Transaction, Sequence, View

Transaction

정의

- 전체 일 처리가 **완결**되어야만 의미가 있는 경우
  - 서비스에서 처리
- **DML** 코드에만 적용 ⇒ **SELECT, INSERT, UPDATE, DELETE**

예) 주문 과정: 주문 목록 등록 → 재고 수량 변화 → 결제 등록 → 배송 등록

- 전체 성공 → commit
- 실패 → rollback (전체 작업 취소)

```
try {
  1. 계좌 출금 (본인)
  2. 계좌 입금 (상대)
} catch (Exception e) {

}
commit();
```

실습 1

```
-- 테이블 및 데이터 복사
CREATE TABLE dept_tcl
    AS SELECT * FROM dept;

-- 60번 부서로 임의로 데이터 입력
INSERT INTO dept_tcl VALUES(60, 'DataBase', '제주', 1004);

-- Update => 40번 부서 번호의 loc => '대구'로 수정
Update dept_tcl SET loc = '대구' WHERE dno = 40;

-- 변경 사항 전부 되돌리기
ROLLBACK;

-- 저장하기
-- 이후에는 롤백해도 돌아가지 않음
COMMIT;
```

Sequence

오라클 실행 과정

1. SQL 파싱

- SQL 구문에 오류가 있는지 여부 확인
- SQL 실행 대상 객체(테이블, 뷰) 존재 여부 검사

2. SQL 최적화(실행 계획)
    - SQL이 실행되는데 필요한 비용(cost) 계산
3. SQL 실행
    - 세워진 실행 계획을 통해 물리적 실행

인덱스 (Index)

- 책갈피
- 데이터에 저장된 주소값을 통해 B트리를 형성하고 일정한 검색 속도 유지

인덱스 사용 전

- 검색 성능이 대용량 데이터에 있어서 현저하게 느림 (full scan)
- 검색 속도가 일정하게 보장되지 않음

인덱스 생성

- 해당 컬럼에 대한 indexing으로 row_id 생성
- 인덱스로 된 컬럼 값과 row_id로 구성된 leaf block, 포인터를 갖는 branch block으로 나누어짐
- B-Tree 구조로 Balacnce 유지

오라클에서 인덱스 생성

- PRIMARY KEY, UNIQUE 갖는 컬럼은 기본적으로 인덱스가 자동으로 생성
- CREATE INDEX 인덱스명

인덱스 생성해야 하는 경우

- WHERE절, JOIN 조건으로 자주 사용하는 컬럼
- 모든 값이 컬럼 내에서 UNIQUE
- 넓은 범위의 값을 가진 컬럼
- 아주 드물게 존재하는 컬럼

```
CREATE SEQUENCE board_seq;

-- 들어갈 때마다 자동 증가로 숫자 만들어 준다
INSERT INTO board VALUES(board_seq.nextval, 'a1', 'a', 'a', sysdate, 0);

INSERT INTO board(seq, title, writer, contents, regdate, hitcnt)
    (select board_seq.nextval, title, writer, contents, regdate, hitcnt from board);

-- 인덱스 사용 시 비용 줄어든 걸 알 수 있음 (F10으로 확인)
-- SQL문의 성능 확인
ALTER TABLE board
    ADD CONSTRAINT board_seq_pk PRIMARY KEY(seq);

SELECT * FROM board
        WHERE seq = 2;
```

- 인덱스 생성

```
-- 해당 인덱스에 대해 컬럼을 생성하겠다
CREATE UNIQUE INDEX index_name
    ON table_name(컬럼명);
```

- ‘title’에 대한 글번호(seq) 12번에 대해 title 값을 ‘a12’로 수정하고 ‘a12’번을 검색할 경우 실행 계획 확인 ⇒ full scan
- 이후 인덱스를 생성하고 다시 검색 후 ⇒ index scan
- 비용 차이 확인

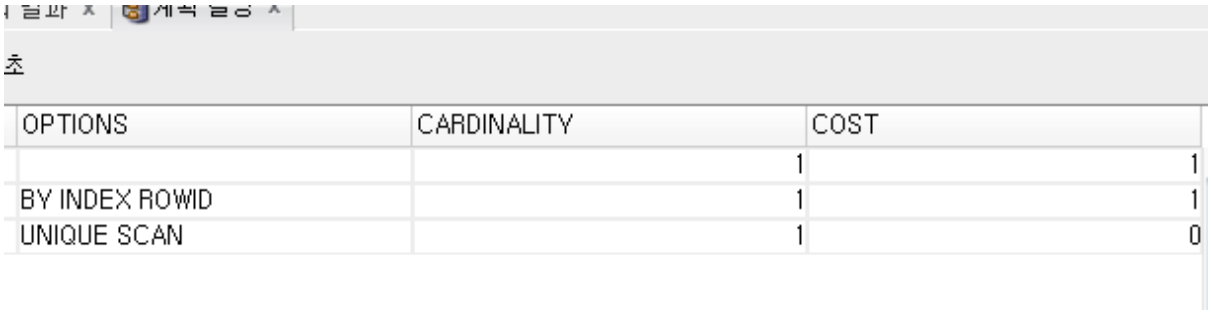
```
UPDATE board set title = 'a12' where seq = 12;

-- 그냥 검색
SELECT * FROM board
    where title = 'a12';
```

OPTIONS	CARDINALITY	COST
	1	2
FULL	1	2

```
-- index 이용하여 검색
SELECT * FROM board
  WHERE seq = 12;
```

```
-- 인덱스 삭제
DROP INDEX seq;
```



초

OPTIONS	CARDINALITY	COST
	1	1
BY INDEX ROWID	1	1
UNIQUE SCAN	1	0

인덱스 결론

1. 빠른 검색 성능(cost 비용 절감)
    - 단, 모든 경우에 그런 것은 아니다
2. 일정한 검색 속도를 유지

View

정의

- 과도한 JOIN에 대한 SQL을 단순화(추상화)하기 위함
- 보안적인 측면에서 강화
- 단, 성능이 좋아지지는 않는다

```
-- 뷰 생성
CREATE[OR REPLACE] VIEW 뷰의 이름 AS
SELECT 칼럼명
FROM 테이블명
WHERE 조건;

-- 뷰 삭제
DROP VIEW 뷰의 이름;
```

실습 2

- 기존에 과도한 조인이 필요한 sql을 view 이용해 단수화

- employees 테이블에서 salary를 제외한 내용으로 view 구현

데이터 모델링

정의

- 정보화 시스템을 구축하기 위해 어떤 데이터가 존재하는지, 업무가 필요로 하는 정보는 무엇인지를 분석하는 방법

순서

1. 엔티티 타입 도출
2. 관계 설정: **기본 키**
3. 속성 도출: 기본 키, 외래 키, 속성

관점

1. 데이터 관점
2. 프로세스 관점
3. 상관 관점

용어

- 어떤 것 → 엔티티 타입 → 엔티티
- 어떤 것과의 관계 → 관계 → 페어링
- 어떤 것의 성격 → 속성 → 속성 값

엔티티 타입

- 업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 것 → 영속적으로 존재하는 단위
- 엔티티가 여러 개 모인 것
- 엔티티와 동의어로 사용하기도 함

엔티티 타입 도출 순서

1. 명사 추출(요구 분석)
2. 불명확하고 광범위한 용어 제거

   예) 추가, 검색, 작품, 영화 관련 미디어, 리뷰 보기, 조회, 출연, 평균, 순서

3. 속성 후보값 제거

   예) 아이디, 비밀번호

4. 포괄적 업무 제거

   예) 콘텐츠, 로그인, 사용자, 리스트

5. 중복된 단어 제거
6. 누락된 엔티티 추가

예비 엔티티 검증

1. 업무에 사용되는지
2. 식별자로 구분 가능한지
3. 영속적으로 존재하는 데이터 갖는지
   - 두 개 이상으로 데이터 값이 계속 들어갈 수 있어야 함
4. 업무 프로세스에 이용되는지
5. 반드시 속성 값을 포함하는지
6. 다른 엔티티 타입과 최소 한 개 이상 관계를 맺는지

엔티티 타입의 분류

- 발생 시점에 따라 분류한다
- 3가지 종류 모두를 가지고 있어야 한다
  - 기본: 해당 업무에 기본적으로 존재하는 정보 예) 사원, 부서
    - 기본 엔티티로는 아무것도 할 수 없다
  - 중심: 업무에 핵심적인 역할을 하는 정보 예) 접수, 계약
  - 행위: 기본과 중심 엔티티 타입을 근간으로 업무가 흘러가며 발생 예) 주문 내역, 계약 진행
- 잘된 ER Digram은 기본 엔티티가 바깥에 있고, 중심 엔티티가 가운데 있다

대리 식별자

교차 엔티티

- 주문 테이블 1 - N 주문 상세 테이블 N - 1 상품 테이블
  - 주문 테이블: 주문 번호(PK)

    - 상품 테이블: 상품 번호(PK)
    - 주문 상세 테이블: 주문 번호(PK, FK)와 상품 번호(PK, FK) 둘 다 가지고 있음

관계(relation)

- 기준 엔티티, 관계 형태, 참여 방법, 관계 엔티티

정의

- 두 개 엔티티 타입 사이의 논리적 관계
- 엔티티끼리 존재나 행위로서 서로의 영향을 주는 형태

기수(cardinality)

- 1 대 다 관계가 제일 안정적이다
- 일대일 관계도 있을 수 있다
- 다 대 다 관계는 존재해서는 안 된다
    - 교차 엔티티를 통해 1 대 다 관계로 만들어 줘야 한다

주식별자와 비식별자

- 주식별자: PK와 FK 역할 동시에 수행 → 실선
    - 상속된 구조면서, 부모 자식 간 강한 결합력
    - 조인을 쓰지 않고도 사용하게 함
- 비식별자: 속성으로서 FK 역할만 수행 → 점선

설정

- 부모 엔티티(PK)와 자식 엔티티(FK)를 선정: 관계의 내용을 정의
- 커디넬리티(1:1, 1:N, N:M) N:M ⇒ 1:M (교차엔티티) 도출
- 관계 필수/선택 설정
- 주식별자/비식별자 관계 설정
    - 주식별자: 부모/자식 상속 관계(교차엔티티, 정규화), 결합력 높음, FK가 주키 영역에 포함
        - 장점: 과도한 조인 감소
        - 단점: 조인 시 코드 증가

- 비식별자: 일반적인 관계, FK가 속성에 포함
  - 장점: 주키의 개수 부담 감소
  - 단점: 과도한 조인 발생

속성

- 해당 엔티티 타입 상태 정보를 표현할 수 있음
- 더 이상 분리되지 않는 최소의 데이터 단위
- 요구 분석의 양과 비례

구성 단위

- PK(식별자 속성), FK(외래키 속성), 일반

실습 3.

```
모든 학생은 고유한 학번을 갖고, 특정 학과에 소속된다.
이름, 주소, 생년월일, 나이도 관리한다.

학과는 학과명, 학과사무실 위치, 전화번호 등을 관리하고, 학교 내에서 같은 이름의 학과는 없다.

학생은 수강할 과목을 등록하는데, 과목에는 과목번호, 과목명, 과목개요 등이 있다.

과목은 여러 색션으로 나누어질 수 있는데, 섹션에는 고유한 섹션번호가 있다.
모든 과목이 섹션으로 나누어지는 것이 아니므로 섹션은 과목이 없으면 존재할 필요가 없다.
또한 다른 과목의 섹션은 같은 섹션번호를 가질 수 있다.

교수는 교수번호로 식별할 수 있고, 교수이름, 전공분야, 보유기술 등을 관리한다.
교수는 여러개의 보유기술을 가질 수 있다.

교수는 과목을 강의하고 학생에 대해 전공지도를 한다. 일부는 학과의 학과장이 된다.
당연히 학과마다 학과장은 한 명씩 있다.
```

* 도서 대여점 관리 프로그램

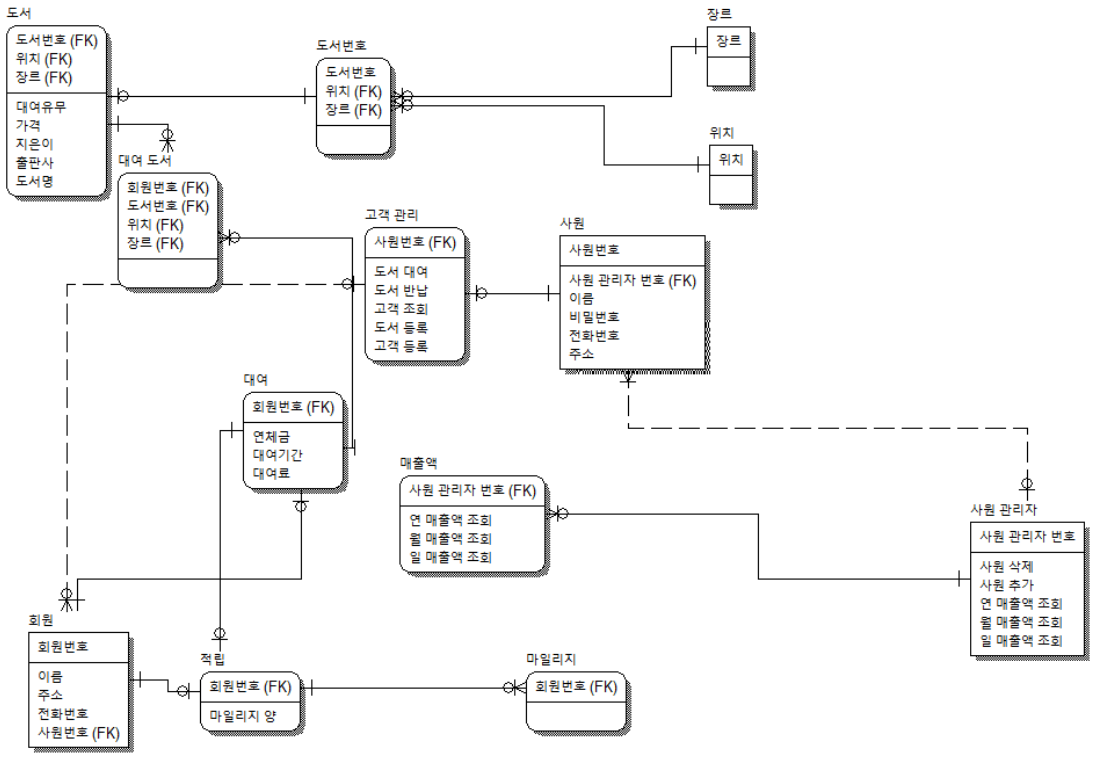
요구사항 분석

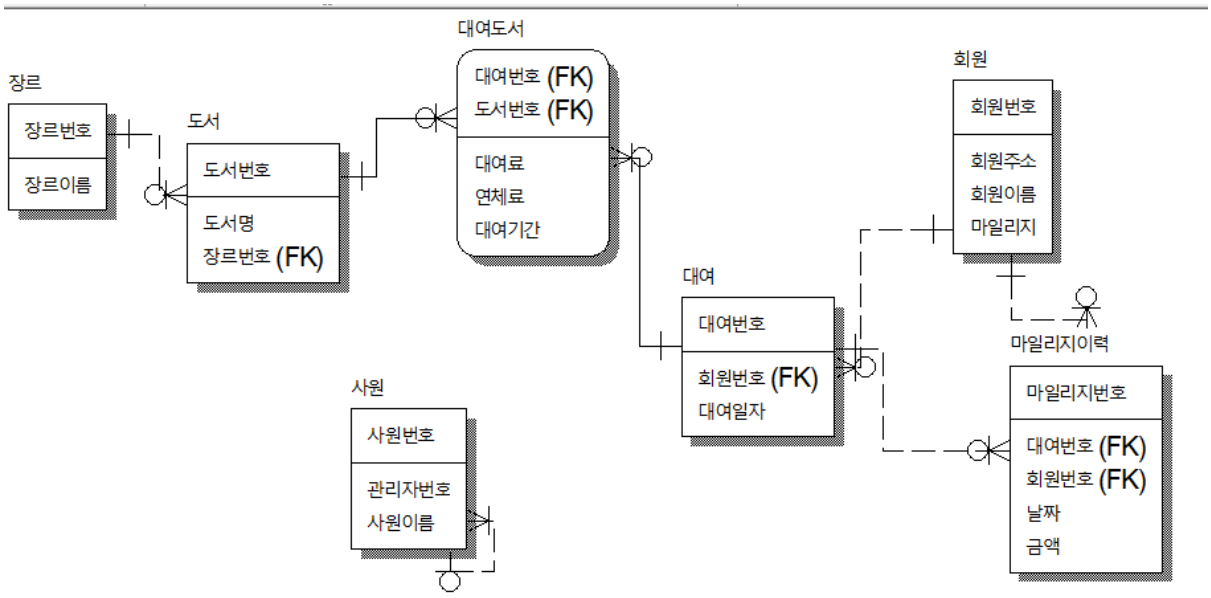
1. 도서에 대해서 도서번호, 도서명, 출판사, 지은이, 장르, 가격, 대여유무,
   등록일을 저장한다. 도서번호에는 장르와 위치를 넣어서 도서를 찾기 쉽게한다.

2. 도서는 코믹스, 순정, 애정판, 소설, 월간지 장르로 구분한다. 장르마다 대여료와
   대여기간, 연체금이 정해져 있다. 도서는 한 장르에만 속해 있다. 대여 중인
   책은 대여 될 수 없다.

3. 회원에 대해서 회원번호, 이름, 전화번호, 주소, 대여 중인 책, 마일리지를 저장한다.
   회원은 도서를 대여 할 수 있다. 회원은 여러 권의 책을 동시에 빌릴 수 있다.
   회원은 도서를 대여할 때마다 대여금액의 10%를 마일리지로 적립받을 수 있고,
   도서를 대여할 때 현금처럼 사용할 수 있다.

4. 사원에 대해서 사원번호, 이름, 주소, 전화번호, 비밀번호를 저장한다.
   사원은 고객등록, 도서등록, 도서대여, 도서반납, 고객조회를 할 수있다.
   사원중에 사원을 관리하는 관리자가 있어서 사원을 추가/삭제를 할 수 있고,
   일별, 월별, 연별 매출액을 조회할 수 있다.




장르
장르번호
장르이름
도서
도서번호
도서명
장르번호 (FK)
대여도서
대여번호 (FK)
도서번호 (FK)
대여료
연체료
대여기간
회원
회원번호
회원주소
회원이름
마일리지
대여
대여번호
회원번호 (FK)
대여일자
마일리지이력
마일리지번호
대여번호 (FK)
회원번호 (FK)
날짜
금액
사원
사원번호
관리자번호
사원이름