31. 데이터 모델의 개념

1. 데이터 모델의 정의

현실 정보들을 컴퓨터에 표현하기 위해 단순화, 추상화하여 체계적으로 표현한 개념적 모형

2. 데이터 모델의 구성요소

개체(Entity) – 데이터베이스에 표현, 사람이 생각하는 개념이나 정보 단위 같은 현실 세계 대상체

속성(Attribute) – 데이터의 가장 작은 논리적 단위, 파일 구조상의 데이터 항목 또는 데이터 필드

관계(Relationship) – 개체 관의 관계 또는 속성 간의 논리적인 연결을 의미

3. 개념적 데이터 모델

현실 세계에 대한 인식을 추상적 개념으로 표현하는 과정, 정보모델이라고도 함 ex) E-R모델

4. 논리적 데이터 모델

개념적 모델링 과정에서 얻은 개념적 구조를 컴퓨터가 이해하고 처리할 수 있게 변환하는 과정

데이터 간의 관계 표현 차이에 따라 관계 모델, 계층 모델, 네트워크 모델로 구분(종류)

5. 논리적 데이터 모델의 품질 검증

완성된 데이터 모델이 기업에 적합한지를 확인하기 위해 품질을 검증하는 것

개체 품질 검증 항목 : 단수 명사 여부 상호 배타성, 개체 동의어/명사/정규화 여부, 주 식별자 등

속성 품질 검증 항목 : ‘’, 도메인 정의, 속성 값 존재 개수, 그룹화 가능/반복/다치 종속 속성 등

식별자 품질 검증 항목 : 노드의 기수성과 선택성, 필수적/유효한/중복된 관계, 외부 식별자 존재

기수성 : 가능한 관계의 수를 의미 ex)1:1 1:M 등, 선택성 : 관계가 필수인지 선택인지 표현 ex)1:0

전반적인 품질 검증 항목 : 주제 영역 구성의 적절성, 데이터 모델 상에 정규화 여부, 다대다 관계 해소 여부, 이력 관리 대상 선정 확인, 이력 관리 방법의 적절성 확인

6. 데이터 모델에 표시할 요소

구조(Structure) : 논리적으로 표현된 개체 타입들 간의 관계로서 데이터 구조 및 정적 성질 표현

연산(Operation) : DB에 저장된 실제 데이터를 처리하는 작업에 대한 명세, 기본 조작 도구

제약 조건(Constraint) : DB에 저장될 수 있는 실제 데이터의 논리적인 제약 조건

32, 이상/함수적 종속/정규화

1. 이상(Anomaly)

데이터의 중복(Redundancy)으로 인해 테이블 조작 시 문제가 발생하는 현상을 의미한다.

삽입 이상(Insertion Anomaly) : 데이터 삽입 시 의도와는 달리 원하지 않은 값들로 인해 삽입x

삭제 이상(Deletion Anomaly) : 상관없는 값들도 함께 삭제(연쇄 삭제)되는 현상

갱신 이상(Update Anomaly) : 속성 값 갱신 시 일부 튜플의 정보만 갱신되어 불일치성 발생

2. 함수적 종속(Functional Dependency)

X와 Y는 R의 속성 집합의 부분 집합, X의 값 각각에 대해 시간에 관계없이 항상 Y의 값이 오직 하나만 연관되어 있을 때 Y는 X에 함수적 종속 또는 X가 Y를 함수적으로 결정한다고 함, X->Y

X = 결정자(Determinant), Y = 종속자(Dependent), 완전 함수적 종속(Full Functional Dependency)

3. 정규화(Normalization)의 개념

테이블의 속성들이 상호 종속적인 관계를 갖는 특성을 이용하여 테이블을 무손실 분해하는 과정

4. 정규화 과정

제 1정규형

도메인이 원자 값만으로 되어 있는 정규형, 즉 테이블의 모든 속성 값이 원자 값으로만 되어있음

제 2정규형

1NF + 기본키가 아닌 모든 속성이 기본키에 대하여 완전 함수적 종속을 만족하는 정규형

제3정규형

모든 속성이 이행적 함수적 종속(Transitive Functional Dependency)를 만족하지 않는 정규형

BCNF 모든 결정자가 후보키(Candidate Key)인 정규형

제 4정규형

다중 값 종속(MVD; Multi Valueed Dependency) A-->B일 때 모든 속성이 A에 함수적 종속

제 5정규형 – 모든 조인 종속(JD;Join Dependancy)이 후보키를 통해서만 성립되는 정규형

두(메인 원자값) 부(차적) 이(행적) 결(정자면서 후보키X 제거) 다(치 종속) 조(인 종속성 이용)

33. 논리 데이터 모델의 물리 데이터 모델로 변환

1. 테이블(Table)

데이터를 저장하는 데이터베이스의 가장 기본적인 오브젝트, 컬럼(Column), 행(Row)

행 : 튜플, 인스턴스, 어커런스 / 열 : 각 속성 항목에 대한 값 저장

기본키(Primary Key) : 후보키에서 선택한 주키, 특정 튜플을 유일하게 구분할 수 있는 속성

외래키(Foreign Key) : 다를 R의 기본키를 참조하는 속성 또는 속성들의 집합을 의미

2. 엔티티(Entity)를 테이블로 변환

논리 데이터 모델에서 정의된 엔티티를 물리 데이터 모델의 테이블로 변환

테이블 목록 정의서 : 전체 테이블을 목록으로 요약 관리하는 문서, 테이블 목록이라고도 함

3. 슈퍼타입/서브타입을 테이블로 변환

슈퍼타입 기준 테이블 변환 : 서브타입을 슈퍼타입에 통합하여 하나의 테이블로 만드는 것

서브타입에 속성이나 관계가 적을 경우에 적용하는 방법, 서브타입의 모든 속성이 포함되어야 함

장점 : 액세스 상대적 용이, 뷰를 이용해 각각의 서브타입만을 액세스하거나 수정 가능, SQL 문장 구성이 단순, 조인이 적어 수행 속도 향상, 서브타입 구분이 없는 임의 집합에 대한 처리가 용이

단점 : 컬럼 증가로 디스크 저장 공간 증가, 인덱스 효율 떨어짐, 서브타입 구분 필요 많이 발생

서브타입 기준 테이블 변환 : 슈퍼타입 속성들을 각각의 서브타입에 추가하여 테이블 생성

장점 : 서브타입 유형 구분 필요x, 테이블당 크기 감소, 전체 테이블 스캔 시 유리

단점 : 수행 속도 감소, sql 통합의 어려움, 통합 뷰는 조회만 가능, 식별자 유지관리 어려움

개별타입 기준 테이블 변환 : 슈퍼, 서브타입들을 각각의 개별적인 테이블로 변환하는 것

장점 : 저장 공간이 상대적으로 적음, 문장 작성 용이

단점 : 슈퍼타입 또는 서브타입의 정보를 같이 처리 시 항상 조인이 발생해 성능 저하

4. 속성을 컬럼으로 변환

일반 속성 변환 : 개발자와 사용자간 소통을 위해 표준화된 약어 사용 및 일치

Primary UID를 기본키로 변환

Primary UID(관계의 UID Bar)를 기본키로 변환 : 다른 엔티티와의 관계로 인해 생성된 키도 동일

Secondary(Alternate) UID를 유니크키로 변환 : 물리 모델에서 유니크키로 변환

5. 관계를 외래키로 변환 : 정의된 관계는 기본키와 이를 참조하는 외래키로 변환

1:1 관계 : 개체 A의 기본키를 개체 B의 외래키로 추가, 반대로도 추가하여 표현 가능

1:M 관계 : 개체 A의 기본키를 개체 B의 외래키로 추가하여 표현하거나 별도의 테이블로 표현

N:M 관계 : A 와 B의 기본키를 모두 포함한 별도의 릴레이션으로 표현, 이때 생성된 별도의 릴레이션을 교차 릴레이션(Intersection Relation) 또는 교차 엔티티(Intersection Entity)라고 함

1:M 순환 관계 : 개체 A에 개체 A의 기본키를 참조하는 외래키 컬럼을 추가하여 표현

6. 관리 목적의 테이블/컬럼 추가

논리 데이터 모델에 존재하지 않는 테이블이나 컬럼을 DB의 관리, 수행속도를 높이기 위해 추가

7. 데이터 타입 선택

CHAR : 고정길이 문자열 Data 최대 2000Byte까지 저장가능 VARCHAR(4000Btye)

NUMBER : 38자릿수의 숫자 저장 가능 date : 날짜저장

34. 반정규화(Denormalization)

1. 반정규화의 개념

시스템의 성능 향상, 개발 및 운영의 편의성 등을 위해 정규화된 데이터 모델을 통합, 중복, 분리

시스템의 성능 향상 및 관리 효율성 증가 , 데이터의 일관성 및 정합성 저하

2. 테이블 통합

두 개의 테이블이 자주 조인(Join)되 하나의 테이블로 만드는 것이 성능 향상에 도움이 될 경우

고려사항 : 검색은 간편하지만 레코드 증가로 인해 처리량 증가, 입력/수정/삭제 규칙 복잡

Not null, default, check 등의 제약조건을 설계하기 어려움

3. 테이블 분할 : 테이블을 수직 또는 수평으로 분할하는 것

수평 분할(Horizontal Partitioning) : 레코드를 기준으로 테이블을 분할, 빈도에 따라 분할

수직 분할(Vertical Partitioning) : 하나의 테이블에 속성이 너무 많을 경우 속성을 기준으로 분할

갱신 위주의 속성 분할 : 갱신 시 레코드 잠금으로 인해 다른 작업을 수행할 수 없어 갱신이 자주 일어나는 속성들을 수직 분할하여 사용

레코드 잠금 : 데이터 무결성 유지를 위해 변경 작업이 완료될 때까지 타 프로세스가 변경 불가

자주 조회되는 속성 분할 : 자주 조회되는 속성이 극히 일부일 경우 해당 속성들을 수직 분할

크기가 큰 속성 분할 : 이미지나 2GB 이상 저장될 수 있는 텍스트 형식으로 된 속성 분할

보안을 적용해야 하는 속성 분할 : 보안을 적용할 속성들을 수직 분할

고려 사항 : 기본키 관리 어려워짐, 수행 속도 느려질 수 있음, 데이터 검색에 중점

4. 중복 테이블 추가 – 데이터를 추출해서 사용해야 하거나 다른 서버에 저장된 테이블을 이용해야 하는 경우 중복 테이블을 추가하여 작업을 효율성을 향상시킬 수 있다.

정규화로 인해 느려진 수행 속도, 많은 범위의 데이터 처리 빈번, 특정 범위의 데이터 처리 빈번, 처리 범위를 줄이지 않고는 수행 속도를 개선할 수 없는 경우에 중복 테이블을 추가함

집계 테이블의 추가 : 집계 데이터를 위한 테이블 생성 후 원본 테이블에 트리거 설정, 오버헤드 유의

진행 테이블의 추가 : 이력 관리 등의 목적으로 추가하는 테이블, 활용도를 높이기 위해 기본키 적절하게 설정

특정 부분만을 포함하는 테이블의 추가 : 해당 부분만으로 새로운 테이블을 생성

5. 중복 속성 추가 – 조인 시 데이터 조회 경로를 단축하기 위해 자주 사용하는 속성을 하나 더 추가하는 것

추가 시 데이터의 무결성 확보가 어렵고 디스크 공간이 추가로 필요하다

빈번한 조인 발생, 접근 경로가 복잡한 속성, 액세스의 조건으로 자주 사용되는 조건의 경우에 사용

고려사항 : 테이블 중복과 속성의 중복 고려, 일관성 및 무결성에 유의, 저장 공간의 낭비 고려

SQL 그룹 함수를 이용하여 처리할 수 있어야 함

35. 인덱스 설계

1. 인덱스(Index)의 개념 – 데이터 레코드를 빠르게 접근하기 위해 <키, 포인터> 쌍으로 구성되는 데이터 구조

데이터가 저장된 물리적 구조와 밀접한 관계, 레코드가 저장된 물리적 구조에 접근하는 방법을 제공

파일의 레코드에 대한 액세스를 빠르게 수행, 레코드의 삽입/삭제 수시 발생 시 인덱스의 개수를 최소로 함

인덱스가 없을 시 특정 값을 찾기 위해 모든 데이터 페이지를 확인하는 Table Scan이 발생한다

Table Scan : 테이블에 있는 모든 레코드를 순차적으로 읽는 것

클러스터드(Clustered) 인덱스 : 레코드의 물리적 순서가 인덱스의 엔트리 순서와 일치하게 유지되도록 구성

2. 트리 기반 인덱스 – 인덱스를 저장하는 블록들이 트리 구조를 이루고 있는 것

B 트리 인덱스 : 일반적으로 사용되는 인덱스 방식, 루트 노드에서 하위 노드로 키 값의 크기 비교

B+ 트리 인덱스 : 비 단말 노드로 구성된 인덱스 세트(Index set)와 단말 노드로만 구성된 순차 세트로 구분

인덱스 세트에 있는 노드들은 단말 노드에 있는 키 값을 찾아갈 수 있는 결로로만 제공, 순차 세트에 있는 단말 노드가 해당 데이터 레코드의 주소를 가리킴, 단말 노드만을 이용한 순차 처리 가능

3. 비트맵 인덱스 – 인덱스 컬럼의 데이터를 Bit 값이 0 또는 1로 변환하여 인덱스 키로 사용하는 방법

키 값을 포함하는 로우의 주소를 제공하는 것이 목적, 분포도가 좋은 컬럼에 적합, 성능 향상 효과 기대 가능

데이터가 Bit으로 구성되어 있기 때문에 효율적인 논리 연산 가능, 저장공간 작음, 다중 조건을 만족하는 튜플의 개수 계산에 적합, 동일한 값이 반복되는 경우가 많아 압축 효율이 좋음

4. 함수 기반 인덱스 – 컬럼의 값 대신 컬럼에 특정 함수나 수식(Expression)을 적용하여 산출된 값을 사용

B+트리 또는 비트맵 인덱스를 생성하여 사용, 데이터 입력 또는 수정 시 함수 적용으로 인해 부하 발생 가능, 사용자 정의 함수를 사용할 경우 시스템 함수보다 부하가 더 큼, 대소문자 띄어쓰기 상관없이 조회할 때 유용

적용 가능한 함수의 종류 : 산술식(Arithmetic Expression), 사용자 정의함수, PL/SQL Function, package, C callout

5. 비트맵 조인 인덱스 – 다수의 조인된 객체로 구성된 인덱스, 비트맵 인덱스와 물리적 구조 동일

6. 도메인 인덱스 – 개발자가 필요한 인덱스를 직접 만들어 사용하는 것, 확장형 인덱스(Extensible Index)

7. 인덱스 설계 – 기본 인덱스를 먼저 지정 후 개발 단계에서 필요한 인덱스의 설계를 반복적으로 진행

인덱스 설계 순서

대상 테이블이나 컬럼 등을 선정 -> 효율성을 검토하여 인덱스 최적화를 수행 -> 인덱스 정의서를 작성

8. 인덱스 대산 테이블 선정 기준 : multi block read 수에 따라 판단, 랜덤 액세스 빈번한 테이블, 특정 범위나 특정 순서로 데이터 조회가 필요한 테이블, 다른 테이블과 순차적 조인이 발생되는 테이블

Multi block read : 테이블 액세스 시 메모리에 한번에 읽어 들일 수 있는 블록의 수

9. 인덱스 대상 컬럼 선정 기준 : 인덱스 컬럼의 분포도가 10~15% 이내인 컬럼

10. 인덱스 설계 시 고려사항 : 인덱스 추가 시 기존 액세스 경로에 영향, 많은 인덱스/넓은 범위 처리는 오버헤드 유발, 추가적인 저장 공간 필요, 인덱스와 테이블 데이터의 저장 공간이 분리되도록 설계

36. 뷰 설계

1. 뷰(View)의 개요

사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 테이블로부터 유도된 이름을 가지는 가상 테이블

저장장치 내에 물리적으로 존재x, 임시 작업을 위한 용도, 조인문의 사용 최소화로 사용상의 편의성을 최대화

2. 뷰의 특징

논리적 독립성 제공 가능, 관리가 용이하고 명령문이 간단해짐, 뷰에 나타나지 않는 데이터를 안전하게 보호, 기본 테이블의 기본키를 포함한 속성 집합으로 뷰를 구성해야만 삽입, 삭제, 갱신 연산이 가능

정의된 뷰는 다른 뷰의 정의에 기초가 될 수 있음, 연결된 테이블이나 뷰 삭제 시 연결된 뷰 자동 삭제

3. 뷰의 장 단점

장점 : 논리적 데이터 독립성 제공, 동일 데이터에 대해 동시에 여러 사용자의 상이한 응용이나 요구 지원, 사용자의 데이터 관리를 간단하게 해준다, 접근 제어를 통한 자동 보안 제공

단점 : 독립적인 인덱스X, 뷰의 정의를 변경 불가는, 삽입/갱신/삭제 연산에 제약이 따름

4. 뷰 설계 순서 : 대상 테이블 선정(빈번한 조인 등) -> 대상 컬럼 선정 -> 정의서 작성

5. 뷰 설계 시 고려 사항

테이블 구조가 단순화 될 수 있도록 반복적으로 조인을 설정하여 사용, 동일한 조건절을 사용하는 테이블을 뷰로 생성, 사용할 데이터를 다양한 관점에서 제시, 데이터의 보안 유지를 고려하여 설계

37, 클러스터 설계

1. 클러스터(Cluster)의 개요

데이터 저장 시 데이터 액세스 효율을 향상시키기 위해 동일한 성격의 데이터를 동일한 데이터 블록에 저장하는 물리적 저장 방법, Clustering키로 지정된 컬럼 값의 순서대로 저장, 여러 개의 테이블이 하나의 클러스터에 저장

클러스터링 키 : 클러스터링 된 테이블에서 각각의 행을 접근할 때 기준이 되는 열

2. 클러스터(Cluster)의 특징

데이터 조회 속도 향상, 데이터 입력/수정/삭제 성능 저하, 데이터의 분포도가 넓을수록 유리, 저장 공간 절약, 처리 범위가 넓은 경우 단일 테이블 클러스터링, 조인이 많이 발생하는 경우에는 다중 테이블 클러스터링, 파티셔닝된 테이블은 클러스터링 사용X, 디스크 I/O 감소, 클러스터링 인덱스 생성 시 접근 성능 향상

3. 클러스터 대상 테이블 – 분포도 넓음, 대량의 범위 자주 조회, 자주 조인, 입력/수정/삭제 발생 낮음

38. 파티션 설계

1. 파티션(Partition)의 개요 : 대용량의 테이블이나 인덱스를 작은 논리적 단위인 파티션으로 나누는 것

물리적으로 별도의 공간에 데이터가 저장, 테이블 단위로 데이터 처리, 파티션별로 데이터 저장

2. 파티션의 장 단점

장점 : 액세스 범위를 줄여 쿼리 성능 향상, 데이터 분산 저장으로 디스크 성능 향상, 파티션별로 백업 및 복구 수행해 속도가 빠름, 시스템 장애 시 데이터 손상 최소화, 데이터 가용성 향상, 파티션 단위로 입 출력 분산

단점 : 테이블을 세분화하여 관리해 세심한 관리 요구, 조인 비용 증가, 용량이 작은 테이블은 오히려 성능 저하

3. 파티션의 종류

범위 분할(Range Partitioning) : 지정한 열의 값을 기준으로 분할

해시 분할(Hash Partitioning) : 해시 함수를 적용한 결과 값에 따라 데이터 분할, 특정 데이터 위치 판단 불가, 데이터가 고른 컬럼에 효과적, 특정 파티션에 데이터가 집중되는 범위 분할의 단점을 보완한 것

조합 분할(Composite Partitioning) : 범위 분할 후 해시 함수를 적용해 다시 분할, 관리가 어려울 때 유용

4. 파티션 키 선정 시 고려 사항

테이블 접근 유형에 따라 파티셔닝이 이뤄지도록 선정, 데이터 관리의 용이성을 위해 이력성 데이터는 파티션 생선주기와 소멸주기를 일치시켜야 함, 날짜/파티션간이동x/데이터 분포가 양호한 컬럼 등을 파티션키로 선정

5. 인덱스 파티션 – 파티션된 테이블의 데이터를 관리하기 위해 인덱스를 나눈 것

Local Partitioned Index : 테이블 파티션과 인덱스 파티션이 1:1 대응되도록 파티셔닝, 데이터 관리 상대적 쉬움

Global Partitioned Index : 테이블 파티션과 인덱스 파티션이 독립적으로 구성되도록 파티셔닝

Prefixed Partitioned Index : 인덱스 파티션키와 인덱스 첫 번째 컬럼이 같다.

Non-Prifixed Partitioned Index : 인덱스 파티션키와 인덱스 첫 번째 컬럼이 다르다.

39. 데이터베이스 용량 설계

데이터가 저장될 공간을 정의하는 것, 테이블에 저장할 데이터양과 인덱스, 클러스터 등이 차지하는 공간 예측

2. 데이터베이스 용량 설계의 목적

DB의 용량을 정확히 산정하여 디스크의 저장 공간을 효과적으로 사용하고 확장성 및 가용성 높임

디스크의 특성을 고려하여 설계해 디스크의 입 출력 부하를 분산시키고 채널의 병목 현상 최소화

디스크에 대한 입 출력 경합이 최소화되도록 설계해 데이터 접근성 향상

데이터 접근성을 향상시키는 설계 방법 : 테이블과 인덱스의 테이블스페이스를 분리하여 구성, 테이블을 마스터/트랜잭션 테이블로 분류, 테이블스페이스와 임시 테이블스페이스를 분리하여 구성,

3. 데이터베이스 용량 분석 절차

1. 데이터 예상 건수, 로우 길이, 보존 기간, 증가율 등 기초 자료를 수집하여 용량을 분석

2. 분석된 자료를 바탕으로 DBMS에 이용될 테이블, 인덱스 등 오브젝트별 용량을 선정

3. 테이블과 인덱스의 테이블스페이스 용량 산정

4. DB에 저장될 모든 데이터 용량과 데이터베이스 설치 및 관리를 위한 시스템 용량을 합해 디스크 용량 산정