데이터베이스 시스템 4장

태그	데이터베이스 시스템

4.1 조인 표현식

4.1.1 자연 조인

```
select name, course_id
from student, takes
where student.ID = takes.ID;

select name, course_id
from student
natural join takes;

select name, title
from (student natural join takes) join course using (course_i
```

- 두 개의 릴레이션에 대해 수행되고 하나의 릴레이션을 결과로 생산
    - 속성값이 같은 튜플의 짝만 고려
    - 속성이 반복되지 않고 한 번만 표현됨

처음에는 공통 속성, 이후 첫 번째 릴레이션 속성, 마지막으로 두 번째 릴레이션 속성 나열

4.1.2 조인 조건

```
select *
from student
join takes on student.ID = takse.ID
```

```
select student.ID as ID, name, dept_name,
            tot_cred, course_id, semester, year, grade
from student
join takes on student.ID = takes.ID;
```

- on 절의 속성이 릴레이션마다 나타남
- 왜 where 대신 on 을 쓰는가?
  - 외부 조인의 경우 on 조건은 where 조건과 다른 방식으로 동작
  - on 절과 where 절을 나눠서 쓰면 SQL 질의를 이해하기 쉬움

4.1.3 외부 조인

- 몇몇 튜플은 자연 조인에서 빠지게 됨
  - 조인 조건을 만족하지 못하기 때문
- 외부 조인 조건으로 위의 문제를 해결할 수 있음
  - 드라이빙 테이블의 속성은 포함하고 조인 조건을 만족하지 않는 드라이븐 테이블의 속성값들은 널 값으로 대체
- 세 가지 유형의 외부 조인
  - 왼쪽 외부 조인: left outer join 왼쪽에 나타난 릴레이션의 튜플만 보존
  - 오른쪽 외부 조인: right outer join 오른쪽에 나타난 릴레이션의 튜플 보존
  - 전체 외부 조인: 두 릴레이션의 모든 튜플 보존
- where 절과 on 절은 외부 조인에서 다르게 동작
  - on 절은 조건만 만족하지 않더라도 null 값이 채워짐
  - where 절은 조건을 만족하지 못하면 필터링되어 결과에 나오지 않음

4.1.4 조인의 종류와 조인 조건

- outer가 붙지 않은 조인은 모두 내부 조인
  - 자연 조인 = 자연 내부 조인

4.2 뷰

- 가상 릴레이션은 미리 계산해서 저장하는 것이 아닌 가상 릴레이션을 사용할 때 질의를 수행해서 결과를 얻어냄
    - with 절을 사용할 수도 있음, 해당 질의에서만 국한됨
    - 뷰를 정의해서 단일 질의 이상으로 개념을 확장할 수도 있음

4.2.1 뷰 정의

- create view v as <query expression>;
- 튜플을 미리 계산하고 저장하는 것은 아님
    - 접근할 때마다 질의 결과를 계산하여 튜플 생성

4.2.2 SQL 질의에서 뷰 사용

```
create view departments_total_salary(dept_name, total_salary)
    select dept_name, sum(salary)
    from instructor
    group by dept_name;
```

- 뷰를 정의를 저장하고 뷰 릴레이션이 나타낼 때마다 저장된 질의 표현식을 바꿔 넣음
    - 질의 평가시마다 재계산됨
- 뷰를 이용해 또 다른 뷰를 정의할 수 있음

4.2.3 실제화된 뷰

- 뷰 릴레이션을 저장하는 방법
    - 릴레이션 수정에도 뷰는 최신 상태 유지
    - 훨씬 빠른 계산 결과
- 실체화된 뷰를 관리해줘야 함
    - 뷰가 참조하는 릴레이션에 데이터가 추가되면 갱신 필요

4.2.4 뷰의 갱신

- 질의에는 유용하지만 갱신, 삽입, 삭제 등의 연산에 뷰를 사용하면 많은 문제점이 생김
- 뷰에서 릴레이션으로의 표현 수정이 문제가 됨

    - 뷰가 여러 릴레이션과 조인된 경우 insert를 하면 어떻게 해야하는가?
- 일부 경우를 제외하고는 뷰 릴레이션에 대한 변경을 허용하지 않음
    - 뷰 릴레이션의 갱신을 허용하는 조건은 벤더사마다 다름
- 아래를 만족하면 뷰를 갱신가능이라고 함
    - from 절은 오직 한 개의 데이터베이스 릴레이션
    - select 절은 오직 릴레이션의 속성 이름만 포함해야 하며 표현, 집계, distinct 명세를 가져서는 안됨
    - select 절에 나열되지 않은 어떤 속성도 널 값으로 될 수 있어야 함 ⇒ not null, 기본키 일부 안됨
    - group by, having 절은 없어야 함
- with check option 으로 뷰에 불필요한 데이터가 삽입되는 것을 막음
    - 갱신 역시 where 절의 조건을 만족하지 않는다면 실행되지 않음
- 뷰를 통해 데이터베이스를 수정하는 대안은 트리거가 있음

4.3 트랜잭션

트랜잭션은 다음의 SQL 문 중 하나로 끝나야 함(work 생략 가능)

- Commit work: 현재 수행 중인 트랜잭션을 커밋
    - 모든 갱신 영구 반영
    - 커밋 수행한 경우 롤백에 의해 더는 취소될 수 없음
- Rollback work: 현재 수행 중인 트랜잭션을 롤백
    - 모든 갱신 취소
    - 모든 시스템 장애 상황에서 커밋 전에 롤백을 보장

트랜잭션의 성질

모든 동작이 수행되어서 커밋을 하거나, 모든 동작을 수행하지 못한 경우 롤백을 수행함으로써 트랜잭션의 원자적인 성질을 보장

SQL문과 트랜잭션

- MySQL과 PostgreSQL을 포함한 많은 구현에서 기본적으로 각 SQL 문은 트랜잭션이 됨

    - 그것이 수행되자마자 커밋이 됨
    - begin ... commit or rollback 가능
- set autocommit off로 자동 커밋 해제 가능

4.4. 무결성 제약 조건

- 권한이 주어진 사용자로부터의 데이터베이스 변경이 데이터 일관성에 손실을 초래하지 않음을 보장함
    - not null, 종복 없음. 특정 조건

4.4.1 단일 릴레이션에 관한 제약 조건

- create table을 통해서 설정 가능

4.4.2 Not Null 제약 조건

- name varchar(20) not null
    - 해당 속성에 널 값의 삽입을 금지함
    - 도메인 제약 조건
    - 주 키는 명시적으로 not null 을 선언하지 않아도 됨

4.4.3 Unique 제약 조건

- unqiue(Aj1, Aj2,...,Ajm)
- 어떠한 두 개의 튜플도 나열된 속성의 값과 같을 수 없음
    - not null 이 없는 한 null 값을 가질 수 있음

4.4.4 Check 절

- check(P)
    - check semester in ('Fall', 'Winter', 'Spring', 'Summer')
    - 모든 튜플이 충족해야 하는 술어 P를 명시
    - unknwon 은 참/거짓을 판별할 수 없으므로 조건 위반이 아님, not null 이 필요하면 명시해야 함

4.4.5 참조 무결성

- 한 릴레이션의 값이 또 다른 릴레이션의 특정한 속성 집합에 반드시 나타내야 하는 경우를 말함
- 외래 키는 참조된 속성이 참조된 릴레이션의 주 키를 이루는 참조 무결성 제약의 한 형태
- foreign key (property_name) refrences table_name(property_name)
  - 외래 키는 참조되는 테이블의 주 키 속성을 참조
  - 모든 DBMS는 주 키가 아닌 키를 참조할 수 있는 기능을 제공하지는 않음
- 참조 무결성이 위배되었을 때 기본적으로 그 연산을 거부할 수 있지만 연관된 튜플에 같이 적용할 수 있는 방법도 있다
  - on delete cascade, on update cascade
  - 참조되는 키가 바뀌면 연쇄적으로 삭제되거나 업데이트 됨
  - 연쇄적으로 적용되는 연산을 수행할 수 없으면 트랜잭션이 롤백됨

4.4.6 제약 조건 명명

- 무결성 제약 조건에 이름을 할당할 수 있음
- salary numerice(8, 2), constraint minsalary check (salary >29000)

4.4.7 트랜잭션 수행 중 무결성 제약 조건 위반

- A 테이블이 B 테이블의 속성을 참조
- A 테이블에 B 테이블에 없는 튜플을 삽입하면 오류가 발생
- 이를 연기가능을 통해서 A 테이블에는 참조하는 속성에 null을 넣고 B 테이블에 추후 삽입을 통해서 제약 조건 판단을 트랜잭션 마지막에 수행할 수 있음
  - 이론상 가능하나 많은 DBMS 벤더사에서는 지원하지 않음
  - 비용이 많이 들고 까다롭기 때문임, 만약 어떤 A 테이블에 B 테이블을 참조하는 속성이 not null 이라면?

4.4.8 복잡한 Check 조건과 주장

- 무결성 제약 조건을 명시할 수 있는 추가적인 구조
- 많은 DBMS에서는 지원되지 않음
  - 이 조건을 판단하는데 비용이 너무 비쌈

```
check (time_slot_id in (select time_slot_id from time_slot))
```

- assertion 을 통해서 데이터베이스가 항상 만족하길 원하는 조건을 표현할 수 있음
- create assertion <assertion-name> check <predicate>

```
create assertion credits_earned_constraint check
(not exists (select ID
                        from student
                        where tot_cred <> (select coalesce(s
```

- 복잡한 assertions은 시스템의 상당한 부하를 일으킬 수도 있음
- 위의 구문을 지원하는 DBMS는 거의 없음
- 트리거를 이용해서 구현 가능

4.5 SQL의 데이터 타입과 스키마

4.5.1 SQL에서 날짜와 시간 타입

- date: 연도, 월, 일로 구성되는 날짜
- time: 시, 분, 초로 나타낸 그날의 시간
  - time(p) 는 초를 위한 소수 단위까지 명시하는데 사용됨
  - time with timezone
- timestamp: date와 time의 조합
  - timestamp(p) 는 초를 위한 소수 단위까지 명시하는데 사용됨
  - timestamp with timezone으로 시간대 정보 또한 저장 가능
- 현재 날짜와 시간을 얻기 위한 유용한 함수 제공
  - current_date, current_time, localtime, current_timestamp, localtimestamp
- 날짜 비교 연산 지원

4.5.2 타입 변환 및 서식 함수

- cast(Id as numeric(5)) 와 같이 타입 변환 가능
- format, to_char, to_number, to_date, convert 와 같이 DBMS 벤더사가 제공하는 형식 지정함수 사용 가능
- null 인 경우 해당 필드를 비우도록 하지만 coalesce 와 같이 값을 대체할 수도 있음

4.5.3 기본값

- tot_cred numeric(3, 0) default 0
- 삽입 시 값을 안 빼먹고 넣을 수 있음

4.5.4 대형 객체 타입

- clob, blob 과 같은 대형 객체 데이터 타입 제공
  - image blob (10MB)
- 효율을 위해서 쿼리 시 해당 데이터 전체가 아닌 위치만 가져오도록 함

4.5.5 사용자 정의 타입

- DBMS 벤더사마다 지원 여부가 다를 수도 있음
- 두 가지 타입이 존재
  1. 고유 타입
  2. 정형 데이터 타입(중첩 레코드 타입, 배열, 다중 집합)
- distinct type을 통해서 구분 가능

사용자 정의 타입

- create type Dollars as numeric(12, 2) final;
  - 연산을 위해서 타입을 캐스팅 해야함

사용자 정의 타입과 유사하지만 미묘하게 다른 도메인 타입

- create domain DDollars as numeric(12, 2) not null;
- 도메인과 타입은 다름
  1. 도메인은 not null과 같은 제약 조건을 가질 수 있지만 사용자 정의 타입은 그럴 수 없음

2. 도메인은 엄격하게 타입을 지킬 필요가 없음

4.5.6 고유 키 값 생성하기

- ID number(5) generated always as identity
  - MySQL에서는 auto increment
- create sequence로 시퀀스 카운터 객체를 이용할 수도 있음

4.5.7 Create Table 확장

```
create table temp_instructor like instructor

create table t1 as
        (select *
         from instructor
         where dept_name = 'Music')
with data;
```

4.5.8 스키마, 카탈로그, 환경

- 현재 DBMS는 3단계 계층 구조를 제공함
  1. 카탈로그
  2. 스키마
  3. 릴레이션, 뷰
- 복수 개의 카탈로그와 스키마로 릴레이션 이름 충돌을 걱정하지 않고 독립적으로 운영될 수 있음
- 카탈로그와 스키마를 통해서 각 데이터베이스 연결에 대한 SQL 환경을 만들 수 있음
  - 권한과 연결지을 수도 있음
- create schema, drop schema

4.6 SQL 인덱스의 정의

- 릴레이션의 모든 튜플을 살펴보지 않고도 효과적으로 원하는 튜플을 찾을 수 있는 자료구조

- 정확성 때문이 아닌 효율성 때문임, 물리 스키마의 일부
- 트랜잭션의 효율적인 처리에 필수적
    - 주 키, 외래 키 제약조건과 같은 무결성 제약 조건을 효율적으로 강제하기도 함
- 인덱스가 소모하는 공간과 갱신 비용 때문에 어떤 인덱스를 생성할지 결정해야 함
- create index <index-name> on <relation-name> (<attribute-list>)
- create unique index <index-name> on <relation-name> (<attribute-list>)

4.7 권한

- 데이터를 읽을 권한, 삽입할 권한, 갱신할 권한, 삭제할 권한 ⇒ 특권

4.7.1 특권 부여 및 취소

- 릴레이션 생성한 사용자는 모든 특권을 자동 부여 받음

```
grant <privilege list>
on <relation name or view name>
to <user/role list>;

grant update(budget) on department to Amit, Satoshi;

revoke <privilege list>
on <relation name or view name>
from <user/role list>
```

- update, insert는 특정 속성만 적용될 수 있도록 할 수 있음
- 튜플에 대한 권한 부여는 불가능

4.7.2 역할

- 사용자에게 부여하는 방식과 같이 역할에 권한을 부여할 수 있음
- `create role instructor;`
- `grant select on taks to instructor;`

4.7.3 뷰에 대한 권한

- 해당 뷰가 참조하는 릴레이션의 권한을 확인

4.7.4 스키마에 대한 권한

- `grant references (dept_name) on department to Mariano;`
  - 외래 키 생성 권한

4.7.5 특권 양도

- grant select on department to Amit with grant option;
  - 특권 양도 가능

4.7.6 특권 취소

- 특권이 연쇄적으로 취소되지 않으면 문제가 있을 수도 있음
  - 사용자끼리 특권을 주고 받으면 특권이 계속 있을 수도 있음
- `revoke select on department from Amit, Satoshi restrict/cascade`
  - restirct : 연쇄 취소 방지
  - cascade : 기본값, 연쇄 취소 허용
- granted by current_role
  - 특권은 연쇄적으로 취소되면 기존에 권한이 박탈될 수도 있기 때문에 현재 세션에 역할로 설정된 사용자게에 특권을 부여할 수 있음

4.7.7 행 수준 권한

- DBMS 벤더사마다 지원 방식이 다름
- 질의의 의미가 크게 변경될 수도 있음 ⇒ 내가 볼 수 있는 행만의 통계를 낸다면?

- 하나의 타임존을 두고 거기를 맞춰서 대응하거나 업데이트하자
- IANA 타임존 데이터베이스
- 여러 타임존을 다루면 머리가 아프다
  - 정기결제의 경우 매월 초에 해줘야 한다면 어느 나라에 맞춰야 하는가?

참고자료

https://vladmihalcea.com/date-timestamp-jpa-hibernate/

https://medium.com/finda-tech/mysql-timestamp-와-y2k38-problem-d43b8f119ce5