# 데이터베이스 시스템 5장



# 5.1 프로그래밍 언어에서 SQL 접근

1. 동적 SQL
    a. JDBC, ODBC

2. 내장 SQL

# 5.1.1 JDBC

- Java 프로그램이 데이터베이스에 접속할 수 있는 응용 프로그램 인터페이스

## 5.1.1.1 데이터베이스 접속

- JDBC를 지원하는 DBMS는 Java에서 데이터베이스로 접근하기 위해서 동적으로 적재되어야 하는 JDBC 드라이버를 제공함
  - 적절한 드라이버가 다운로드되면 getConnection에서 필요한 드라이버를 찾을 것
  - jdbc:mysql: ⇒ mysql이 지원하는 특정 규약 명시

## 5.1.1.2 데이터베이스 시스템에 SQL 전달

- Statement  클래스를 통해서 SQL 구문을 전달

## 5.1.1.3 예외 및 자원 관리

- JDBC 동작에 따른 예외 처리 필요
- Connection, Statement 자원을 적절하게 정리해야함

## 5.1.1.4 질의 결과 검색

- ResultSet과 next() → getString(칼럼명) or getFloat(위치)로 조회 가능

## 5.1.1.5 Prepared statement

- 몇 개의 값을 ?로 대체하여 준비된 구문을 만들 수 있음
  - 실제 값이 나중에 제공되도록 명시
- setType(location, value)를 설정함
- SQL 삽입과 같은 공격을 막아줌
  - 이스케이프 문자열이 있어서 SQL 삽입 방지 가능
  - select * from instructor where name = 'X\' or \'Y\' = \'Y'
- 문자열을 직접 이어붙이는 것이 아닌 이러한 질의를 통해서 데이터베이스에 질의해야 보안을 지킬 수 있음

## 5.1.1.6 Callable statement

- 프로시저나 함수를 호출하기 위한 인터페이스

#### 5.1.1.7 메타데이터 특성

- ResultSetMetaData를 반환하는 getMetaData()로 데이터베이스 메타데이터 획득 가능
  - 열 개수, 열의 이름, 열의 타입
- getColumns(), getTables() 등으로 메타데이터 획득 가능

#### 5.1.1.8 다른 특징

- ResultSet 갱신 가능한 결과 집합을 제공
- 질의를 통한 결과 집합에서 갱신 가능
- setAutoCommit(False)로 자동 커밋을 비활성화 할 수 있음
  - 단, 트랜잭션을 롤백하거나 커밋해야 함

### 5.1.3 ODBC

- 응용이 데이터베이스에 접속을 열고 질의와 갱신을 전송하고 결과를 얻어 가이 위해 사용하는 표준 API
  - DBMS에서 라이브러리 제공

### 노트 5.1 내장형 데이터베이스

- 데이터베이스 추상화를 위한 응용 프로그램 안의 데이터베이스
  - HSQLDB, SQLite, H2DB

## 5.2 함수와 프로시저

- 비즈니스 규칙을 데이터베이스 외부에 저장하여 프로그래밍 언어 프로시저로 인코딩 하는 것에 비해 데이터베이스 안의 프로시저에 저장하는 것이 더 좋음
  - 다수의 응용이 접근 가능
  - 응용 변경 없이 데이터베이스만 손쉬운 변경 가능

### 5.2.1 SQL 함수 및 프로시저의 선언과 호출

```
create function instructor_of (dept_name varchar(20))
    return table (
        ID varchar(5),
```

```
        name varchar(20),
        dept_name varchar(20),
        salary numeric(8, 2))
return table
    (select ID, name, dept_name, salary
    from instructor
    where instructor.dept_name = instructor_of.dept_name);

    select *
    from table(instructor_of('Finance'));
```

- 테이블 자체를 함수의 결과로 반환하는 함수도 지원 ⇒ 매개변수도 지원하기에 매개변수화된 뷰로 생각할 수도 있음
- 프로시저도 만들 수 있음

```
create procedure dept_count_proc(in dept_name varchar(20),

    begin
        select count(*) into d_count
        from instructor
        where instructor.dept_name = dept_count_proc.dept_
    end

    declare d_count integer;
    call dept_name_proc('Physics', d_count);
```

## 5.2.2 프로시저와 함수를 위한 언어 구문

- 범용 프로그래밍 언어와 거의 같은 기능을 가진 다양한 구조 지원
- 이러한 구조를 다루는 SQL 표준의 일부분을 영구 저장 모듈이라 부름
- declare : 변수 선언
- set : 값 할당
- while, repeat, for : 반복 가능

```
while boolean expression do
    sequence of statements;
```

```
end while

repeat
    sequence of statements;
until boolean expression
end repeat

declare n integer default 0;
for r as
    select budget from department
    where dept_name = 'Music'
do
    set n = n - r.budget
end for
```

- begin ... end를 통해서 다수의 SQL 구문 배치
- if-then-else : 조건 설정 가능
- case
- 예외 조건 신호와 예외 처리 핸들러를 선언할 수 있음

```
declare out_of_classroom_seats condition
declare exit handler for out_of_classroom_seats
begin
sequence of statements
end
```

# 5.2.3 외부 언어 루틴

- SQL 표준 이전부터 프로시저와 함수를 지원했기에 대부분의 DBMS가 SQL 표준을 엄격하게 따르지 않거나 문법에서도 차이가 있음
- 프로그래밍 언어로 프로시저 정의 가능
- DB-시스템 코드와 함께 적재되어 실행됨
    - 프로그램의 버그가 있거나 접근 제어를 무시할 수 있기에 보안에 취약할 수도 있음
- 보안이 우선시 된다면 별도의 분리된 부분으로 실행시켜 매개변수를 이용해 프로세스 간의 통신을 사용할 수도 있음

- 코드를 데이터베이스 실행 프로세스 자체 내에 있는 샌드박스에서 실행할 수도 있음
  - 코드 → 코드 자신이 가지는 메모리 영역 ⇒ 접근 가능
  - 코드 → 질의 실행 프로세스 메모리, 파일 시스템 ⇒ 접근 불가능
- 질의 실행 프로세스 안의 샌드박스에서 외부 언어 루틴 실행 지원

# 5.3 트리거

- 데이터베이스에서 발생하는 특정 사건에 대한 반응으로 시스템이 자동으로 수행하는 구문
- 트리거가 실행될 시점을 명시해야 함 → 사건, 조건
- 실행될 때 수행되어야 할 동작을 명시해야 함

## 5.3.1 트리거의 필요성

- SQL의 제약 조건 방법을 사용해서 명세할 수 없는 무결성 제약 조건을 구현하기 위해 사용됨

## 5.3.2 SQL에서 트리거

- SQL 표준에 있기는 하나 데이터베이스는 비표준 문법 사용

```
create trigger timeslot_check 1 after insert on section
referencing new row as nrow
for each row
when (nrow.time_slot_id not in (
                select time_slot_id
                from time_slot)) /* time_slot_id not present

begin
    rollback
end;

create trigger timeslot_check 2 after delete on timeslot
referencing old row as orow
for each row
when (orow.time_slot_id not in (
                select time_slot_id
```

```
                from time_slot) /* last tuple for time_slot_i
        and orow.time_slot_id in (
                select time_slot_id
                from section))  /* and time_slot_id still ref

begin
    rollback
end;
```

- 삽입, 삭제, 갱신 후나 사건 전에 활성화 될 수 있음
- SQL 문에 영향을 받은 각 행에 대한 동작이 아닌 구문 전체에 단일 동작 수행 가능
  - for each statement
- SQL 문에 영향을 받는 모든 행을 포함하는 임시 테이블을 참조할 수 있음 ⇒ 이행 테이블
- 트리거를 활성화하거나 비활성화 할 수 있음
- DBMS 벤더사마다 트리거의 문법이 다름

### 5.3.3 트리거가 부적합한 경우

- 어떤 경우에는 트리거 대신 다른 기술이 더 적합할 수도 있음
- 외래 키 제약 조건과 on delete cascade를 통해서 구현할 수도 있음, 트리거는 구현을 더 어렵게 하고 데이터베이스 구조를 이해하기 쉽지 않게 만듦
- 트리거는 최대한 피하고 매우 신중히 사용되어야 함 ⇒ 프로시저로 대체하는 것이 좋음?
  - 무한 트리거, 트리거 내부의 실행 시간 오류

## 5.4 재귀 질의

### 5.4.1 반복을 통한 이행 폐포

- 반복을 통해서 재귀를 풀어내는 것
  - 어떤 과목의 선행 과목을 반복을 통해서 찾음 ⇒ 더 이상 찾을 수 없을 때까지
  - 임시 테이블을 활용

### 5.4.2 SQL에서 재귀

- 재귀적 뷰를 정의할 수 있음
  - 어떤 재귀적 뷰도 반드시 두 개의 하위 질의의 합으로 정의되어야 함
  - 기본 질의는 재귀적일 수 없음
  - 재귀적 뷰에서 집계, 하위 질의에서 not exists, 오른편에서 재귀적 뷰를 사용하는 것을 제외한 차집합 등 복잡하기에 사용해서는 안됨
- with recursive절을 이용
- 기본 질의를 계산 → 뷰 릴레이션 추가 → 재귀 질의 계산 → 반복
- 질의가 단조로울 때 사용해야 함

# 5.5 고급 집계 기능

## 5.5.1 순위화

- SQL에서는 구현이 어려워 프로그래밍 언어를 사용하고는 하지만 SQL이 지원하기는 함

```
select ID, rank over (order by(GPA) desc) as s_rank
from student_grades
order by s_rank

select ID, dept_name, rank () over (partition by dept_name or
from dept_grades
order by dept_name, dept_rank;
```

- dense_rank : 같은 값을 가져도 같은 순위를 주지 않고 차례대로 순위를 줌
- null first, null last : null 값이 우선시되므로 처리 순서 명시할 수도 있음

## 5.5.2 윈도우

- 튜플의 범위에 대한 집계 함수 계산
  - 고정된 범위에 대한 집계 계산 시 유용
  - 동향 분석

```
select year, avg(num_credits)
                            over (order by year rows 3 preced
```

```
                                as avg_total_credits
from tot_credits
```

- 2017년, 2018년, 2019년 데이터가 존재한다면 위의 쿼리는 3개년의 평균이 나옴

## 5.5.3 피벗팅

```
select *
from sales
pivot (
        sum(quantity)
        for color in ('dark', 'pastel', 'white')
)
```

- 어떤 릴레이션 R의 특정 속성(A)의 값이 속성 이름이 되는 테이블
- A를 피벗 속성이라고 함

## 5.5.4 롤업과 큐브

- cube와 rollup을 사용하여 group by 연산자의 일반화를 제공

```
select item_name, color, sum(quantity)
from sales
group by rollup(item_name, color);
```

- 위의 질의는(item_name, color), (item_name, null), (null, null)과 같이 세 개의 그룹을 만듬

```
select item_name, color, cloth_size, sum(quantity)
from sales
group by cube(item_name, color, clothes_size);
```

- 위의 질의는 (item_name, color, clothes_size), (item_name, color, null), (item_name, null, clothes_size), (null, color, clothes_size), (item_name, null, null), (null, color, null), (null, null, clothes_size), (null, null, null)과 같은 그룹들을 만듬
- rollup을 연속해서 사용하면 생성된 그룹 간의 카티션 곱이 만들어 짐

```
select item_name, color, clothes_size, sum(quantity)
from sales
group by grouping sets ((color, clothes_size), (clothes_size,
```

- grouping sets로 그룹 명시 가능