물리:91|광주,서구,치평동|email:choi@example.com,wechat:choi456
EOF
```

Step 3: 복합 타입 데이터 로딩 및 쿼리

```sql
-- 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/student_detail.txt'
OVERWRITE INTO TABLE student_detail;

-- ARRAY 타입 조회
SELECT stu_name, hobbies FROM student_detail;

-- ARRAY 요소 접근(첫 번째 취미)
SELECT stu_name, hobbies[0] as first_hobby FROM
student_detail;

-- ARRAY 크기 조회
SELECT stu_name, SIZE(hobbies) as hobby_count FROM
student_detail;

-- MAP 타입 조회
SELECT stu_name, scores FROM student_detail;

-- MAP 요소 접근(수학 성적)
SELECT stu_name, scores['수학'] as math_score FROM
student_detail;

-- MAP 의 모든 키 조회
SELECT stu_name, MAP_KEYS(scores) as subjects FROM
```

RIGHT: CN

```
```

步骤 2: 准备复杂类型数据

```bash(/usr/datas/unit3/student_detail.txt)
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
```

LEFT: KR

```
student_detail;

-- MAP 의 모든 값 조회
SELECT stu_name, MAP_VALUES(scores) as all_scores
FROM student_detail;

-- STRUCT 타입 조회
SELECT stu_name, address FROM student_detail;

-- STRUCT 필드 접근
SELECT stu_name,
       address.province as province,
       address.city as city,
       address.district as district
FROM student_detail;

-- 복합 쿼리: 수학 성적이 85 점 이상이고 취미에 코딩이 포함된 학생
SELECT stu_name, scores['수학'] as math_score,
hobbies
FROM student_detail
WHERE scores['수학'] > 85 AND
ARRAY_CONTAINS(hobbies, '코딩');
```

연습문제:

1. 각 학생의 취미 개수를 조회하고 개수 내림차순으로 정렬하세요
2. 모든 학생의 평균 수학 성적을 구하세요
3. 서울에 거주하는 학생의 이름과 이메일을 조회하세요

2.2 내부 테이블과 외부 테이블 비교

실습 목표: 내부 테이블과 외부 테이블의 차이점 이해

Step 1: 내부 테이블 확인

```sql
-- 내부 테이블(이미 생성됨)
-- students 테이블이 내부 테이블임
```

RIGHT: CN

```
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:

1. 查询每个学生的爱好数量,并按数量降序排列
2. 查询所有学生的平均数学成绩
3. 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
```

LEFT: KR

```
-- 테이블 정보 조회
DESC FORMATTED students;
```

Step 2: 외부 테이블 생성

```sql
-- 외부 테이블 생성
CREATE EXTERNAL TABLE students_external (
    stu_id INT COMMENT '학번',
    stu_name STRING COMMENT '학생이름',
    gender STRING COMMENT '성별',
    age INT COMMENT '나이',
    college STRING COMMENT '학과',
    enroll_date DATE COMMENT '입학일',
    phone STRING COMMENT '전화번호',
    city STRING COMMENT '도시'
)
COMMENT '학생정보외부테이블'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/external/students';

-- 테이블 정보 조회
DESC FORMATTED students_external;
```

Step 3: 비교 테스트

```sql
-- Linux 터미널에서 외부 테이블 데이터 디렉토리 생성
-- hadoop fs -mkdir -p /user/hive/external/students
-- hadoop fs -put /usr/datas/unit3/students.txt
/user/hive/external/students/

-- 외부 테이블 데이터 조회
SELECT * FROM students_external LIMIT 5;

-- 내부 테이블 삭제(데이터도 삭제됨)
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;

-- 외부 테이블 삭제(데이터는 유지됨)
```

RIGHT: CN

```
-- 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)
```

```
-- 테스트 생략, 실수로 삭제 방지
```

```
-- 测试略过,避免误删除
```

지식 요약:

- 내부 테이블(MANAGED TABLE): DROP TABLE 시 메타데이터와 데이터 모두 삭제
- 외부 테이블(EXTERNAL TABLE): DROP TABLE 시 메타데이터만 삭제, 데이터 파일은 유지

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 파티션 테이블 실습

2.3 分区表实践(shí jiàn)

실습 목표: 파티션 테이블의 생성과 사용법 숙지

实践目标: 掌握分区表的创建和使用

Step 1: 파티션 테이블 생성

```sql
-- 날짜별 파티션 주문 테이블 생성
CREATE TABLE orders_partitioned (
    order_id STRING COMMENT '주문 ID',
    customer_id INT COMMENT '고객 ID',
    order_time TIMESTAMP COMMENT '주문시간',
    amount DECIMAL(10,2) COMMENT '주문금액',
    status STRING COMMENT '주문상태'
)
COMMENT '주문파티션테이블'
PARTITIONED BY (order_date STRING COMMENT '주문날짜파티션')
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

步骤 1: 创建分区表

```sql
-- 创建按日期分区的订单表
CREATE TABLE orders_partitioned (
    order_id STRING COMMENT '订单 ID',
    customer_id INT COMMENT '客户 ID',
    order_time TIMESTAMP COMMENT '订单时间',
    amount DECIMAL(10,2) COMMENT '订单金额',
    status STRING COMMENT '订单状态'
)
COMMENT '订单分区表'
PARTITIONED BY (order_date STRING COMMENT '订单日期分区')
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Step 2: 파티션 데이터 준비

```bash
# 날짜별 주문 데이터 생성
cat > /usr/datas/unit3/orders_20240101.txt << 'EOF'
0001    1001    2024-01-01 10:30:00    1580.50 결제완료
0002    1002    2024-01-01 14:20:00    2399.00 결제완료
```

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'
0001    1001    2024-01-01 10:30:00    1580.50 已支付
0002    1002    2024-01-01 14:20:00    2399.00 已支付
```

LEFT: KR

```
EOF

cat > /usr/datas/unit3/orders_20240102.txt << 'EOF'
0003    1003    2024-01-02 16:45:00    599.00  배송중
0004    1001    2024-01-02 09:15:00    3200.00 결제완료
EOF

cat > /usr/datas/unit3/orders_20240103.txt << 'EOF'
0005    1004    2024-01-03 11:30:00    899.00  결제대기
0006    1002    2024-01-03 15:20:00    1299.00 결제완료
EOF
```

Step 3: 파티션 데이터 로딩

```sql
-- 2024-01-01 파티션 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/orders_20240101.txt'
INTO TABLE orders_partitioned
PARTITION (order_date='2024-01-01');

-- 2024-01-02 파티션 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/orders_20240102.txt'
INTO TABLE orders_partitioned
PARTITION (order_date='2024-01-02');

-- 2024-01-03 파티션 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/orders_20240103.txt'
INTO TABLE orders_partitioned
PARTITION (order_date='2024-01-03');

-- 모든 파티션 조회
SHOW PARTITIONS orders_partitioned;

-- 특정 파티션 데이터 조회
SELECT * FROM orders_partitioned WHERE order_date =
'2024-01-01';

-- 다중 파티션 데이터 조회
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;

-- 모든 파티션 데이터 조회
SELECT * FROM orders_partitioned;
```

RIGHT: CN

```
EOF

cat > /usr/datas/unit3/orders_20240102.txt << 'EOF'
0003    1003    2024-01-02 16:45:00    599.00  已发货
0004    1001    2024-01-02 09:15:00    3200.00 已支付
EOF

cat > /usr/datas/unit3/orders_20240103.txt << 'EOF'
0005    1004    2024-01-03 11:30:00    899.00  待支付
0006    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; (执行时间:64.136 秒)

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

Step 4: 동적 파티션 삽입

```sql
-- 동적 파티션 활성화
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;

-- 임시 주문 테이블 생성
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 ',';

-- 테스트 데이터 삽입
INSERT INTO orders_temp VALUES
('O007', 1005, '2024-01-04', '2024-01-04 10:10:00', 4599.00, '배송중'),
('O008', 1003, '2024-01-04', '2024-01-04 13:30:00', 799.00, '배송완료'),
('O009', 1006, '2024-01-05', '2024-01-05 16:00:00', 1899.00, '결제완료');

-- 동적 파티션 삽입
INSERT INTO TABLE orders_partitioned
PARTITION(order_date)
SELECT order_id, customer_id, order_time, amount, status, order_date
FROM orders_temp;

-- 동적 파티션 검증
SHOW PARTITIONS orders_partitioned;
SELECT * FROM orders_partitioned WHERE order_date IN ('2024-01-04', '2024-01-05');
```

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;

-- 动态分区插入
INSERT INTO TABLE orders_partitioned
PARTITION(order_date)
SELECT order_id, customer_id, order_time, amount, status, order_date
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');
```

연습문제:

1. 각 파티션의 주문 총 금액을 통계내세요
2. 금액이 2000 을 초과하는 주문이 있는 파티션을 조회하세요
3. order_date='2024-01-05' 파티션을 삭제하세요

练习题:

1. 统计每个分区的订单总金额
2. 查询金额超过 2000 的订单所在的分区
3. 删除 order_date='2024-01-05'的分区

제 3 부: 연산자 및 함수 종합 실습 (90 분)

第三部分:运算符与函数综合实践 (90 分钟)

3.1 산술 및 비교 연산자

3.1 算术与比较运算符

기본 테이블 생성

```sql
-- 성적 테이블 생성
CREATE TABLE scores (
    stu_id INT COMMENT '학번',
    course_id STRING COMMENT '과목 ID',
    score INT COMMENT '성적',
    exam_date DATE COMMENT '시험일자'
)
COMMENT '과목성적테이블'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

-- 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/scores.txt'
OVERWRITE INTO TABLE scores;

-- 데이터 확인
SELECT * FROM scores LIMIT 5;
```

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;
```

산술 연산 실습

```sql
-- 기본 산술 연산
SELECT
    stu_id,
    course_id,
    score,
    score + 5 as bonus_score,          -- 가산점
    score - 5 as penalty_score,        -- 감점
    score * 1.1 as scaled_score,       -- 배율 적용
    score / 10 as score_level,         -- 점수 등급
    score % 10 as score_remainder      -- 나머지
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;
```

```sql
-- 성적 통계
SELECT
    stu_id,
    COUNT(*) as course_count,              -- 과목수
    SUM(score) as total_score,             -- 총점
    AVG(score) as avg_score,               -- 평균
    MAX(score) as max_score,               -- 최고점
    MIN(score) as min_score,               -- 최저점
    MAX(score) - MIN(score) as score_diff  -- 점수 격차
FROM scores
GROUP BY stu_id
ORDER BY avg_score DESC;

-- 합격률 계산
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;
```

비교 연산 실습

```sql
-- 기본 비교 연산
SELECT stu_id, course_id, score
FROM scores
WHERE score > 85;  -- 85점 초과

SELECT stu_id, course_id, score
FROM scores
WHERE score >= 90;  -- 90점 이상(우수)

SELECT stu_id, course_id, score
FROM scores
WHERE score < 60;  -- 60점 미만(불합격)

SELECT stu_id, course_id, score
FROM scores
WHERE score <= 80;  -- 80점 이하
```

```sql
-- 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
```

LEFT: KR

```
SELECT stu_id, course_id, score
FROM scores
WHERE score = 90;  -- 90 점

SELECT stu_id, course_id, score
FROM scores
WHERE score != 85;  -- 85 점이 아님(<>도 가능)

SELECT stu_id, course_id, score
FROM scores
WHERE score <> 85;  -- 85 점이 아님

-- BETWEEN 범위 쿼리
SELECT stu_id, course_id, score
FROM scores
WHERE score BETWEEN 80 AND 90;  -- 80~90 점 사이

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

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

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

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

연습문제:

1. 평균 점수가 85~95 점 사이인 학생 ID 를 조회하세요
2. 각 과목의 최고점, 최저점, 평균점을 계산하세요
3. 불합격 과목이 있는 학생 ID 를 조회하세요

3.2 논리 연산자 응용

RIGHT: CN

```
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:

1. 查询平均分在 85-95 之间的学生 ID
2. 计算每门课程的最高分、最低分和平均分
3. 查询有不及格科目的学生 ID

3.2 逻辑运算符应用

LEFT: KR **RIGHT: CN**

```sql
-- AND 연산자(그리고)
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 연산자(또는)
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 연산자(부정)
SELECT stu_id, course_id, score
FROM scores
WHERE NOT score < 60;  -- score >= 60 과 동일

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

-- 복합 논리 연산
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');

-- 다중 조건 조합 쿼리
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    s.score,
    CASE
        WHEN s.score >= 90 THEN '우수'
        WHEN s.score >= 80 AND s.score < 90 THEN
'양호'
        WHEN s.score >= 70 AND s.score < 80 THEN
'보통'
        WHEN s.score >= 60 AND s.score < 70 THEN
```

```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
```

LEFT: KR

```
        '합격'
        ELSE '불합격'
    END as grade_level
FROM scores s
JOIN students st ON s.stu_id = st.stu_id
WHERE (s.score >= 85 OR st.college = '컴퓨터학과')
    AND s.exam_date >= '2024-01-15'
ORDER BY s.score DESC;
```

연습문제:

1. 점수가 60~80 점 사이이거나 90 점 이상인 기록을 조회하세요
2. 컴퓨터학과이면서 우수 성적(>=90)인 학생을 조회하세요
3. C001 과목이 아니면서 불합격인 기록을 조회하세요

3.3 문자열 함수 실습

과목 테이블 생성

```sql
-- 과목 테이블 생성
CREATE TABLE courses (
    course_id STRING COMMENT '과목 ID',
    course_name STRING COMMENT '과목명',
    credits INT COMMENT '학점',
    teacher STRING COMMENT '담당교수'
)
COMMENT '과목정보테이블'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

-- 데이터 로딩
LOAD DATA LOCAL INPATH
'/usr/datas/unit3/courses.txt'
OVERWRITE INTO TABLE courses;
```

문자열 함수 종합 응용

RIGHT: CN

```
        '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:

1. 查询分数在 60-80 之间或 90 以上的记录
2. 查询计算机学院且成绩优秀(>=90)的学生
3. 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

LEFT: KR **RIGHT: CN**

```sql
-- LENGTH/CHAR_LENGTH: 문자열 길이
SELECT
    stu_name,
    LENGTH(stu_name) as name_byte_length,
    CHAR_LENGTH(stu_name) as name_char_length
FROM students;

-- CONCAT: 문자열 연결
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: 구분자를 사용한 문자열 연결
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: 문자열 추출
SELECT
    stu_name,
    SUBSTR(stu_name, 1, 1) as surname,
-- 성
    SUBSTRING(phone, 1, 3) as phone_prefix,
-- 전화번호 앞 3 자리
    SUBSTRING(phone, -4) as phone_suffix
-- 전화번호 뒤 4 자리
FROM students;

-- UPPER/LOWER: 대소문자 변환
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: 공백 제거
SELECT
    TRIM('  Hello World  ') as trimmed,
    LTRIM('  Hello World  ') as left_trimmed,
    RTRIM('  Hello World  ') as right_trimmed;
```

```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;
```

LEFT: KR

```
-- LPAD/RPAD: 문자열 채우기
SELECT
    stu_id,
    LPAD(stu_id, 10, '0') as padded_id,
-- 왼쪽 채우기
    RPAD(stu_name, 10, '*') as padded_name
-- 오른쪽 채우기
FROM students
LIMIT 5;

-- INSTR: 부분문자열 위치 찾기
SELECT
    stu_name,
    INSTR(stu_name, '김') as position,
    INSTR(phone, '186') as has_186
FROM students;

-- LOCATE: 부분문자열 위치 찾기(INSTR과 유사, 파라미터 순서 다름)
SELECT
    stu_name,
    LOCATE('김', stu_name) as position
FROM students;

-- REPLACE: 문자열 치환
SELECT
    college,
    REPLACE(college, '학과', 'Department') as college_en,
    phone,
    REPLACE(phone, '186', '138') as new_phone
FROM students;

-- SPLIT: 문자열 분할
SELECT
    stu_name,
    SPLIT(phone, '123')[0] as phone_part1
FROM students
LIMIT 3;

-- REVERSE: 문자열 역순
SELECT
    stu_name,
    REVERSE(stu_name) as reversed_name
FROM students
LIMIT 5;

-- REPEAT: 문자열 반복
SELECT
    stu_name,
    REPEAT('*', 3) as stars,
    CONCAT(REPEAT('=', 10), stu_name, REPEAT('=', 10)) as decorated_name
```

RIGHT: CN

```
-- 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
```

LEFT: KR

```
FROM students
LIMIT 5;

-- 종합 사례: 학생 정보 포맷팅
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, '학과', '') as short_college
FROM students;
```

연습문제:

1. 모든 학생의 성을 추출하고 성별 인원수를 통계내세요
2. 모든 과목명을 "[과목 ID]-과목명" 형식으로 변환하세요
3. 전화번호를 마스킹 처리하여 앞 3 자리와 뒤 4 자리만 표시하세요

3.4 날짜/시간 함수 실습

```sql
-- 현재 날짜시간
SELECT
    CURRENT_DATE() as current_date,                -- 현재 날짜
    CURRENT_TIMESTAMP() as current_timestamp,      -- 현재 타임스탬프
    NOW() as now_time,                             -- 현재 시간
    UNIX_TIMESTAMP() as unix_ts;                   -- Unix 타임스탬프

-- 날짜 추출
SELECT
    enroll_date,
    YEAR(enroll_date) as enroll_year,              -- 연도
    MONTH(enroll_date) as enroll_month,            -- 월
    DAY(enroll_date) as enroll_day,                -- 일
```

RIGHT: CN

```
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:

1. 提取所有学生的姓氏,并统计每个姓氏的人数
2. 将所有课程名称转换为"[课程 ID]-课程名称"格式
3. 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
```

LEFT: KR **RIGHT: CN**

```
    DAYOFMONTH(enroll_date) as day_of_month,     -- 월의 몇 번째 날
    DAYOFWEEK(enroll_date) as day_of_week,       -- 주의 몇 번째 날(1=일요일)
    DAYOFYEAR(enroll_date) as day_of_year,       -- 연도의 몇 번째 날
    WEEKOFYEAR(enroll_date) as week_of_year,     -- 연도의 몇 번째 주
    QUARTER(enroll_date) as quarter              -- 분기
FROM students;

-- 날짜 포맷팅
SELECT
    enroll_date,
    DATE_FORMAT(enroll_date, 'yyyy-MM-dd') as formatted_date,
    DATE_FORMAT(enroll_date, 'yyyy 년 MM 월 dd 일') as korean_date,
    DATE_FORMAT(enroll_date, 'yyyy-MM') as year_month
FROM students;

-- 날짜 변환
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;

-- 날짜 계산
SELECT
    enroll_date,
    DATE_ADD(enroll_date, 7) as add_7days,
-- 7 일 추가
    DATE_ADD(enroll_date, 1) as tomorrow,
-- 내일
    DATE_SUB(enroll_date, 7) as sub_7days,
-- 7 일 빼기
    ADD_MONTHS(enroll_date, 6) as add_6months,
-- 6 개월 추가
    ADD_MONTHS(enroll_date, -3) as sub_3months
-- 3 개월 빼기
FROM students;

-- 날짜 차이 계산
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
```

```
    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;

-- 월의 마지막 날 구하기
SELECT
    enroll_date,
    LAST_DAY(enroll_date) as last_day_of_month
FROM students;

-- 다음 요일 구하기
SELECT
    CURRENT_DATE() as today,
    NEXT_DAY(CURRENT_DATE(), 'MON') as next_monday,
    NEXT_DAY(CURRENT_DATE(), 'FRI') as next_friday;

-- 타임스탬프 변환
SELECT
    exam_date,
    UNIX_TIMESTAMP(exam_date) as timestamp,
    FROM_UNIXTIME(UNIX_TIMESTAMP(exam_date),
'yyyyMMdd') as formatted_ts
FROM scores
LIMIT 5;

-- 종합 사례: 학생 입학 분석
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;

-- 시험 날짜의 요일 계산
SELECT
    exam_date,
    CASE DAYOFWEEK(exam_date)
        WHEN 1 THEN '일요일'
        WHEN 2 THEN '월요일'
        WHEN 3 THEN '화요일'
        WHEN 4 THEN '수요일'
        WHEN 5 THEN '목요일'
        WHEN 6 THEN '금요일'
        WHEN 7 THEN '토요일'
    END as weekday_name,
    COUNT(*) as exam_count
FROM scores
GROUP BY exam_date, DAYOFWEEK(exam_date);
```

```
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);
```

연습문제:

1. 각 학생의 입학일부터 현재까지의 일수를 계산하세요
2. 월별 입학 학생 수를 통계내세요
3. 입학한지 1 년 이상인 학생 정보를 조회하세요

练习题:

1. 计算每个学生从入学到现在的天数
2. 统计每个月入学的学生数量
3. 查询入学超过 1 年的学生信息

3.5 조건 함수 실습

```sql
-- IF 함수
SELECT
    stu_id,
    course_id,
    score,
    IF(score >= 60, '합격', '불합격') as result,
    IF(score >= 90, '우수', '비우수') as level
FROM scores;

-- CASE WHEN - 단순 형식
SELECT
    stu_id,
    course_id,
    score,
    CASE score >= 60
        WHEN true THEN '합격'
        ELSE '불합격'
    END as pass_status
FROM scores;

-- CASE WHEN - 검색 형식(권장)
SELECT
    stu_id,
    course_id,
    score,
    CASE
        WHEN score >= 90 THEN 'A-우수'
        WHEN score >= 80 THEN 'B-양호'
        WHEN score >= 70 THEN 'C-보통'
        WHEN score >= 60 THEN 'D-합격'
        ELSE 'F-불합격'
    END as grade,
    CASE
        WHEN score >= 90 THEN 4.0
        WHEN score >= 80 THEN 3.0
```

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
```

LEFT: KR

```
        WHEN score >= 70 THEN 2.0
        WHEN score >= 60 THEN 1.0
        ELSE 0.0
    END as gpa
FROM scores;

-- COALESCE: 첫 번째 NULL 이 아닌 값 반환
SELECT
    stu_id,
    COALESCE(NULL, NULL, score, 0) as
score_or_default,
    COALESCE(NULL, score) as score_coalesce
FROM scores
LIMIT 5;

-- NVL: 첫 번째 파라미터가 NULL 이면 두 번째 파라미터
반환
SELECT
    stu_id,
    score,
    NVL(score, 0) as score_with_default,
    NVL(NULL, 60) as default_pass_score
FROM scores
LIMIT 5;

-- NULLIF: 두 파라미터가 같으면 NULL 반환, 아니면 첫
번째 파라미터 반환
SELECT
    score,
    NULLIF(score, 85) as nullif_85,  --
score=85 이면 NULL 반환
    NULLIF(score, 0) as nullif_0      --
score=0 이면 NULL 반환
FROM scores
LIMIT 10;

-- 종합 사례 1: 성적 분석 보고서
SELECT
    s.stu_id,
    st.stu_name,
    s.course_id,
    c.course_name,
    s.score,
    CASE
        WHEN s.score >= 90 THEN '우수'
        WHEN s.score >= 80 THEN '양호'
        WHEN s.score >= 70 THEN '보통'
        WHEN s.score >= 60 THEN '합격'
        ELSE '불합격'
    END as level,
    IF(s.score >= AVG(s.score) OVER(), '평균이상',
'평균이하') as vs_avg,
```

RIGHT: CN

```
        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,
```

LEFT: KR

```
    CASE
        WHEN s.score >= 85 THEN '다음학기면제'
        WHEN s.score < 60 THEN '재시험필요'
        ELSE '정상'
    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;

-- 종합 사례 2: 학생 프로필
SELECT
    stu_id,
    stu_name,
    age,
    CASE
        WHEN age < 20 THEN '19 세이하'
        WHEN age >= 20 AND age < 22 THEN '20-21 세'
        ELSE '22 세이상'
    END as age_group,
    college,
    CASE college
        WHEN '컴퓨터학과' THEN '공학'
        WHEN '수학과' THEN '이학'
        WHEN '물리학과' THEN '이학'
        WHEN '화학과' THEN '이학'
        ELSE '기타'
    END as college_type,
    DATEDIFF(CURRENT_DATE(), enroll_date) as
days_enrolled,
    CASE
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date)
< 365 THEN '1 학년'
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date)
< 730 THEN '2 학년'
        WHEN DATEDIFF(CURRENT_DATE(), enroll_date)
< 1095 THEN '3 학년'
        ELSE '4 학년이상'
    END as grade_level
FROM students;
```

RIGHT: CN

```
    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;
```

연습문제:

1. 점수를 5 등급제로 변환하는 쿼리를 작성하세요(우수/양호/보통/합격/불합격)
2. 각 성적 등급별 학생 수를 통계내세요
3. 각 과목 평균 이상의 성적을 받은 학생을 조회하세요

练习题:

1. 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)
2. 统计每个成绩等级的学生人数
3. 查询成绩在各科平均分以上的学生

3.6 집계 함수 응용

```sql
-- 기본 집계 함수
SELECT
    COUNT(*) as total_records,                    -- 총 레코드 수
    COUNT(DISTINCT stu_id) as unique_students,    -- 중복 제외 학생 수
    COUNT(DISTINCT course_id) as unique_courses,  -- 중복 제외 과목 수
    SUM(score) as total_score,                    -- 총점
    AVG(score) as avg_score,                      -- 평균
    MAX(score) as max_score,                      -- 최고점
    MIN(score) as min_score,                      -- 최저점
    VARIANCE(score) as variance,                  -- 분산
    VAR_POP(score) as var_population,             -- 모집단 분산
    STDDEV(score) as stddev,                      -- 표준편차
    STDDEV_POP(score) as stddev_population        -- 모집단 표준편차
FROM scores;

-- 학생별 그룹 집계
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;

-- 과목별 그룹 집계
SELECT
    course_id,
    COUNT(*) as student_count,
    ROUND(AVG(score), 2) as avg_score,
    MAX(score) as max_score,
```

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 절 필터링
SELECT
    stu_id,
    COUNT(*) as course_count,
    ROUND(AVG(score), 2) as avg_score
FROM scores
GROUP BY stu_id
HAVING AVG(score) > 85;

-- 다중 컬럼 그룹화
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 과 COLLECT_LIST
SELECT
    stu_id,
    COLLECT_SET(course_id) as courses_taken,       -- 중복 제거 과목 리스트
    COLLECT_LIST(score) as all_scores,
-- 모든 점수 리스트
    COLLECT_LIST(course_id) as course_list
-- 과목 리스트(중복 포함)
FROM scores
GROUP BY stu_id;

-- 윈도우 집계 함수
SELECT
    stu_id,
    course_id,
    score,
    AVG(score) OVER() as overall_avg,
-- 전체 평균
    AVG(score) OVER(PARTITION BY course_id) as course_avg,  -- 과목별 평균
    AVG(score) OVER(PARTITION BY stu_id) as student_avg,    -- 학생별 평균
    MAX(score) OVER(PARTITION BY course_id) as course_max,  -- 과목별 최고점
```

```
    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
```

LEFT: KR

```
    MIN(score) OVER(PARTITION BY course_id) as course_min   -- 과목별 최저점
FROM scores
ORDER BY stu_id, course_id;

-- 순위 함수
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;

-- 종합 사례: 성적 순위 보고서
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 '상위 3 명'
        WHEN s.score >= AVG(s.score) OVER(PARTITION BY s.course_id) THEN '중상'
        ELSE '중하'
    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;
```

RIGHT: CN

```
    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;
```

연습문제:

1. 각 학과의 학생 수와 평균 나이를 통계내세요
2. 평균 점수 상위 3 명의 학생 정보를 조회하세요
3. 각 과목의 합격 인원과 합격률을 통계내세요

练习题:

1. 统计每个学院的学生人数和平均年龄
2. 查询平均分排名前 3 的学生信息
3. 统计每门课程的及格人数和及格率

제 4 부: 쿼리 및 분석 종합 실습 (60 분)

4.1 복잡한 WHERE 조건 쿼리

```sql
-- 주문 관련 테이블 준비
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;

-- 복잡 조건 쿼리 예제
-- 쿼리 1: 다중 조건 조합
SELECT *
FROM orders
WHERE amount > 1000
    AND status IN ('결제완료', '배송중')
    AND DATE(order_time) >= '2024-01-02';

-- 쿼리 2: 서브쿼리를 조건으로
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT stu_id
    FROM students
    WHERE college = '컴퓨터학과'
```

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'
```

```
);

-- 쿼리 3: 상관 서브쿼리
SELECT o.*
FROM orders o
WHERE amount > (
    SELECT AVG(amount)
    FROM orders
    WHERE customer_id = o.customer_id
);

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

-- 쿼리 5: NOT EXISTS 서브쿼리
SELECT s.*
FROM students s
WHERE NOT EXISTS (
    SELECT 1
    FROM scores sc
    WHERE sc.stu_id = s.stu_id AND sc.score < 60
);

-- 쿼리 6: ALL 과 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
);

-- 쿼리 7: 복잡한 패턴 매칭
SELECT *
FROM students
WHERE stu_name LIKE '김%'     -- 성이 김
   OR stu_name LIKE '%수%'    -- 이름에 수 포함
   OR college LIKE '%컴퓨터%'; -- 학과명에 컴퓨터 포함
```

```
);

-- 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
```

LEFT: KR

```
-- 쿼리 8: 정규표현식 매칭
SELECT *
FROM students
WHERE stu_name REGEXP '^[김이박]'  -- 성이 김, 이, 박
   OR phone REGEXP '^186';         -- 전화번호가 186 으로 시작
```

연습문제:

1. 구매 금액이 가장 높은 고객의 모든 주문을 조회하세요
2. 3 개 이상의 과목을 수강한 학생 정보를 조회하세요
3. 모든 과목을 합격한 학생을 조회하세요

4.2 GROUP BY 와 집계 분석

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

-- 다중 컬럼 그룹화
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(소계)
SELECT
    college,
    gender,
    COUNT(*) as count
FROM students
GROUP BY college, gender WITH ROLLUP;

-- GROUPING SETS(그룹 집합)
```

RIGHT: CN

```
-- 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:

1. 查询购买金额最高的客户的所有订单
2. 查询至少选修了 3 门课程的学生信息
3. 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)
```

LEFT: KR

```
SELECT
    college,
    gender,
    COUNT(*) as count
FROM students
GROUP BY college, gender
GROUPING SETS (
    (college, gender),  -- 학과와 성별로 그룹화
    (college),          -- 학과로만 그룹화
    (gender),           -- 성별로만 그룹화
    ()                  -- 전체 합계
);

-- CUBE(다차원 분석)
SELECT
    college,
    gender,
    COUNT(*) as count
FROM students
GROUP BY college, gender WITH CUBE;

-- HAVING 으로 그룹 결과 필터링
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;

-- 집계 후 계산
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;

-- 종합 사례: 학생 성적 분석
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,
```

RIGHT: CN

```
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;
```

```
    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;
```

연습문제:

1. 각 도시의 학생 수와 평균 나이를 통계내세요
2. 각 과목의 합격률을 조회하되, 합격률이 90% 미만인 과목만 표시하세요
3. 각 학생의 수강 과목 수와 평균 점수를 통계내고 평균 점수 상위 5 명을 찾으세요

练习题:

1. 统计每个城市的学生数量和平均年龄
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
3. 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 와 정렬

```sql
-- 단일 컬럼 정렬
SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC;  -- 내림차순

SELECT stu_id, course_id, score
FROM scores
ORDER BY score ASC;   -- 오름차순(기본값)

-- 다중 컬럼 정렬
SELECT stu_id, course_id, score
FROM scores
ORDER BY stu_id ASC, score DESC;

-- NULL 값 정렬
SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC NULLS LAST;   -- NULL 값을 마지막에

SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC NULLS FIRST;  -- NULL 값을 먼저
```

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
```

LEFT: KR

```
-- 표현식으로 정렬
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;

-- 집계 결과로 정렬
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 절(페이징)
SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC
LIMIT 10;  -- 상위 10 명

SELECT stu_id, course_id, score
FROM scores
ORDER BY score DESC
LIMIT 5, 5;  -- 앞의 5 건을 건너뛰고 다음 5 건 조회(6~10 위)

-- 종합 사례: 성적 순위표
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
```

RIGHT: CN

```
-- 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
```

LEFT: KR

```
LIMIT 10;
```

연습문제:

1. 각 과목 성적 상위 3 명의 학생을 조회하세요
2. 학과와 평균 점수로 정렬하여 상위 10 명의 학생을 표시하세요
3. 각 학생의 순위를 평균 점수로 조회하세요

4.4 JOIN 다중 테이블 연관

```sql
-- INNER JOIN(내부 조인)
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(왼쪽 외부 조인)
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(오른쪽 외부 조인)
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;
```

RIGHT: CN

```
LIMIT 10;
```

练习题:

1. 查询每门课程成绩最高的 3 名学生
2. 按学院和平均分排序,显示前 10 名学生
3. 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;
```

```sql
-- FULL OUTER JOIN(전체 외부 조인)
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;
```

```sql
-- 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;
```

```sql
-- CROSS JOIN(교차 조인/카티션 곱)
SELECT
    st.stu_id,
    st.stu_name,
    c.course_id,
    c.course_name
FROM students st
CROSS JOIN courses c
WHERE st.college = '컴퓨터학과'
ORDER BY st.stu_id, c.course_id
LIMIT 20;
```

```sql
-- 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;
```

```sql
-- SELF JOIN(자기 조인)
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  -- 점수 차이가
5 점 이내
ORDER BY s1.course_id, s1.stu_id;
```

```sql
-- 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;
```

```sql
-- 다중 테이블 복잡 연관
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;
```

```sql
-- 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;
```

```sql
-- 서브쿼리를 포함한 JOIN
```

```sql
-- 带子查询的 JOIN
```

LEFT: KR

```
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;

-- 종합 사례: 학생 성적표
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;
```

RIGHT: CN

```
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;
```

연습문제:

1. "빅데이터기술" 과목을 수강한 모든 학생 정보를 조회하세요
2. 시험 기록이 없는 학생을 조회하세요
3. 각 학과의 각 과목별 평균 점수를 조회하세요

练习题:

1. 查询选修了"大数据技术"课程的所有学生信息
2. 查询没有考试记录的学生
3. 查询每个学院各门课程的平均分

4.5 서브쿼리 응용

```sql
-- WHERE 서브쿼리
-- 평균보다 높은 성적 조회
SELECT stu_id, course_id, score
FROM scores
WHERE score > (SELECT AVG(score) FROM scores);

-- FROM 서브쿼리(인라인 뷰)
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 서브쿼리(스칼라 서브쿼리)
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;

-- 상관 서브쿼리
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 서브쿼리
SELECT *
FROM students
WHERE stu_id IN (
    SELECT stu_id
```

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
```

LEFT: KR

```
    FROM scores
    WHERE score >= 90
);

-- NOT IN 서브쿼리
SELECT *
FROM students
WHERE stu_id NOT IN (
    SELECT stu_id
    FROM scores
    WHERE score < 60
);

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

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

-- 다층 중첩 서브쿼리
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
);

-- 종합 사례: 복잡한 분석 쿼리
SELECT
    st.stu_id,
    st.stu_name,
    st.college,
    overall.course_count,
    overall.avg_score,
    overall.max_score,
    overall.min_score,
```

RIGHT: CN

```
    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,
```

LEFT: KR

```
    CASE
        WHEN overall.avg_score >= 90 THEN '우등생'
        WHEN overall.avg_score >= 80 THEN '양호학생'
        WHEN overall.avg_score >= 70 THEN '보통학생'
        ELSE '일반학생'
    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;
```

연습문제:

1. 평균 점수가 가장 높은 학생의 상세 정보를 조회하세요
2. 각 과목에서 성적 상위 3 명의 학생을 조회하세요
3. C001 과 C002 과목을 동시에 수강한 학생을 조회하세요

RIGHT: CN

```
    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:

1. 查询平均分最高的学生的详细信息
2. 查询每门课程中成绩排名前 3 的学生
3. 查询同时选修了 C001 和 C002 课程的学生

제 5 부: 데이터 입출력 실습 (30 분)

5.1 다양한 데이터 입력 방식

```sql
-- 방식 1: LOAD DATA LOCAL INPATH(로컬 파일시스템에서 로딩)
```

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)
```

LEFT: KR **RIGHT: CN**

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

-- 방식 2: LOAD DATA INPATH(HDFS 에서 로딩)
-- 먼저 HDFS 에 파일 업로드
-- hadoop fs -put /usr/datas/unit3/students.txt
/tmp/
-- LOAD DATA INPATH '/tmp/students.txt'
-- OVERWRITE INTO TABLE students;

-- 방식 3: INSERT INTO SELECT(다른 테이블에서 삽입)
CREATE TABLE students_backup AS SELECT * FROM
students;

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

-- 방식 4: INSERT INTO VALUES(직접 데이터 삽입)
CREATE TABLE test_insert (
    id INT,
    name STRING,
    age INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';

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

SELECT * FROM test_insert;

-- 방식 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;

-- 방식 6: Hive 테이블을 MySQL 로 내보내기(Sqoop 사용 -
예시만)
-- sqoop export \
-- --connect jdbc:mysql://localhost:3306/unit3db \
-- --username hiveuser \
```

```
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 \
```

LEFT: KR

```
-- --password 123456 \
-- --table students_mysql \
-- --export-dir
/user/hive/warehouse/unit3db.db/students \
-- --input-fields-terminated-by ','

-- 방식 7: 다중 파일 일괄 로딩
-- 같은 디렉토리의 여러 파일을 한 번에 로딩
-- LOAD DATA LOCAL INPATH '/usr/datas/unit3/*'
-- INTO TABLE target_table;
```

5.2 데이터 출력 실습

```sql
-- 방식 1: INSERT OVERWRITE LOCAL DIRECTORY(로컬
파일시스템으로 출력)
INSERT OVERWRITE LOCAL DIRECTORY
'/usr/datas/unit3/output/students_export'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM students;

-- 방식 2: 쿼리 결과 출력
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;

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

-- 방식 4: 여러 파일로 출력(파티션별)
INSERT OVERWRITE LOCAL DIRECTORY
'/usr/datas/unit3/output/by_college'
ROW FORMAT DELIMITED
```

RIGHT: CN

```
-- --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
```

LEFT: KR

```
FIELDS TERMINATED BY ','
SELECT college, stu_id, stu_name, age
FROM students
DISTRIBUTE BY college;  -- 학과별로 다른 파일에 분산

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

-- 방식 6: MySQL 로 출력(Sqoop 사용 - 예시만)
-- 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 종합 사례

사례 1: 전자상거래 주문 분석 시스템

```sql
-- 분석 결과 테이블 생성
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;

-- 분석 결과 조회
SELECT * FROM order_analysis ORDER BY total_amount
DESC;

-- 분석 결과 출력
```

RIGHT: CN

```
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
```

LEFT: KR

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

RIGHT: CN

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

사례 2: 학생 성적 종합 분석

案例 2: 学生成**绩综**合分析

```sql
-- 성적 분석 와이드 테이블 생성
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 급학생'
        WHEN AVG(s.score) >= 80 THEN 'B 급학생'
        WHEN AVG(s.score) >= 70 THEN 'C 급학생'
        ELSE 'D 급학생'
    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;

-- 분석 결과 조회
SELECT * FROM student_performance_analysis
ORDER BY avg_score DESC;

-- 학과별 통계
SELECT
    college,
    COUNT(*) as student_count,
    ROUND(AVG(avg_score), 2) as college_avg_score,
    SUM(excellent_count) as total_excellent,
```

```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,
```

LEFT: KR

```
    SUM(fail_count) as total_fail
FROM student_performance_analysis
GROUP BY college
ORDER BY college_avg_score DESC;

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

RIGHT: CN

```
    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;
```

사례 3: 다차원 데이터 피벗 분석

```sql
-- 학과-과목-성적 피벗 분석
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;

-- 결과 조회
SELECT * FROM pivot_analysis
WHERE college IS NOT NULL AND course_id IS NOT NULL
ORDER BY college, avg_score DESC;

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

종합 연습 문제

연습 1: 학생 성적 관리 시스템

요구사항:

1. 각 학생의 학번, 이름, 수강 과목 수, 평균 점수, 최고점, 최저점을 표시하는 뷰를 생성하세요
2. 평균 점수 상위 5 명의 학생 상세 정보를 조회하세요
3. 각 학과의 우수율(평균 점수>=90 인 비율)을 통계내세요
4. 불합격 학생 명단을 로컬 파일로 출력하세요

```sql
-- 여기에 SQL 문을 작성하세요
-- 힌트: CREATE VIEW, JOIN, GROUP BY, HAVING, ORDER BY, LIMIT 등 사용
```

연습 2: 과목 수강 분석

요구사항:

1. 각 과목의 수강 인원, 평균 점수, 합격률을 조회하세요
2. 가장 인기 있는 3 개 과목(수강 인원이 가장 많은)을 찾으세요
3. 가장 어려운 3 개 과목(평균 점수가 가장 낮은)을 찾으세요
4. 각 과목의 점수 분포(점수 구간별 인원수)를 표시하는 보고서를 작성하세요

```sql
-- 여기에 SQL 문을 작성하세요
```

综合练习题

练习 1: 学生成绩管理系统

要求:

1. 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
2. 查询平均分排名前 5 的学生详细信息
3. 统计每个学院的优秀率(平均分>=90 的比例)
4. 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:

1. 查询每门课程的选修人数、平均分、及格率
2. 找出最受欢迎的 3 门课程(选修人数最多)
3. 找出最难的 3 门课程(平均分最低)
4. 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
```

연습 3: 데이터 품질 검사

요구사항:

1. 시험 기록이 없는 학생이 있는지 확인하세요
2. 학생이 없는 과목이 있는지 확인하세요
3. 이상 성적(음수 또는 100 점 초과)이 있는지 확인하세요
4. 각 테이블의 레코드 수와 주요 필드의 NULL 값 현황을 통계내세요

```sql
-- 여기에 SQL 문을 작성하세요
```

练习 3: 数据质量检查

要求:

1. 检查是否有学生没有考试记录
2. 检查是否有课程没有学生选修
3. 检查是否有异常成绩(如负分或超过 100 分)
4. 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
```

실습 정리

지식 요점 복습

1. 데이터 타입: 기본 타입, ARRAY, MAP, STRUCT
2. 테이블 타입: 내부 테이블, 외부 테이블, 파티션 테이블
3. 연산자: 산술, 비교, 논리 연산자
4. 함수 종류:
   - 문자열 함수
   - 날짜/시간 함수
   - 조건 함수
   - 집계 함수
   - 윈도우 함수
5. 쿼리문:
   - WHERE 조건 필터링

实践总结

知识点回顾

1. 数据类型: 基本类型、ARRAY、MAP、STRUCT
2. 表类型: 内部表、外部表、分区表
3. 运算符: 算术、比较、逻辑运算符
4. 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
5. 查询语句:
   - WHERE 条件过滤

LEFT: KR

- GROUP BY 그룹화
- ORDER BY 정렬
- JOIN 다중 테이블 연관
- 서브쿼리

6. 데이터 입출력:
- LOAD DATA
- INSERT INTO/OVERWRITE
- CREATE TABLE AS SELECT

일반적 문제 해결

문제 1: beeline 연결 실패

```bash
# HiveServer2 가 실행 중인지 확인
ps aux | grep HiveServer2

# HiveServer2 재시작
nohup hiveserver2 &
```

문제 2: 데이터 로딩 실패

```bash
# 파일 존재 여부 확인
ls -l /usr/datas/unit3/

# 파일 권한 확인
chmod 644 /usr/datas/unit3/*.txt
```

문제 3: 한글 깨짐

```sql
-- 테이블 생성 시 인코딩 지정
ALTER TABLE students SET SERDEPROPERTIES
('serialization.encoding'='UTF-8');
```

모범 사례 권장사항

RIGHT: CN

- GROUP BY 分组
- ORDER BY 排序
- JOIN 多表关联
- 子查询

6. 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

1. 명명 규칙: 의미 있는 테이블명과 컬럼명 사용
2. 주석 습관: 테이블과 컬럼에 상세한 주석 추가
3. 파티션 전략: 합리적인 파티션 사용으로 쿼리 효율 향상
4. 데이터 백업: 중요한 데이터 작업 전 백업
5. 성능 최적화:
   - 파티션 필터링 사용
   - SELECT * 지양
   - 합리적인 JOIN 사용
   - 적절한 WHERE 필터 조건 사용

1. 命名规范: 使用有意义的表名和列名
2. 注释习惯: 为表和列添加详细注释
3. 分区策略: 合理使用分区提高查询效率
4. 数据备份: 重要数据操作前先备份
5. 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

확장 학습 자료

1. Hive 공식 문서: https://hive.apache.org/
2. Hive SQL 구문 참조
3. 빅데이터 기술 블로그 및 커뮤니티
4. 실제 프로젝트 사례

扩展学习资源

1. Hive 官方文档: https://hive.apache.org/
2. Hive SQL 语法参考
3. 大数据技术博客和社区
4. 实际项目案例

실습 종료

Hive 제 3 장 종합 실습을 완료하신 것을 축하합니다!

본 실습을 통해 다음을 습득하셨습니다:

- Hive 의 기본 문법과 데이터 타입
- 각종 연산자와 함수 사용법
- 복잡한 쿼리와 다중 테이블 연관
- 데이터 입출력 방법

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

LEFT: KR

다음 단계 권장사항:

1. 본 실습의 핵심 내용 복습
2. 종합 연습 문제 독립적 완성
3. 실제 프로젝트에 학습한 지식 적용
4. Hive 의 고급 기능 계속 학습(이후 장)

실습 코드와 노트를 꼭 저장하세요!

교수 평가란:

- 완성도: □우수 □양호 □합격 □보완필요
- 평가: _____
- 날짜: _____

RIGHT: CN

下一步建议:

1. 复习本次实践的重点内容
2. 独立完成综合练习题
3. 尝试在实际项目中应用所学知识
4. 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: _____