**물리 데이터 모델링**

**1. 물리 데이터 모델 정의**

- 하나의 논리적 집합은 하나 이상의 테이블이 될 수 있다.

- 테이블은 경우에 따라 속성의 일부만으로 생성될 수 있다.

- DBMS의 특성을 고려하여 논리 모델을 데이터베이스 저장구조로 변환하는 것이다.

- 물리 데이터 모델링 : 데이터의 구조에 관련된 것들을 물리적인 모습까지 설계

- 데이터베이스 디자인 : 물리모델을 DBMS 관점의 오브젝트로 생성하는 최적의 설계를 하는 것

- 데이터베이스 디자인의 예 : 오브젝트 파티셔닝 설계, 최적의 인덱스 설계, 오브젝트별 저장공간의 효율적 사용계획 등

**2. 물리 데이터 모델 의의**

- 관계 데이터 모델링이라고도 한다.

- 논리 데이터 모델을 RDBMS의 특성, 기능, 성능을 고려하여 데이터베이스의 물리적 구조를 작성해나가는 과정

- 물리 데이터 모델링 단계 : 논리 데이터 모델에서 도출된 내용 변환을 포함하여 데이터의 저장공간, 데이터의 분산, 데이터 저장 방법 등을 함께 고려하는 단계

- 물리 데이터 모델 단계에서 데이터베이스 운용 성능으로 나타나므로 소홀히 다루면 안된다.

**3. 논리 데이터 모델 - 물리 데이터 모델**

하나의 논리적 데이터 모델이 서로 다른 형태의 물리적 데이터 모델을 설계하는 경우

- 분산 데이터베이스 구축시

- 물리 데이터 모델 비교

- 물리적 환경의 변환

- 물리적 모델의 형상 관리

**물리 요소 조사 및 분석**

- 시스템 구축 관련 명명 규칙

- 하드웨어 자원 파악 : CPU / Memory / Disk / I/O Controller / Network

- 운영체제 및 DBMS 버전 파악

- 데이터베이스 운영과 관련된 관리 요소 파악

(1) 사용자 관리 기법 및 정책

(2) 백업 / 복구 기법 및 정책

(3) 보안 관리 정책

**논리 - 물리 모델 변환**

**1. 논리 데이터 모델 - 물리 데이터 모델 변환 용어**

분석 설계

논리 데이터 모델링 -> 물리 데이터 모델링

ER Model Physical Model

Entity Table

Attribute Column

Primary UID Primary Key

Secondary UID Unique Key

Relationship Foreign Key

Business Constraints Check Constraints

**2. 엔터티 - 테이블 변환**

1) 테이블 설명

- 테이블, 로우, 컬럼, 기본키, 외래키

2) 서브 타입 변환

|  |  |  |  |
| --- | --- | --- | --- |
| 종류 | 구성요건 | 장점 | 단점 |
| 통합 | 주로 서브타입에 적은 량의 속성이나 관계를 가진 경우 | - 데이터 액세스가 좀 더 간편  - 뷰를 활용하여 각 서브타입만을 액세스 하거나 수정 가능  - 수행속도가 좋은 경우가 많음  - 서브타입 구분 없는 임의 집합의 가공이 용이함  - 다수의 서브타입을 통합한 경우 조인 감소 효과 큼  - 복잡한 처리를 하나의 sql로 통합하기 용이 | - 특정 서브타입의 Not Null 제한 불가  - 테이블의 칼럼 수 증가  - 테이블의 블록 수 증가  - 처리시 마다 서브타입의 구분 필요가 많음  - 인덱스 크기 증가 |
| 각각의 테이블로 | 주로 서브타입에 많은 양의 속성이나 관계를 가진 경우 적용 | - 각 서브타입의 속성들의 선택 사양이 명확한 경우 유리  - 처리시마다 서브타입 유형 구분 불필요  - 전체 테이블 스캔 시 유리  - 단위 테이블의 크기 감소 | - 서브타입 구분 없이 데이터 처리하는 경우 Union 발생  - 처리 속도가 길어지는 경우가 많음  - 트랜잭션 처리 시 여러 테이블을 처리하는 경우 증가  - 복잡한 처리의 SQL 통합이 어려워 짐  - 부분 범위 처리가 불가능해질 수 있음  - 여러 테이블을 합친 뷰는 조회만 가능  - UID 유지 관리가 어려움 |
| 아크관계 | - 전체 데이터 처리가 빈번하게 발생하는 경우  - 서브타입의 처리는 주로 독립적으로 발생할 경우  - 테이블을 통합했을 때 컬럼 수가 너무 많아지는 경우  - 서브타입의 컬럼 수가 많아지는 경우  - 트랜잭션이 주로 슈퍼타입에서 발생하는 경우  - 슈퍼타입의 처리 범위가 넓고 빈번하여 단일 테이블 클러스터링을 해야할 경우 |  |  |

**3. 속성 - 컬럼 변환**

일반 속성 변환

- 컬럼의 명칭은 속성의 명칭과 일치할 필요는 없으나, 프로그래머와 사용자의 혼돈을 피하기 위하여 가능한 표준화된 약어를 사용

- SQL 예약어는 피한다

- 컬럼 명칭은 짧은 것이 좋다

- 가능하다면 표본 데이터를 입력시킨다.

Primary UID -> 기본키 변환

- 키 형태란에 엔터티의 Primary UID에 속하는 속성들에 PK를 표시

- PK로 표시된 모든 컬럼들은 Nulls / Unique 란에 반드시 NN, U 표시

- 여러 개의 컬럼으로 UID가 구성되어 있는 경우는 각각의 컬럼에 NN, U1 표시

- 또 다른 Unique Key가 있다면 U2 표시

Primary UID(관계의 UID Bar) -> 기본키 변환

- 테이블에 외래키 컬럼을 포함

- PK의 일부분으로 표시

Secondary UID -> Unique 키 변환

- 변환 절차는 기본적으로 Primary UID 변환 절차와 동일

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 관계 | | 특성 | 변환절차 | 주의점 |
| 1:M | | M쪽 관계의 형태에 따라 관계 컬럼의 선택사양 결정 | - 1에 있는 PK를 M의 FK로 변환  1) FK의 명칭 결정  2) 키 형태란에 FK 표시  3) 필수 관계일 경우 NN표시  - 표본 데이터 추가  - UID Bar가 있는 경우는 전 단계에서 실시 | 부모쪽이 Mandatory 관계일 때의 주의 사항  - 자식쪽의 레코드가 반드시 하나 이상은 되어야만 부모 쪽의 레코드를 생성가능  - 자식쪽 레코드를 삭제할 경우에는 전체를 다 삭제할 수는 없고 반드시 하나 이상의 자식 레코드를 남겨둬야 한다. 또는 자식, 부모 레코드를 동시에 삭제해야 한다. |
| 1:1 | | - 논리 모델에서는 자주 발생하지 않음  - 물리 모델로 변환하는 과정은 관계의 선택사양에 따라 다른 방법으로 적용 | - 선택 테이블의 PK를 필수 테이블의 FK로 변환  - NN 표시 | - 모든 FK 부분은 Unique Key가 필수  - 필수-선택 관계 : 필수가 FK를 가짐  - 선택-선택 관계 : 빈번하게 사용되는 테이블이 FK를 가짐  - 필수-필수 관계 : 어느 쪽에 FK를 가질지 선택 필요 |
| 1:M 순환 관계 | | - 최상위 관계 속성은 항상 선택인 형태  - 하지만 경우에 따라서는 최상위의 관계 속성에 특정 값을 지정하는 경우도 존재 | - 해당 테이블 내에 FK 컬럼을 추가  - FK는 동일 테이블 내의 다른 로우의 PK 컬럼 참조  - FK 컬럼 명칭은 가능한 관계 명칭 반영  - FK는 결코 Not Null이 될 수 없음 |  |
| 아크 | FK 분리방법 | - 각각의 관계를 관계 컬럼으로 생성  - FK 제약 조건을 생성할 수 있는 장점  - 각각의 FK 컬럼들이 선택이어야 한다.  - 추가적인 체크 제약조건 |  |  |
| FK 결합방법 | - 각각의 관계를 하나의 관계 컬럼으로 생성  - 구분 컬럼 추가  - 단점  1) FK 제약조건을 생성할 수 없다.  2) 각각의 관계를 선택적으로 구분할 수 있는 구분컬럼 추가 필요 |  |  |

**4. 데이터 타입 선택**

- 개념

- 문자타입

- 숫자타입

- 날짜타입

**5. 데이터 표준 적용**

- 개념 : 표준 용어, 표준 도메인, 표준 명명규칙

- 데이터 표준 적용 대상 : 데이터베이스, 스토리지 그룹, 테이블스페이스, 테이블, 컬럼, 인덱스, 뷰

- 데이터 표준 적용 방법

**반정규화**

**테이블 분할**

**1) 수평 분할**

- 개념 : 레코드를 기준으로 분할, 파티션

- 사용 의의

(1) 하나의 테이블에 데이터가 많고, 레코드 중에서 특정 범위만을 주로 액세스 하는 경우 사용

(2) 분할된 각 테이블은 서로 다른 디스크에 위치시켜 물리적인 디스크의 효용성 극대화 가능

(3) DBMS 차원에서 제공

(4) 대표적인 방법은 범위 분할, 해쉬 분할, 복합 분할 기법

**2) 수직 분할**

- 개념

(1) 하나의 테이블이 가지는 컬럼의 수가 많아 분할

(2) 조회 위주의 컬럼과 갱신 위주의 컬럼으로 나뉘는 경우 -> Locking 문제

(3) 특별히 자주 조회되는 컬럼이 있는 경우

(4) 특정 컬럼의 크기가 아주 큰 경우 (LOB Type)

(5) 특정 컬럼에 보안을 적용해야하는 경우

**중복 테이블 생성**

1) 개념 : 집계 테이블 경우 중복

2) 중복 테이블 생성 판단 근거

- 수행 속도 증가 시

- 많은 범위를 자주 처리해야 하는 경우

- 특정 범위만 자주 처리되는 경우

- 처리 범위를 줄이지 않고는 수행속도를 개선할 수 없는 경우

- 요약 자료만 주로 요구되는 경우

- 추가된 테이블의 처리를 위한 오버헤드를 고려하여 결정

- 인덱스의 조정이나 부분범위 처리 또는, 클러스터링 활용 등을 검토한 후 결정

- 위의 상황에서는 논리 데이터 모델에는 존재하지 않지만, 물리 데이터 모델에서 중복 테이블을 추가하여 생성 가능

**3) 중복 테이블 유형**

**- 집계(통계) 테이블**

(1) 집계 테이블 유형 : 단일 테이블의 Group by, 여러 테이블의 조인 Group by

(2) 중복 테이블 생성 시 유의 사항

- 로우 수와 활용도를 분석하고, 시뮬레이션을 통해서 그 효용성에 대한 면밀한 검토가 선행돼야 한다.

- 집계 테이블에 단일 테이블 클러스터링을 한다면 집계 레벨을 좀 더 낮춰 활용도를 높일 수 있는 지 검토 필요

- 클러스터링, 결합 인덱스, 고급 SQL을 집계테이블 대신 활용 여부 사전 검토

- 지나친 집계 테이블을 만들지 않는 것이 좋다

- 추가된 집계 테이블을 기존 응용 프로그램이 이용할 수 있는지를 찾아 보정시키는 노력이 필요하다.

- 데이터베이스 트리거의 오버헤드에 주의, 데이터의 일관성 보장에 유의

-> 집계 테이블과 원본 데이터는 일관성 유지가 매우 중요

**(3) 진행 테이블 추가**

진행 테이블 추가 상황

- 여러 테이블의 조인이 빈번히 발생하며 처리 범위도 넓은 경우

- M:M 관계가 포함된 처리의 과정을 추적, 관리하는 경우

- 검색 조건이 여러 테이블에 걸쳐 다양하게 사용되며 복잡하고 처리량이 많은 경우

진행 테이블 생성 시 유의 사항

- 데이터량이 적절하고 활용도가 좋아지도록 기본키를 선정

- 필요에 따라 적절한 추출 컬럼을 추가하여 집계 테이블의 역할도 하는 다목적 테이블 구상

- 다중 테이블 클러스터링이나 조인 SQL을 적절히 이용하면 굳이 진행테이블을 만들지 않아도 양호한 수행속도를 낼 수 있는 경우가 많다.

**중복 컬럼 생성**

1) 개념

- 논리모델 : 정규화 과정을 통해 중복 제거 -> 일관성 확보

- 물리모델 : 성능을 위한 반정규화, 중복 허용

2) 중복 컬럼 생성 상황

- 빈번하게 조인을 일으키는 컬럼에 대해 고려

- 조인의 범위가 다량인 경우를 온라인화 해야하는 경우

- 액세스의 조건으로 자주 사용되는 컬럼

- 빈번히 사용되는 액세스 조건이 다른 테이블에 분산되어 있어 상세한 조건에도 불구하고 액세스 범위를 줄이지 못하는 경우, 자주 사용되는 조건들을 하나의 테이블로 모아서 조건의 변별성을 극대화할 수 있다.

- 복사된 컬럼의 도메인은 원본 컬럼과 동일해야 한다 -> 데이터 일관성(필수 사항)

- 접근 경로의 단축을 위해서 부모 테이블의 컬럼을 자식 테이블에 중복

- 상위 레벨의 테이블에 집계된 컬럼을 추가할 수 있다 -> 집계 컬럼 추가

- 하위 레벨의 테이블로 중복 컬럼을 복사할 수 있다.

- 연산된 결과를 주로 사용하는 경우에도 미리 연산을 하여 중복 컬럼 생성

- 여러 개의 로우로 구성된 값을 하나의 로우에 나열하는 경우

- 기본키의 컬럼이 길거나 여러 개의 컬럼으로 구성되어 있는 경우 인위적인 기본키를 추가할 수 있다.

3) 중복 컬럼 생성 시 유의 사항

- 다중 테이블 클러스터링으로 해결할 수 있는지 검토

- SQL Group 함수 이용하여 처리할 수 있는지 검토

- 저장 공간의 지나친 낭비를 고려하여 적절한 대비책 마련

- 반복 컬럼은 특별한 경우를 제외하고는 절대 사용할 필요가 없고, 있다면 sum(decode..) 용법과 같은 SQL 기법을 활용하여 이러한 부분을 피할 수 있도록 한다.

- 데이터의 일관성 보장에 유의해야 한다.

- 컬럼의 중복이 지나치게 심하면 데이터 처리의 오버헤드가 발생하게 된다.

- 사용자나 프로그램은 반드시 원본 컬럼만 수정하는 것이 바람직하다.

- 많은 중복 컬럼을 생성하고 있는 것이 현실이나 가능하면 중복 컬럼을 적게 가져가는 것이 바람직하다.

- 클러스터링, 결합 인덱스, 적절한 SQL을 이용하면 특별한 경우를 제외하고는 거의 해결 가능하기 때문에 이부분을 먼저 적극적으로 고려해보는 것이 바람직하다.

- 중복 컬럼을 이용하면 손쉽게 액세스 효율을 개선할 수 있으나, 지나친 중복화는 반드시 데이터 일관성 오류 발생의 개연성 증가 및 데이터 처리 오버헤드 증가라는 반대 급부가 있다는 것을 고려해야 한다.

- 조인, 서브쿼리 액세르 경로의 최적화 방안을 보다 철저히 간구해야한다.