데이터베이스 보안

데이터베이스 보안 개요

데이터 베이스 보안 개요

- 안전한 데이터베이스 응용 설계
  - 보안 : 권한없는 사용자에겐 정보 누출 x
  - 무결성 : 권한있는 사용자에겐 데이터 수정 허용
  - 가용성 : 권한있는 사용자에겐 접근 제한 x
- 명확하고 일관성 있는 보안 정책의 설정 필요
- DBMS는 데이터 보안에 대한 메커니즘제공

TIP

데이터베이스 관리 시스템(DBMS)은 운영체제와는 별개로 독자적인 권한 관리 시스템을 가지고 있습니다. DBMS에서 그룹(Group)은 데이터베이스 객체(Database Object)에 대한 접근 권한을 관리하기 위한 기능 중 하나입니다. 즉, 그룹을 이용하여 여러 사용자에 대한 권한을 한번에 관리할 수 있습니다.

관리자 권한으로 SQL 모든 권한을 얻는 경우, 이는 데이터베이스 객체를 생성하거나 삭제하며, 데이터베이스의 구조를 변경하거나, 데이터를 수정, 삭제, 조회 등 모든 작업을 수행할 수 있는 권한을 갖는 것을 의미합니다. 이는 데이터베이스의 전반적인 관리와 관련된 권한으로, 보안과 관련된 작업 등을 포함합니다.

반면, 일반 권한을 가진 사용자는 보다 제한적인 권한을 갖게 됩니다. 일반 권한을 갖는 사용자는 데이터베이스 객체를 조회하거나, 데이터를 추가하는 등의 작업만 수행할 수 있습니다. 이는 일반 사용자가 데이터베이스의 일부만을 접근하고, 보안과 권한 관리를 강화하기 위한 것입니다.

따라서, 그룹을 이용하여 여러 사용자에 대한 권한을 한번에 관리하며, 관리자 권한으로는 데이터베이스의 전반적인 관리 작업을 수행할 수 있고, 일반 권한을 가진 사용자는 보다 제한적인 권한을 갖게 됩니다. 이를 통해 데이터베이스의 보안과 권한 관리를 보다 효율적으로 수행할 수 있습니다.

LDAP

LDAP는 Lightweight Directory Access Protocol의 약자로, 인터넷 프로토콜 스택에서 사용되는 디렉터리 서비스 프로토콜입니다.

LDAP는 디렉터리 서비스에 접근하기 위한 표준 프로토콜로서, 디렉터리 서비스는 사용자, 컴퓨터, 네트워크 장치 등의 정보를 저장하고 조회할 수 있는 중앙집중형 데이터 저장소입니다. 디렉터리 서비스는 데이터를 계층 구조로 구성하며, 트리 구조를 이용하여 데이터를 효율적으로 저장하고 검색할 수 있습니다.

LDAP 프로토콜은 이러한 디렉터리 서비스에 접근하기 위해 사용됩니다. LDAP를 사용하면 클라이언트는 디렉터리 서비스로부터 정보를 검색하거나 수정할 수 있으며, 네트워크에 연결된 다른 장치에서도 사용할 수 있습니다.

LDAP는 인증 및 권한 부여, 주소록 관리, 사용자 인증 등의 목적으로 널리 사용되며, 인증 및 권한 부여에 있어서는 현재도 많이 사용되고 있습니다. 예를 들어, 사용자 계정 정보를 디렉터리 서비스에 저장하고, 사용자 인증을 LDAP 프로토콜을 이용하여 처리하는 경우가 많습니다. LDAP는 OSI 참조 모델의

```
X.500 디렉터리 서비스를 기반으로 하지만, 더 가볍고 단순한 프로토콜로 설계되어 있습니다.


MySQL에서 LDAP은

 보통 MySQL 사용자 인증 및 권한 부여를 구성하는 데 사용됩니다.

MySQL은 기본적으로 내장 인증 및 권한 부여 시스템을 가지고 있지만, 대규모 조직에서는 LDAP을 사용하여 중앙 집중식 인증 및 권한 부여 시스템을 유지하는 것이 효과적일 수 있습니다.

LDAP을 사용하여 MySQL 인증 및 권한 부여를 구성하면, MySQL 서버는 LDAP 서버로부터 사용자 인증 정보와 권한 정보를 검색합니다.

이를 위해 MySQL은 LDAP 인증 플러그인을 제공합니다. 이 플러그인을 사용하여 MySQL 서버는 LDAP 서버에 연결하여 사용자 인증 정보를 검색하고, 사용자가 MySQL 데이터베이스에 액세스할 수 있는 권한을 확인할 수 있습니다.

또한 MySQL에서는 PAM (Pluggable Authentication Modules)을 사용하여 LDAP 인증을 구성할 수도 있습니다. 이를 통해 MySQL 사용자 인증을 외부 인증 시스템으로 연결할 수 있으며, 다양한 인증 메커니즘을 사용할 수 있습니다.
```

USER 개체

- 데이터 베이스 관리 시스템은 시스템에 대한 모든 권한을 가지는 관리 사용자를 가짐
  - DBMS에 따라 관리 사용자의 종류가 세분화 되는 방식이 다름
- 데이터 베이스 관리 시스템은 데이터베이스의 사용자를 생성/관리
  - 운영체제와 통합되거나, 독립적으로 관리
  - 데이터베이스의 사용자는 데이터베이스에 대한 여러 형태의 권한을 가짐
- 사용자 조회

```
  SELECT host, user FROM mysql.user;
IDENTIFID는 패스워드 설정하는 명령어
```

- 사용자 생성

```
CREATE USER <User ID> IDENTIFIED BY '<Password>'
내건
create user Celline identified by 'C1ell@i#ne'

IDENTIFID는 패스워드 설정하는 명령어
```

접근제어

- 대부분의 DB 사용자들은 자신의 업무를 수행하는데 필요한 데이터에만 접근
- DBMS는 크게 두 가지의 접근 제어 방식을 제공

- 재량 접근 제어(Discretionary Access Control)
  - 특권(Privileges)이라는 접근 권한의 개념에 기반하며, 사용자에게 특권을 부여하는 메커니즘 사용
- 필수 접근 제어 (Mandatory Access Control)
  - 사용자 개개인에 따라 변경할 수 없는 시스템 전반에 걸친 정책 사용
  - 모든 데이터베이스 개체에 비밀 등급을 매기고 사용자들마다 허가 등급을 부여
- SQL 1999 표준은 필수 접근 제어를 제공하지 않음

tip

```
권한 : 삽입, 삭제, 수정
특권 : 사용자에게 권한을 부여하고, 권한을 뺏고
```

재량 접근 제어

- SQL 1999는 GRANT 명령과 REVOKE 명령을 통해 재량 접근 제어를 지원
- GRANT
- 사용자에게 기반 테이블이나 뷰에 대한 특권을 부여

```
GRANT <특권> ON <개체> TO <사용자> [WITH GRANT OPTIONS]
grant select on Module02.Aircraft to Celline;

mysql -u Celline -p를 통해 select 명령어로 확인가능
update insert  명령어를 쓰면 권한 거부
```

- REVOKE
  - 사용자에게 부여된 특권 취소

```
  REVOKE <특권> ON <개체> FROM <사용자> [CASCADE]
  REVOKE SELECT ON Module02.Aircraft from Celline;
```

특권

- 특정 조치 또는 작업을 실행하는데 필요한 권한
- 권한이 부여된 사용자는 개체를 작성하고, 소유한 개체에 접근하며, GRANT문을 사용해 소유한 개체에 대한 권한을 다른 사용자에게 전달할 수 있음

뷰

- 관계 데이터베이스 관리 시스템의 매우 강력한 기능의 하나
- 접근 제어 향상, 논리적 데이터 독립성을 제공
  - 데이터베이스 개념 스키마의 변경을 외부 사용자에게 감추는 역할
  - 보안관점에서만 작성된 개체 형태는 아님
- 테이블과 같은 방식으로 동작하지만, 소속 투플이 실제로 데이터베이스에 저장된 것이 아님
- 필요할 때마다 뷰 정의에 따라 계산됨

- 질의를 수행하거나 다른 뷰를 생성할 때 기존 뷰는 테이블과 같은 형태로 사용할 수 있음

```
-- SQL 뷰 실습
USE MODULE (AIRCRAFT가 담긴 모듈)
create view ReserveInfo
AS
SELECT P.PASSENGERNO, P.PASSENGERNAME,P.GRADE,P.AGE,R.RESERVEDDATE,
F.FLIGHTNO  FROM PASSENGER AS P INNER JOIN RESERVATION AS R ON
P.PASSENGERNO = R.PASSENGERNO
    INNER JOIN FLIGHT AS F ON F.FLIGHTNO=R.FLIGHTNO;
    SELECT * FROM RESERVEINFO;

select * from ReserveInfo;
```

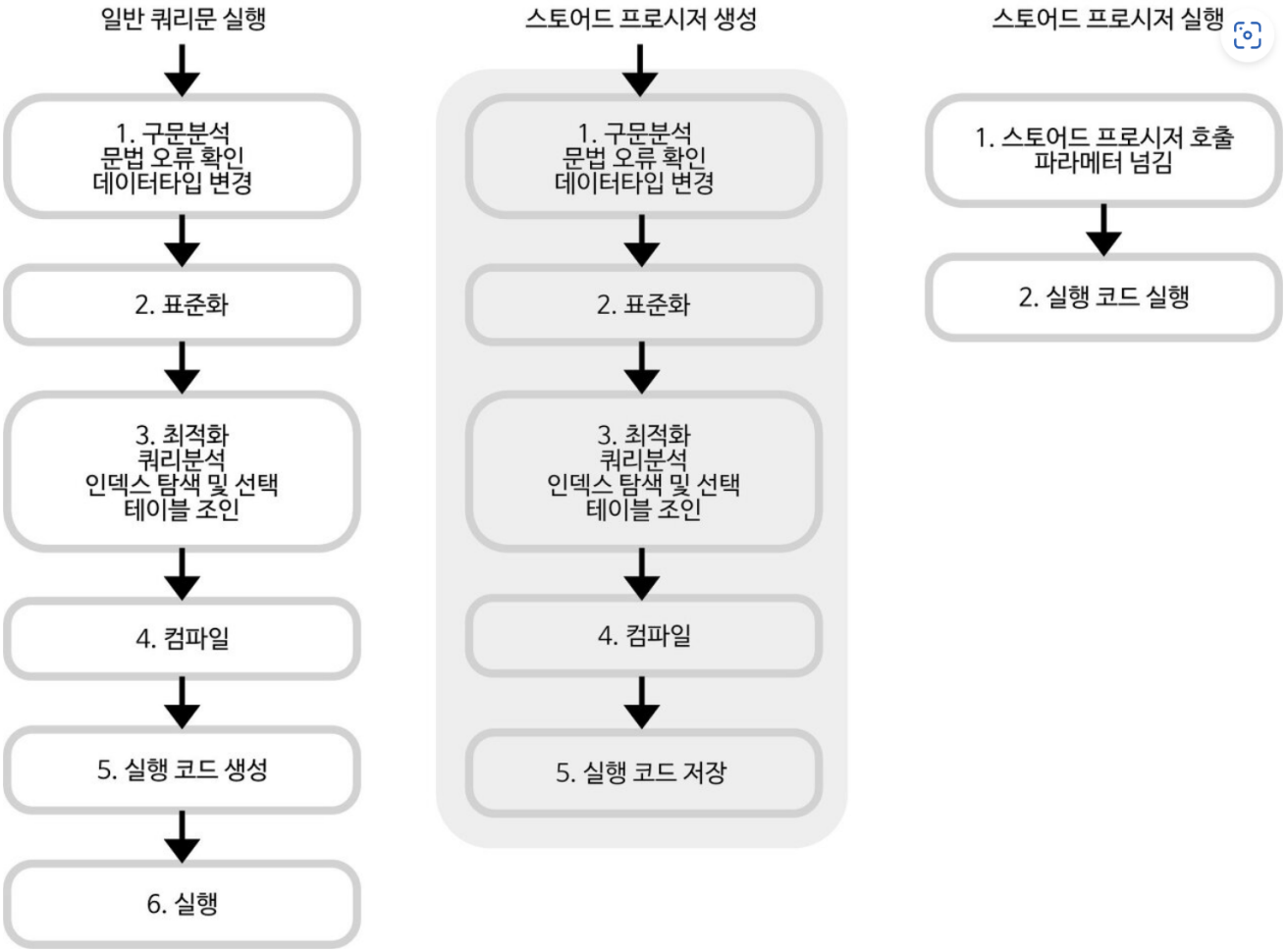
```
-- ROOT
GRANT SELECT ON Module02.ReserveInfo to Celline;
--- Celline
SELECT * FROM ReserveInfo;
```

TIP

```
뷰를 쓰는 이유는 join 이 계속 되는 부분은 간단하게 select * from view로 출력할수 있기때문에 사용한다.

뷰에 데이터 삽입하는 방법도 있다고함. 대신 원래 테이블에는 안담김
근데 여기서 무결성이 깨질 수도 있다고함

그래서 둘중 하나를 기준으로 동기화를 해야함.
그 기능은 Mview(Materialized Views)

oracle은 가능
mysql은 안됨
```

프로시저

프로시저란?



- 일련의 쿼리를 모아 하나의 함수처럼 실행하기 위한 쿼리의 집합이다.
- 장점
    - 하나의 요청으로 여러 SQL문 실행(네트워크에 대한 부하 감소)
    - 미리 구문 분석 및 내부 중간 코드로 변환을 끝내야하므로 처리 시간이 줄어든다
    - 데이터베이스 트리거와 결합해 복잡한 규칙에 의한 데이터의 참조 무결성 유지가 가능해짐 즉 응용프로그램측 로직을 가지지 않고도 데이터베이스의 데이터 앞뒤가 맞게 될 수 있다.
- 단점
    - 코드의 자산으로써 재사용성이 나쁨
    - 업무의 사양 변경 시 외부 응용프로그램과 함께 프로시저의 정의를 변경할 필요가 있다. 파라미터 타입은 “IN”, “OUT”, “INOUT” 중 하나입니다. “IN”은 스토어드 프로시저 안에서 사용할 변수 값을 파라미터로 받는 것이고, “OUT”은 스토어드 프로시저에서 생성한 값을 스토어드 프로시저 밖으로 전달할 대 사용합니다.
      “INOUT”은 입출력에 모두 사용하는 변수에 사용합니다. 파라미터 타입이 없으면 “IN” 타입으로 정의됩니다.
- 요소
    - DELIMITER : 프로시저 앞, 뒤에 위치해 안에 있는 부분은 한번에 실행될 수 있게 하는 역할
    - CREATE PROCEDURE [프로시저 이름](IN [데이터] [데이터타입],OUT [데이터] [데이터타입])
    - IN, OUT, INOUT
    - IN : 사용할 변수값을 파라미터로 받음
    - OUT : 스토어드 프로시저에서 생성하 값을 프로시저 밖으로 전달할 때 사용

- INOUT : 입출력에 모두 사용하는 변수에 사용
- BEGIN - END : 프로시저 실행 내용 정의
- CALL 프로시저명(매개변수들)
- SHOW CREATE PROCEDURE 프로시저명 : 프로시저 내용 조회
- DECLARE EXIT HANDLER FOR : 자바의 try-catch와 비슷

```
DROP PROCEDURE IF EXISTS calculate_grade;
PROCEDURE 확인 하기
SELECT routin_name, routine,definition FROM INFORMATION_SCHEMA.ROUTINES
    WHERE routin_schema = '데이터베이스명' AND routine_type = 'PROCEDURE';

DELIMITER $$
CREATE PROCEDURE calculate_grade(
  IN in_mid DOUBLE, -- 매개변수는 IN으로 가져온다.
  IN in_end DOUBLE,
  IN in_att INT,
  IN in_rep INT,
  IN in_class_num INT,
  IN in_student_num INT )

BEGIN
  -- 지역 변수 선언
  DECLARE total DOUBLE DEFAULT 0;
  -- double total = 0; 과 같다고 보면 된다.
  DECLARE grade VARCHAR(2);

  -- 변수 초기화
  SET total = in_mid + in_end + in_att + in_rep; -- 인자를 다 더한다.

  -- 분기
  IF total >= 95 AND total <= 100 THEN
    SET grade = 'A+';

  ELSEIF total >=90 AND total < 95 THEN
    SET grade = 'A';

  ELSEIF total >=85 AND total < 90 THEN
    SET grade = 'B+';

  ELSEIF total >=80 AND total < 85 THEN
    SET grade = 'B';

  ELSEIF total >=75 AND total < 80 THEN
    SET grade = 'C+';

  ELSEIF total >=70 AND total < 75 THEN
    SET grade = 'C';

  ELSEIF total >=65 AND total < 70 THEN
    SET grade = 'D+';

  ELSEIF total >=60 AND total < 65 THEN
```

```
    SET grade = 'D';

  ELSEIF total >=0 AND total < 60 THEN
    SET grade = 'F';
  END IF;

  -- 쿼리문
  UPDATE university.course
  SET
    course_mid = in_mid,
    course_end = in_end,
    course_report = in_rep,
    course_attendance = in_att,
    course_total = total,
    course_grade = grade
  WHERE course_student_num = in_student_num AND course_class_num =
in_class_num AND course_num >= 1;

END $$
DELIMITER ;
```

DBMS 질의 실행 순서
1. 문법 검사
2. 개체 요인성 검사?
3. 컴파일
4. 실행할수 있는 버킷이 나옴

4만 만들면 되게 빨라짐, 1,2,3은 뺴버림

쿼리를 짜면 비슷한 유형의 쿼리만 나옴 바뀌는 값에대한 값만 처리해주면 됨
메서드를 하나만들어놓고 메서드안에다가 다른 메서드를 부르고
또 순차적으로 실행도되고
반복문도 만들수 있겠네
while반복,
for반복을 지원하는 dbms는 오라클

스토어프로시저
프로시저를 한번 만들면 질의 실행순서의 1, 2, 3은 생략가능
create 프로시저

프로시저는 기본 ;으로 끝나던 딜리미트? 를 바꿀수 있다

프로시져 실습

```
DELIMITER $$
create PROCEDURE GetReserveInfo()
```

```
BEGIN
SELECT P.PASSENGERNO, P.PASSENGERNAME,P.GRADE,P.AGE,R.RESERVEDDATE,
F.FLIGHTNO  FROM PASSENGER AS P INNER JOIN RESERVATION AS R ON
P.PASSENGERNO = R.PASSENGERNO
    INNER JOIN FLIGHT AS F ON F.FLIGHTNO=R.FLIGHTNO;

END $$
DELIMITER ;


DELIMITER $$
create PROCEDURE GetReserveInfoForPassenger(
passenger int
)
BEGIN
SELECT P.PASSENGERNO, P.PASSENGERNAME,P.GRADE,P.AGE,R.RESERVEDDATE,
F.FLIGHTNO  FROM PASSENGER AS P INNER JOIN RESERVATION AS R ON
P.PASSENGERNO = R.PASSENGERNO
    INNER JOIN FLIGHT AS F ON F.FLIGHTNO=R.FLIGHTNO
    Where p.PassengerNo=passenger;

END $$
DELIMITER ;


call GetReserveInfo();
call GetReserveInfoForPassenger(1);
```