**과목1 데이터 모델링의 이해**

**제1장 데이터 모델링의 이해**

**제1절 데이터 모델의 이해**

1. 모델링의 이해

- 현실세계를 **추상화, 단순화, 명확화** 하기 위해 일정한 표기법에 의해 표현하는 기법

- 특징

① **추상화** : 현실세계를 일정한 형식에 맞추어 간략하게 표현

② **단순화** : 복잡한 현실세계를 약속된 규약에 의해 제한된 표기법이나 언어로 표현

→ 누구나 쉽게 이해

③ **명확화** : 대상에 대한 애매모호함을 제거, 정확하게 현상 기술

- 세가지 관점

① 데이터 관점 : 비즈니스 프로세스에서 사용되는 데이터 의미 (업무가 어떤 데이터와 관련이 있는지, 데이터 간의 관계는 무엇인지에 대해 모델링)

- 구조분석, 정적 분석

② 프로세스 관점 : 수행하는 작업 의미 (업무가 실제하고 있는 일은 무엇인지, 무엇을 해야 하는지 모델링 - 시나리오 분석, 도메인 분석, 동적 분석

③ 상관 관점 : 업무가 처리하는 일의 방법에 따라 데이터는 어떻게 영향을 받고 있는지

모델링 - CRUD 분석

2. 데이터 모델의 기본 개념의 이해

- 정보시스템 구축을 위한 데이터 관점의 업무 분석기법

- 현실세계 데이터에 대해 약속된 표기법에 의해 표현하는 과정

- 데이터베이스 구축하기 위한 분석, 설계과정

- 분석된 모델을 가지고 데이터베이스를 생성하여 개발 및 데이터 관리에 사용

- 데이터모델링 자체로서 업무의 흐름을 설명하고 분석

- 기능 : 가시화, 명세화, 구조화, 문서화, 다양한 관점, 상세수준 표현

3. 데이터 모델링의 중요성 및 유의점

- 유의점 : 중복, 비유연성, 비일관성

- 중요성 : 파급효과가 크다 (Leverage)

복잡한 정보요구사항의 간결한 표현 (Conciseness)

데이터 품질을 유지 (Data Quality)

4. 데이터 모델링의 3단계 진행

|  |  |
| --- | --- |
| 데이터 모델링 단계 | 설명 |
| 개념적 모델링 단계 | - **전사적 관점**에서 기업의 데이터를 모델링  - **추상화 수준이 높고** 업무중심적이며 포괄적인 수준의 모델링  - 엔터티와 속성을 도출하고 ERD 작성 |
| 논리적 모델링 단계 | - 개념적 모델링을 논리적 모델링으로 변환  - 식별자를 정의하고 관계, 속성 등을 모두 표현  - 정규화를 통해 데이터 모델의 독립성을 확보하고 **재사용성을 높임** |
| 물리적 모델링 단계 | - 테이블, 인덱스, 함수를 생성하여 데이터베이스를 실제 구축  - **성능, 보안, 가용성 물리적인 성격 고려** |

5. 데이터 독립성

- **3층 스키마** : 3단계 계층으로 분리해서 데이터베이스의 독립성을 확보하기 위한 방법

|  |  |
| --- | --- |
| 독립성 | 설명 |
| 논리적 독립성 | 저장구조가 변경되어도 응용 프로그램 및 개념 스키마에 영향이 없음 |
| 물리적 독립성 | 데이터베이스 논리적 구조가 변경되어도 응용 프로그램에 변화가 없음 |

|  |  |
| --- | --- |
| 구조 | 설명 |
| 외부 스키마 | - **사용자 관점**, 업무상 관련이 있는 데이터 접근  - 관련 데이터베이스의 VIEW를 표시  - 응용 프로그램이 접근하는 데이터베이스를 정의 |
| 개념 스키마 | - 설계자 관점, 사용자 **전체 집단의 데이터베이스 구조**  - 전체 데이터베이스 내의 규칙과 구조 표현  - 통합 데이터베이스 구조 |
| 내부 스키마 | - 개발자 관점, 데이터베이스의 **물리적 저장구조**  - 데이터 저장구조, 레코드 구조, 필드 정의, 인덱스 의미 |

- 필요성 : 관리 및 유지보수 비용 절감, 데이터 중복 제거, 복잡성 감소, 요구사항 변경에

따른 대응력 향상

**제2절 엔터티(Entity)**

1. 엔터티의 개념

- 정의 : 업무에서 관리해야 하는 데이터집합, 저장되고 관리되어야 하는 데이터, 명사

- 특징

① 업무에서 **필요로 하는 정보**

② 유일한 **식별자**

③ **인스턴스의 집합** – 2개 이상의 인스턴스를 가져야 함

④ 업무 프로세스에 의해 이용

⑤ **속성을 포함**

⑥ 다른 엔터티와 최소 **1개 이상의 관계** (통계성 엔터티나 코드성 엔터티 제외)

2. 엔터티의 종류

- 유무형에 따른 분류 **(유개사)**

|  |  |
| --- | --- |
| 종류 | 설명 |
| 유형 엔터티 | - 업무에서 도출되며 지속적으로 사용되는 엔터티  - 물리적인 형태가 있고 안정적이며 지속적으로 활용  예) 고객, 강사, 사원 등 |
| 개념 엔터티 | - 물리적인 형태가 없으며 **관리해야 할 개념적 정보**로 구분  예) 거래소 종목, 코스닥 종목, 생명보험 상품 |
| 사건 엔터티 | - 비즈니스 프로세스를 실행하면서 생성되는 엔터티  - 발생량이 많고 각종 통계자료로 이용  예) 주문, 체결, 최소주문 수수료 청구 |

- 발생시점에 따른 분류 **(기중행)**

|  |  |
| --- | --- |
| 종류 | 설명 |
| 기본 엔터티  (키 엔터티) | - 다른 엔터티로부터 영향을 받지 않고 **독립적으로 생성**  - 타 엔터티의 부모 역할로 키엔터티라고도 함  - 다른 엔터티로부터 주식별자를 상속받지 않고 자신의 고유한 주 식별자를 가짐  예) 고객, 상품, 부서 등 |
| 중심 엔터티  (메인 엔터티) | - **기본 엔터티로부터 발생**되고 **행위 엔터티를 생성**  - 데이터 발생량이 많고 업무에 있어서 중심적인 역할  예) 계좌, 주문, 최소, 체결 등 |
| 행위 엔터티 | - **2개 이상의 부모 엔터티**로부터 발생  예) 주문 이력, 체결 이력 등 |

**제3절 속성(Attribute)**

1. 속성의 개념

- 정의 : 업무에서 필요로 하는 인스턴스의 구성요소로 관리하고자 하는 의미상 더 이상

분리되지 않는 데이터의 최소단위

- 특징

① 해당 업무에서 필요하고 관리하고자 하는 정보

② 정해진 **주식별자에 함수적으로 종속**

③ **하나의 값만 가짐**

♣ 엔터티와 인스턴스, 속성, 속성값의 관계

- 한 개의 엔터티는 두 개 이상의 인스턴스의 집합

- 한 개의 엔터티는 두 개 이상의 속성을 가짐

- 한 개의 속성은 한 개의 속성값을 가짐

2. 속성의 종류

- 분해여부에 따른 분류

|  |  |
| --- | --- |
| 종류 | 설명 |
| 단일 속성 | - 하나의 의미로 구성된 것으로 회원 ID, 이름 등 |
| 복합 속성 | - 여러 개의 의미가 있는 것으로 대표적으로 주소 |
| 다중값 속성 | - 속성에 **여러 개의 값을 가질 수 있는 것**으로 상품 리스트  - 별도의 엔터티로 분해 가능 |

- 특성에 따른 분류 **(기설파)**

|  |  |
| --- | --- |
| 종류 | 설명 |
| 기본 속성 | - 업무로부터 도출된 본래의 속성으로 가장 일반적이고 많음  예) 회원 ID, 이름, 계좌번호, 주문 일자 등 |
| 설계 속성 | - 데이터 모델링 과정에서 발생하는 속성  - 유일한 값을 부여  예) 상품코드, 지점코드 등 |
| 파생 속성 | - 다른 속성에 의해서 만들어지는 속성으로 보통 **합계, 평균** 등 |

3. 도메인 : 속성이 가질 수 있는 범위(문자열 포함)로 엔터티 내에서 속성에 대한 데이터 타입과 크기, 제약 사항을 지정

**제4절 관계(Relation)**

1. 관계의 개념

- 엔터티 간의 관련성으로 존재 관계(소속된다)와 행위 관계(주문한다)로 분류

- UMR : 연관관계와 의존관계로 구분

2. **관계 차수**(Cardinality)

- 두 개의 엔터티 간에 관계에 참여하는 수를 의미

- 하나의 릴레이션에서 튜플의 전체 개수

① 1대1 관계

|  |  |
| --- | --- |
| 종류 | 설명 |
| 완전 1대1 | - 하나의 엔터티에 관계되는 엔터티의 관계가 하나 있는 경우  - 반드시 존재 |
| 선택적 1대1 | - 하나의 엔터티에 관계되는 엔터티가 하나이거나 없는 경우 |

② M대N 관계

예) 한 명의 학생이 여러 개의 과목을 수강할 수 있고, 반대로 한 개의 과목은 여러

명의 학생이 수강 → 수강이라는 엔터티를 추가적으로 도출하여 1대N, N대1로

③ 필수적 관계와 선택적 관계

- 필수적 관계는 반드시 하나 존재해야 하는 관계이고 선택적 관계는 없을 수도 있는 관계

- 필수적 관계는 ‘|’로 표현되고 선택적 관계는 ‘O’로 표현

- 예) 고객은 반드시 있어야 계좌를 개설할 수 있음 → 필수적 관계

고객은 있지만 계좌가 없을 수도 있음 → 선택적 관계

3. 식별 관계와 비식별 관계

① **식별 관계**(강한 연결관계)

- 부모로부터 받은 식별자를 자식엔터티의 주식별자로 이용하는 것

- null값이 존재 안하므로 반드시 부모엔터티가 생성되어야 자기 자신의 엔터티 생성

- 실선으로 표현

- 모든 관계가 식별자로 연결되면 WHERE 절에서 비교하는 항목 증가 → 조인에 참여하

는 테이블에 따라 SQL 문이 길어져 복잡성 증가

② **비식별 관계**(약한 연결관계)

- 부모 엔터티로부터 속성을 받았지만 주식별자로 사용하지 않고 일반적인 속성으로만

사용하는 경우

- 점선으로 표현

♣ 관계 도출 시 체크 사항

- 두 개의 엔터티 사이에 관심있는 연관규칙이 존재하는가?

- 두 개의 엔터티 사이에 정보의 조합이 발생되는가?

- 업무기술서, 장표에 관계연결에 대한 규칙이 서술되어 있는가?

- 업무기술서, 장표에 관계연결을 가능하게 하는 동사가 있는가?

♣ 관계 읽기 : 각각의/하나의 기준엔터티는 관계차수(하나의/여러) 관련엔터티에 선택사

양(필수/선택) 관계명

**제5절 식별자**

1. 식별자의 개념

- 정의 : 엔터티를 대표할 수 있는 유일성을 만족하는 속성

- 주식별자의 특징

① **유일성**과 **최소성**을 만족하는 키

② 자주 변경되지 않아야 함(**불변성**)

③ 반드시 데이터 값이 존재(**존재성**)

♣ 키의 종류

|  |  |
| --- | --- |
| 데이터베이스 키 | 설명 |
| 기본키 | 후보키 중에서 엔터티를 대표할 수 있는 키 |
| 후보키 | 유일성과 최소성을 만족하는 키 |
| 슈퍼키 | 유일성은 만족하지만 최소성을 만족하지 않는 키 |
| 대체키 | 여러 개의 후보키 중에서 기본키를 선정하고 남은 키 |

2. 식별자의 종류

① 대표성 여부

- **주식별자** : 유일성과 최소성을 만족하면서 엔터티를 대표하는 식별자

다른 엔터티와 참조 관계로 연결될 수 있음

- **보조 식별자** : 유일성과 최소성은 만족하지만 대표성을 만족하지 못함

② 생성 여부

- **내부 식별자** : 엔터티 내부에서 스스로 생성 예) 부서코드, 주문번호 등

- **외부 식별자** : 다른 엔터티와의 관계를 통해 자식 쪽 엔터티에 생성

③ 속성의 수

- **단일 식별자** : 하나의 속성으로 구성

- **복합 식별자** : 두 개 이상의 속성으로 구성

④ 대체 여부

- **본질 식별자** : 비즈니스 프로세스에 의해 만들어짐

- **인조 식별자** : 인위적으로 만들어지는 식별자로 최대한 범용적인 값을 사용해 유일

한 값을 가짐

**제2장 데이터 모델과 성능**

**제1절 성능데이터 모델링의 개요**

1. 성능 데이터 모델링의 정의

: 데이터 베이스 성능 향상을 목적으로 설계단계의 데이터 모델링 때부터 정규화,

반정규화, 테이블 통합/분할, 조인구조, pk, fk 등 여러 가지 성능과 관련된 사항이

데이터 모델링에 반영될 수 있도록 하는 것

2. 성능 데이터 모델링 고려사항

① 정규화 ② 데이터 베이스 용량산정 ③ 발생되는 트랜잭션의 유형 파악

④ 반정규화 ⑤ 성능 관점에서 데이터 모델 검증

3. 좋은 모델링의 요건 : **중복배제, 업무 규칙, 완전성, 데이터 재사용, 의사소통, 통합성**

♣ Response Time = CPU(Service Time) + Queue(대기 시간)

Elapese Time = Response Time + Idle Time

**제2절 정규화**

1. 정규화를 통한 성능 향상 전략

- 데이터의 일관성, 최소한의 데이터 중복, 최대한의 데이터 유연성을 위한 방법

- 데이터를 분해하는 과정으로 데이터 중복을 제거하고 데이터 모델의 독립성 확보

- 함수적 종속성을 가지고 있는 일반 속성을 의존자로 하여 입력/수정/삭제 이상을

제거하여 성능 향상

2. 정규화 절차

① **제1정규화**

- 속성의 원자성 확보, 기본키 설정

② **제2정규화**

- 기본키가 2개 이상의 속성으로 이루어진 경우 부분 함수 종속성을 제거

③ **제3정규화**

- 기본키를 제외한 칼럼 간에 이행 함수 종속성을 제거

- 이행 함수 종속성 : 기본키를 제외하고 칼럼간에 종속성이 발생하는 것

④ **BCNF**

- 기본키를 제외하고 후보키가 있는 경우 후보키가 기본키를 종속시키면 분해

⑤ 제4정규화

- 여러 칼럼들이 하나의 칼럼을 종속시키는 경우 분해하여 다중 값 종속성 제거

⑥ 제5정규화

- 조인에 의해서 종속성이 발생되는 경우 분해

**제3절 반정규화**

1. 반정규화를 통한 성능 향상 전략

- 데이터 중복을 허용하고 조인을 줄이는 데이터베이스 성능 향상 방법

- 조회 속도를 향상하지만 데이터 모델의 유연성은 낮아짐

- 반정규화 정보에 대한 재현의 적시성으로 판단

- 데이터를 조회할 때 디스크 I/O 량이 많아서 성능이 저하되거나 경로가 너무 멀어 조

인으로 인한 성능저하가 예상되거나 칼럼을 계산하여 읽을 때 성능이 저하될 것이 예

상되는 경우 반정규화 수행

2. 반정규화를 수행하는 경우

- 정규화에 충실하면 종속성, 활용성은 향상되지만 수행 속도가 느려지는 경우

- 다량의 범위를 자주 처리해야 하는 경우

- 특정 범위의 데이터만 자주 처리하는 경우

- 요약/집계 정보가 자주 요구되는 경우

3. 반정규화 절차

① 대상 조사 및 검토

- 데이터 처리 범위, 통계성 등을 확인해서 반정규화 대상을 조사

② 다른 방법 검토 (대상 데이터의 크기가 클 때)

- 클러스터링, 뷰, 인덱스 튜닝, 응용 프로그램, 파티션 등을 검토

③ 반정규화 수행

- 테이블, 속성, 관계 등을 반정규화

- 테이블, 칼럼의 반정규화는 데이터 무결성에 영향을 주지만 관계의 반정규화

는 데이터 무결성을 깨트리지 않으면서도 데이터 처리 성능을 향상

♣ 다른 방법 처리

- 지나치게 많은 조인이 걸려 데이터를 조회하는 작업이 기술적으로 어려울 경우

VIEW 사용

- 대량의 데이터처리나 부분처리에 의해 성능이 저하되는 경우에 클러스터링을 적용하

거나 인덱스를 조정함으로써 성능 향상

- 대량의 데이터는 PK 성격에 따라 파티셔닝 기법을 적용하여 성능저하 방지

- 응용 어플리케이션에서 로직을 구사하는 방법을 변경함으로써 성능 향상

4. 반정규화 기법

ㆍ **테이블의 반정규화**

① 테이블 병합

- 1대1 관계의 테이블을 하나의 테이블로 병합

- 1대N 관계의 테이블을 병합 but 많은 양의 데이터 중복 발생

- 슈퍼타입/서브타입 테이블 병합

② 테이블 분할

- 수직분할 : 칼럼을 분할하여 새로운 테이블을 만듦

- 수평분할 : 하나의 테이블에 있는 값을 기준으로 테이블 분할

③ 테이블 추가

- 중복테이블 추가 : 분산데이터베이스에서 원격으로 조인할 때 사용

- 통계테이블 추가 : 사전에 합계, 평균 등을 계산하여 조회 성능 향상

계좌평가 시 평가금액을 배치로 계산해서 추가

- 이력테이블 추가 : 마스터 테이블의 데이터를 이력 테이블에 추가하여 이력 관리

- 부분테이블 추가 : 자주 이용되는 칼럼에 대해서 디스크 IO량을 감소시키기 위해 해당 칼럼을 별도로 분리

ㆍ **칼럼의 반정규화**

① 중복칼럼 추가

② 파생칼럼 추가

③ 이력테이블 칼럼 추가

④ PK에 의한 칼럼 추가

⑤ 응용시스템 오작동을 위한 칼럼 추가

ㆍ 관계의 반정규화 – 중복관계 추가

**제4절 대량 데이터에 따른 성능**

- 트랜잭션 집중 → 분산 처리 될 수 있도록 테이블 단위에서 수평/수직 분할

- 로우 체이닝 : 로우 길이가 너무 길어서 데이터 블록 하나에 데이터가 모두 저장되지 않고 두 개 이상 저장

- 로우 마이그레이션 : 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터

블록에 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식

1. 대량 데이터 저장 및 처리로 인한 성능

- 파티셔닝을 적용하거나 pk에 의해 테이블 분할을 통해 성능 향상

- 파티셔닝을 사용하면 논리적으로는 하나의 테이블이지만 여러 개의 데이터 파일에 분

산되어 저장

- 파티션 전용 인덱스 생성 가능

♣ 파티션 기법

- Range partition : **데이터 값의 범위를 기준으로 파티션** 수행

대상 테이블을 **날짜 또는 숫자값으로 분리** 가능

데이터 보관 주기에 따라 테이블 관리 용이

- List partition : **특정 값을 지정하여 파티션** 수행

지점, 사업장, 핵심적인 코드값 등으로 pk 구성에 적용

- Hash partition : **해시함수를 적용하여 파티션** 수행

- Composite partition : **범위와 해시를 복합적**으로 사용하여 파티션 수행

♣ 파티션 인덱스

|  |  |
| --- | --- |
| 구분 | 주요 내용 |
| Global Index | 여러 개의 파티션에서 하나의 인덱스를 사용 |
| Local Index | 해당 파티션 별로 각자의 인덱스를 사용 |
| Prefixed Index | 파티션 키와 인덱스 키가 동일 |
| Non Prefixed Index | 파티션 키와 인덱스 키가 다름 |

**제5절 데이터베이스 구조와 성능**

1. **슈퍼타입/서브타입 모델**의 성능고려 방법

- 업무를 구성하는 데이터의 특징을 공통과 차이점의 특징을 고려하여 효과적으로 표

현

- 공통의 부분을 슈퍼타입으로 모델링하고 공통으로부터 상속받아 다른 엔터티와 차이가 있는 속성에 대해서는 별도의 서브 엔터티로 구분

- 데이터양이 많이 존재하고 지속적으로 증가하는 양도 많다면 슈퍼/서브 타입에 대해 물리적인 데이터 모델로 변환하는 세 가지 유형에 대해 세심하게 적용

|  |  |
| --- | --- |
| 변환방법 | 설명 |
| One to One Type | - 슈퍼/서브 타입 개별로 트랜잭션 발생 → 개별로 테이블 구성  - 테이블 수가 많아서 조인이 많이 발생하고 관리가 어려움 |
| Plus Type | - 슈퍼+서브 타입에 대해 트랜잭션 발생  - 조인이 발생하고 관리가 어려움 |
| Single Type | - 전체를 하나로 묶어 트랜잭션 발생 → 하나의 테이블로 구성  - 조인 성능이 좋고 관리가 편하지만 I/O 성능이 나쁨 |

2. 인덱스 특성을 고려한 PK/FK 데이터베이스 성능 향상

- PK 순서를 인덱스 특징에 맞게 고려하지 않고 그대로 생성하게 되면 테이블에 접근하는 트랜잭션의 특징에 효율적이지 않은 인덱스가 생성되므로 인덱스의 범위를 넓게 이용하거나 FULL SCAN을 유발하게 되어 성능이 저하됨

3. 물리적인 테이블에 FK 제약이 걸려있지 않은 경우 인덱스 미생성으로 성능 저하

- 제약을 걸었을 때는 반드시 FK 인덱스를 생성하도록 하고 걸리지 않았을 경우에는 FK 인덱스를 생성하는 것을 기본정책으로 하되 발생되는 트랜잭션에 의해 거의 활용되지 않았을 때에만 FK 인덱스를 지우는 것이 적절

**제6절 분산 데이터베이스와 성능**

1. 분산 데이터베이스

- 중앙 집중형 데이터베이스 : 한 대의 물리적 시스템에 데이터베이스 관리 시스템을

설치하고 여러 명의 사용자가 데이터베이스 관리 시스템이 접속하여 데이터베이스

를 사용하는 구조

- 분산 데이터베이스 : 물리적으로 떨어진 데이터베이스에 네트워크로 연결하여 단일 데이터베이스 이미지를 보여주고 분산된 작업 처리를 수행하는 데이터베이스

- 데이터베이스를 연결하는 빠른 네트워크 환경을 이용하여 데이터베이스를 여러 노트로 위치시켜 사용성, 성능을 극대화 시킨 데이터베이스

- 분산 데이터베이스는 사용하는 고객이 시스템이 네트워크로 분산되어 있는지의 여부를 인식하지 못하면서, 자신만의 데이터베이스를 사용하는 것 같은 투명성을 제공

- 투명성의 종류

|  |  |
| --- | --- |
| 투명성 | 설명 |
| 분할 투명성 | 고객은 하나의 논리적 릴레이션이 여러 단편으로 분할되어 각 단편의 사본이 여러 시스템에 저장되어 있음을 인식할 필요 X |
| 위치 투명성 | 고객이 사용하려는 데이터의 저장 장소를 명시할 필요 X  고객은 데이터가 어느 위치에 있더라도 동일한 명령을 사용하여 데이터에 접근 |
| 지역 사상 투명성 | 지역 DBMS와 물리적 데이터베이스 사이의 사상이 보장됨에 따라 각 지역 시스템 이름과 무관한 이름 사용 가능 |
| 중복 투명성 | 데이터베이스 객체가 여러 시스템에 중복되어 존재함에도 고객과는 무관하게 데이터의 일관성 유지 |
| 장애 투명성 | 데이터베이스가 분산되어 있는 각 지역의 시스템이나 통신망에 이상이 발생해도, 데이터베이스의 무결성 보장 |
| 병행 투명성 | 여러 고객의 응용 프로그램이 동시에 분산 데이터베이스에 대한 트랜잭션을 수행하는 경우에도 결과에 이상 X |

2. 장단점

|  |  |
| --- | --- |
| 장점 | 단점 |
| - 데이터베이스 신뢰성과 가용성이 높음  - 병렬 처리를 수행하기 때문에 빠른 응답 가능  - 시스템 용량 확장이 쉬움 | - 관리와 통제 어려움  - 보안관리의 어려움  - 데이터 무결성 관리가 어려움  - 설계 관리의 복잡성과 비용 |

3. 적용기법

① **테이블 위치 분산** : 테이블의 구조는 변하지 않고 테이블이 다른 데이터베이스에 중복 생성 X

② **테이블 분할 분산** : 각각의 테이블을 쪼개어 분산

- 수평분할 : 모든 데이터가 각 지사별로 분리되어 있는 형태

각지사의 데이터들은 항상 베타적으로 존재

데이터를 한군데 집합시켜 놓아도 pk에 의해 중복 X

- 수직분할 : 각각의 테이블에는 동일한 pk와 값을 가지고 있어야 함

데이터 중복 X

③ **테이블 복제 분산** : 동일한 테이블을 다른 지역이나 서버에서 동시에 생성하여 관

리하는 유형

- 부분 복제 : 여러 테이블에 조인이 발생하지 않은 빠른 작업 수행 가능

본사와 지사간 데이터 중복

본사와 지사 양쪽 모두 데이터를 수정하여 전송하는 경우 데이터의 정합성 일치가 힘들기 때문에 가능하면 한쪽(지사)에서 데이터 수정 후 본사로 복제

- 광역 복제 : 통합된 테이블을 각 지사와 본사 모두 가지고 있는 형태

본사에서는 데이터를 관리하고 지사에서는 이 데이터를 읽어 업무 프로세스 진행

④ **테이블 요약 분산** : 지역 간 또는 서버 간에 데이터가 서로 다른 유형으로 존재

- 분산 요약 : 각 지사 별로 존재하는 요약정보를 본사에 통합하여 다시 전체에 대해서 요약정보를 산출

지사 간 통일 내용을 요약, 통합 통계 데이터에 대한 정보제공 용이

- 통합 요약 : 각 지사 별로 존재하는 다른 내용의 정보를 본사에 통합

지사에서 산출한 요약정보를 한 군데 취합하여 보여주는 형태

각 지사 별 데이터 비교에 용이

♣ 선택도 : 특정 조건에 의해서 선택될 것으로 예상되는 레코드의 비율

조건절에서 요청한 값의 범위/전체값

카디널리티 = 선택도 \* 전체 레코드 수

♣ 데이터베이스 분산설계를 적용하여 효율성을 증대시키는 경우

- 공통코드, 기준정보 등 마스터 데이터는 분산데이터베이스에 복제분산 적용

- 거의 실시간 업무적인 특성을 가지고 있을 땐 분산데이터베이스를 사용하여 구성

- 백업 사이트를 구성할 때 간단하게 분산 기능을 적용하여 구성

**과목2 SQL 기본 및 활용**

**제1장 SQL 기본**

**제1절 관계형 데이터베이스 개요**

1. SQL

- 정의 : 관계형 데이터베이스에서 데이터 정의, 조작, 제어를 위해 사용하는 언어

- 종류

|  |  |
| --- | --- |
| 종류 | 설명 |
| DDL  (data definition language) | - 관계형 데이터베이스의 구조를 정의하는 언어  - create, alter, drop, rename |
| DML  (data manipulation language ) | - 테이블에서 데이터를 입력, 수정, 삭제, 조회  - insert, update, delete, select |
| DCL  (data control language) | - 데이터베이스 사용자에게 권한을 부여하거나 회수  - grant, revoke |
| TCL  (transaction control language) | - DML에 의해 조작된 결과를 트랜잭션 별로 제어  - commit, rollback |

- 작업 순서 : TCL로 데이터베이스의 사용자에게 권한을 부여 → DDL로 데이터 구조 정의 → 데이터 입력 후 DML로 데이터 조회

- 실행 순서 : 파싱(Parsing) → 실행(Execution) → 인출(Fetch)

2. **트랜잭션**

- 정의 : 데이터베이스의 작업을 처리하는 단위

- 특성

① **원자성** : 데이터베이스 연산의 전부 또는 일부 실행 (all or nothing)

트랜잭션의 처리가 완전히 끝나지 않았을 경우는 전혀 이루어지지 않는 것과 같아야 함

② **일관성** : 트랜잭션 실행 결과로 데이터베이스의 상태가 모순되지 않아야 함

트랜잭션 실행 후에도 일관성 유지

③ **고립성** : 트랜잭션 실행 중에 생성하는 연산의 중간결과는 다른 트랜잭션이 접근할

수 없음 (부분적인 실행결과를 다른 트랜잭션이 볼 수 없음)

④ **연속성** : 트랜잭션이 그 실행을 성공적으로 완료하면 그 결과는 영구적으로 보장

- Dirty Read : 다른 트랜잭션에 의해 수정되었지만 아직 커밋되지 않은 데이터를 읽는 것

**제2절 DDL**

1. **CREATE**

- 새로운 테이블 생성

- 테이블 생성시 기본키, 외래키, 기타 제약사항 등을 설정할 수 있음

CREATE TABLE 테이블명 (

칼럼명1 데이터유형 제약조건,

칼럼명2 데이터유형 제약조건

);

- 제약조건

① 기본키 : constraint 기본키이름 primary key (칼럼명1,칼럼명2)

(pk = unique + not null)

② 외래키 : constraint 외래키이름 foreign key (칼럼명)

references 테이블명 (칼럼명)

③ not null

④ default 예) default 0, default sysdate

⑤ 고유키(Unique key)

⑥ check

- ON DELETE CASCADE : 자신이 참조하고 있는 테이블의 데이터가 제거되면 자동으로 자신의 데이터도 삭제 (참조 무결성 유지)

(vs. ON INSERT AUTOMATIC)

- ON DELETE SET NULL : 자신이 참조하고 있는 테이블의 데이터가 제거되면 해당 행에서 참조되는 외래 키를 구성하는 모든 값이 Null로 설정

(vs. ON INSERT DEPENDENT)

- 테이블명 변경 : ALTER TABLE 테이블명

RENAME TO 새로운 테이블명; (o)

sp\_rename 테이블명, 새로운 테이블명; (s)

- 테이블 구조 확인 : DESC 테이블명; (o) sp\_help ‘dbo.테이블명’; (s)

- SELET 문을 이용한 테이블 생성 : CREATE TABLE 새로운 테이블명

AS SELECT \* FROM 테이블명; (o)

SELECT \* INTO 새로운 테이블명 FROM 테이블명; (s)

(주의사항 : 제약조건 중 NOT NULL 만 새로운 테이블에 적용)

2. **ALTER**

- 칼럼 추가 : ALTER TABLE 테이블명

ADD (칼럼명 데이터유형 제약조건);

- 칼럼 변경 : ALTER TABLE 테이블명

MODIFY (칼럼명 데이터유형 제약조건); (o)

ALTER TABLE 테이블명

ALTER COLUMN 칼럼명 데이터유형 제약조건; (s)

♣주의사항

- 칼럼의 크기를 늘릴 순 있지만 줄이진 못함

- null 값만 가지거나 아무 행도 없으면 줄이기, 데이터 유형 변경 가능

- default 값을 바꾸면 변경 이후 발생하는 행에만 영향

- null 값이 없는 경우에만 not null 추가 가능

- 칼럼 삭제 : ALTER TABLE 테이블명

DROP COLUMN 칼럼명:

- 칼럼명 변경 : ALTER TABLE 테이블명

RENAME COLUMN 칼럼명 TO 새로운 칼럼명; (o)

sp\_rename 칼렴명, 새로운 칼럼명, ‘COLUMN’; (s)

- 제약조건 추가/삭제 : ALTER TABLE 테이블명

ADD CONSTRAINT 제약조건명 제약조건 (칼럼명);

ALTER TABLE 테이블명

DROP CONSTRAINT 제약조건명;

3. **DROP**

- 테이블 구조와 데이터 모두 삭제, 디스크 사용량 초기화, 스키마 정의 삭제

DROP TABLE 테이블명 CASCADE CONSTRAINT;

- SQL server 에는 CASCADE 옵션이 존재하지 않으므로 테이블을 삭제하기 전 FK 제약조

건 또는 참조하는 테이블을 먼저 삭제

4. **뷰**

- VIEW : 테이블로 유도된 가상의 테이블로 실제 데이터를 가지고 있지 않고 테이블을 참조해서 원하는 칼럼만을 조회할 수 있음

♣ 뷰의 특징

- 참조한 테이블이 변경되면 뷰도 변경

- 뷰의 검색은 참조한 테이블과 동일하게 할 수 있지만, 뷰에 대한 **입력, 수정, 삭제시에**

**는 제약 발생**

- 특정 칼럼만 조회시켜 보완성 향상

- 한번 생성된 뷰는 변경할 수 없고 변경을 원하면 삭제 후 재생성

- ALTER문으로 변경 X

- 뷰 생성 : CREATE VIEW 뷰이름 AS

SELECT \* FROM 테이블명;

- 뷰 삭제 : DROP VIEW 뷰이름;

**제3절 DML**

- 비절차적 DML : 사용자가 무슨 데이터를 원하는 지만 명세

- 절차적 DML : 어덯게 데이터를 접근해야 하는지 명세 (PL/SQL, T-SQL)

- 데이터 부속어 : 호스트 프로그램 속에 삽입되어 사용되는 DML 명령어

1. **INSERT**

INSERT INTO 테이블명 (칼럼명1,칼럼명2) VALUES (데이터1,데이터2);

- 특정 테이블의 모든 칼럼에 삽입하는 경우 칼럼명 생략

- SELECT문을 사용하여 데이터를 조회해서 해당 테이블에 바로 삽입 가능

INSERT INTO 입력할 테이블명

SELECT \* FROM 테이블명;

- NOLOGGING : **INSERT 문의 성능 향상을 위해** BUFFER CACHE의 기록을 생략하는 옵션

ALTER TABLE 테이블명 NOLOGGING;

2. **UPDATE**

- 원하는 조건으로 검색해서 해당 데이터 수정

- 조건문을 입력하지 않으면 모든 데이터 수정

UPDATE 테이블명

SET ENAME = ‘조조’

WHERE EMPNO = 100;

(EMPNO가 100인 직원의 ENAME을 조조로 수정)

3. **DELETE**

- 원하는 조건을 검색해서 해당되는 행 삭제

- 데이터를 삭제한다고 해서 테이블의 용량이 초기화 되지 않음 (vs TRUNCATE)

- 테이블 전체 삭제 시 시스템 부하가 적은 TRUNCATE 권고 (but rollback 불가)

- TRUNCATE는 무결성을 확인하지 않고 로그를 기록하지 않아 데이터를 빠르게 삭제

DELETE FROM EMP

WHERE EMPNO = 100;

|  |  |  |
| --- | --- | --- |
| DROP | TRUNCATE | DELETE |
| DDL | DDL | DML |
| ROLLBACK 불가능 | ROLLBACK 불가능 | ROLLBACK 가능 |
| AUTO COMMIT | AUTO COMMIT | 사용자 COMMIT |
| 테이블의 정의 자체를 완전히 삭제 | 테이블을 최초 생성된 초기상태로 만듬 | 데이터만 삭제 |

4. **SELECT**

- ORDER BY : 특별한 지정이 없으면 오름차순으로 정렬 (oracle 에서는 SELECT에 없는 칼럼으로도 정렬 가능)

SELECT \* FROM 테이블명

ORDER BY 칼럼명1, 칼럼명2 DESD;

(칼럼명1에 대해서는 오름차순, 2에 대해서는 내림차순으로 정렬)

- 인덱스를 사용한 정렬 회피

SELECT /\*+ INDEX\_DESC(A) \*/

FROM 테이블명 A;

- DISTINCT : 칼럼명 앞에 지정하여 중복된 데이터를 한번만 조회

- ALIAS : 공백, 특수문자를 포함하거나 대소문자 구분이 필요한 경우 “” 사용

**제4절 TCL**

1. **COMMIT**

- 변경한 데이터를 데이터베이스에 반영하여 다른 모든 데이터베이스 사용자가 변경된 데

이터를 조회하거나 조작할 수 있음

- 이전 데이터는 잃어버리고 COMMIT을 실행하면 하나의 트랜잭션 과정이 종료됨

2. **ROLLBACK**

- 데이터에 대한 변경 사항을 모두 취소하고 트랜잭션 종료

- 이전에 COMMIT한 곳까지만 복구

3. **SAVEPOINT**

- 트랜잭션을 작게 분할하여 관리하는 것으로 지정된 위치까지만 트랜잭션을 롤백

SAVEPOINT t1;

♣ SQL server 에서의 트랜잭션

① AUTO COMMIT

- DML, DDL을 수행할 때마다 DBMS가 트랜잭션을 컨트롤

② 암시적 트랜잭션

- 트랜잭션의 시작은 DBMS가 처리하고 끝은 사용자가 처리

- 인스턴스 단위 또는 세션 단위로 설정 가능

③ 명시적 트랜잭션

- 트랜잭션의 시작과 끝을 모두 사용자가 지정하는 방식

- BEGIN RTANSACTION 으로 시작하며 ROLLBACK 수행 시 최초의 BEGIN

TRANSACTION 시점까지 모두 ROLLBACK 수행

**제5절 WHERE**

1. SQL 연산자

|  |  |
| --- | --- |
| 연산자 | 설명 |
| LIKE ‘비교 문자열’ | 비교 문자열 조회, 와일드카드( % , \_ )  % : 어떤 것을 포함하는 모든 것 의미  예) ‘조%’ - 조로 시작하는 모든 문자 조회  \_ : 한 개의 단일 문자만을 의미 |
| BETWEEN A AND B | A와 B 사이의 값 조회 |
| IN (list) | OR을 의미하며 list 값 중 하나만 일치해도 조회 |
| IS NULL | NULL 값 조회 |

2. NULL 관련 함수

|  |  |
| --- | --- |
| NULL 함수 | 설명 |
| NVL  /ISNUL 함수 | NULL이면 다른 값으로 바꾸는 함수  예) NVL(MGR,0) - MGR 칼럼이 NULL이면 0으로 바꿈 |
| NVL2 함수 | NVL 함수와 DECODE를 하나로 만든 것  예) NVL2(MGR,1,0) - MGR 칼럼이 NULL이 아니면 1을  NULL이면 0을 출력 |
| NULLIF 함수 | 두 개의 값이 같으면 NULL을 같지 않으면 첫 번째 값 출력  예) NULLIF(exp1,exp2) - exp1과 exp2가 같으면 null을  같지 않으면 exp1을 출력 |
| COALESCE | 예) COALESCE(MGR,1) - MGR이 NULL이 아니면 1 출력 |

- oracle 에서 공백은 null, ms-sql 에서는 공백으로 입력

- 집계함수와 scalar subquery의 경우 결과값이 공집합이어도 null 출력

- 다중행 함수(집계함수, 그룹함수, 윈도우함수)는 일부만 null 인 경우 대상에서 제외하므

로 불필요하게 null 관련 함수를 이용할 필요 x

- **ROWNUM** : SELECT문의 결과에 일련번호를 부여하거나 조회되는 행 수를 제한

2개 이상의 행을 가져올 때는 FROM절에 SELECT문을 사용하는 inline view 이용

SELECT \*

FROM (SELECT ROWNUM list, ENAME

FROM EMP)

WHERE list BETWEEN 5 AND 10;

- TOP() [WITH TIES] ~ ORDER BY

- **ROWID** : 데이터를 구분할 수 있는 유일한 값으로 데이터가 어떤 데이터 파일, 블록에

저장되어 있는지 알 수 있음

|  |  |  |
| --- | --- | --- |
| 구조 | 길이 | 설명 |
| 오브젝트 번호 | 1~6 | 오브젝트 별로 유일한 값을 가지고 있으며, 해당 오브젝트가 속해 있는 값 |
| 상대 파일번호 | 7~9 | 테이블스페이스에 속해 있는 데이터 파일에 대한 상대 파일번호 |
| 블록 번호 | 10~15 | 데이터 파일 내부에서 어느 블록에 데이터가 있는지 알려줌 |
| 데이터 번호 | 16~18 | 데이터 블록에 데이터가 저장되어 있는 순서 의미 |

- **WITH** : 서브쿼리를 사용해서 임시 테이블이나 뷰처럼 사용할 수 있는 구문

서브쿼리 블록에 별칭 지정, 옵티마이저는 SQL을 인라인 뷰나 임시 테이블로 판단 (?)

**제6절 함수**

1. DUAL 테이블

- oracle 데이터베이스에 의해 자동으로 생성되는 테이블

- 모든 사용자가 임시로 사용할 수 있는 테이블로 내장형 함수를 실행할 때도 사용

가능

- DUMMY라는 문자열 유형의 칼럼에 X 라는 값을 가지는 1개의 행 포함

- 단일행 함수는 SELECT, WHERE, ORDER BY, UPDATE의 SET 절에 사용 가능

2. 내장 함수(Built-In Function)

- 문자형 함수

|  |  |
| --- | --- |
| 문자형 함수 | 설명 |
| ACSII(문자) | 문자 혹은 숫자를 ACSII 코드 값으로 반환 |
| CHAR(ACSII 코드 값) | ACSII 코드 값을 문자로 변환 |
| SUBSTR(문자열,m,n) | 문자열에서 m번째 위치부터 n개를 자름 |
| CONCAT(문자열1,문자열2) | 문자열1과 문자열2 결합  Oracle 은 || , ms-sql 은 + 사용 가능 |
| LOWER(문자열) | 영문자를 소문자로 변환 |
| UPPER(문자열) | 영문자를 대문자로 변환 |
| LENGTH / LEN(문자열) | 공백을 포함한 문자열의 길이를 알려줌 |
| LTRIM(문자열,지정문자) | 왼쪽에 지정된 문자를 삭제  지정된 문자를 생략하면 공백을 삭제 |
| RTRIM(문자열,지정문자) |  |
| TRIM(문자열,지정문자) | 왼쪽 및 오른쪽에 지정된 문자를 삭제 |

- 날짜형 함수

|  |  |
| --- | --- |
| 날짜형 함수 | 설명 |
| SYSDATE | 오늘의 날짜를 날짜형 타입으로 알려줌 |
| EXTRACT(‘YEAR’/’MONTH’/’DAY’ FROM d) | 날짜에서 년, 월, 일을 조회 |

- 숫자형 함수

|  |  |
| --- | --- |
| 숫자형 함수 | 설명 |
| ABS(숫자) | 절댓값 |
| SIGH(숫자) | 양수, 음수, 0 구별 |
| MOD(숫자1,숫자2) | 숫자1을 숫자2로 나눈 나머지 계산 (= %) |
| CEIL/CEILING(숫자) | 크거나 같은 최대 정수 |
| FLOOR(숫자) | 작거나 같은 최대 정수 |
| ROUND(숫자,m) | 소수점 m자리에서 반올림, m의 default 값은 0 |
| TRUNC(숫자,m) | 소수점 m자리에서 버림, default값 동일 |

♣ SQL server 의 변환형 함수

CAST (expression AS 데이터타입);

CONVERT (데이터타입, expression);

3. **DECODE** : 특정 조건이 참이면 A, 거짓이면 B

DECODE(EMPNO,1000,’TRUE’,’FALSE’)

FROM EMP;

4. **CASE** : 조건문

SELECT CASE

WHEN EMPNO = 1000 THEN ‘A’

WHEN EMPNO = 1001 THEN ‘B’

ELSE ‘C’

END

FROM EMP;

- SEARCHED\_CASE\_EXPRESSION : WHEN 칼럼명 = ‘’ THEN ‘’

- SIMPLE\_CASE\_EXPRESSION : 칼럼명 WHEN ‘’ THEN ‘’

**제7절 GROUP BY, HAVING**

1. GROUP BY

- 테이블에서 소규모 행을 그룹화하여 합계, 평균, 최댓값, 최솟값 등을 계산

- ORDER BY 를 사용해서 정렬 가능(GROUP BY 표현식인 경우)

- ALIAS 사용불가

2. HAVING

- GROUP BY에 조건절을 사용할 때 이용

- 집계 함수 이용 가능

3. 집계 함수 : COUNT/SUM/AVG/MAX/MIN/STDDEV/VARIAN

- **집계 함수는 WHERE 절에 올 수 없음**

4. SELECT문의 실행 순서

: **FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY**

**제9절 JOIN (WHERE 절에 기술)**

1. EQUI 조인

- 두 개의 테이블 간에 칼럼 값들이 서로 정확히 일치하는 경우에 사용

- 대부분 PK,FK 관계 기반

- WHERE절에 기술

SELECT \* FROM EMP, DEPT

WHERE EMP.DEPTNO = DEPT.DEPTNO;

2. NON-EQUI 조인

- 정확하게 일치하지 않는 것을 조인

- 부등호 사용

**제2장 SQL 활용**

**제1절 표준조인 (FROM 절에 기술)**

1. INNER 조인

- JOIN 조건에서 동일한 값이 있는 행만 반환

- USING 조건절이나 ON 조건절 필수적으로 사용

- 조인에 사용된 같은 이름의 칼럼을 별개의 칼럼으로 표시

- 조인 칼럼에 대해서는 ALIASF 를 붙일 수 없음

SELECT \* FROM EMP (INNER) JOIN DEPT

ON EMP.DEPNO = DEPT.DEPTNO;

2. NATURAL 조인

- 두 테이블 간의 동일한 이름으르 갖는 모든 칼럼들에 대해 EQUI 조인 수행

- 추가로 USING 조건절, ON 조건절, WHERE 절에서 JOIN 조건 정의 X

- 조인에 사용된 같은 이름의 칼럼을 하나로 처리

- 조인 칼럼에 대해서는 ALIAS나 테이블 이름과 같은 접두사를 붙일 수 없음

3. USING 조건절

- 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI 조인

- 조인에 사용된 같은 이름의 칼럼을 하나로 철

- 조인 칼럼에 대해서는 ALIAS나 테이블 이름과 같은 접두사를 붙일 수 없음

4. ON 조건절

- USING 조건절과 달리 칼럼명이 다르더라도 조인 조건 사용 가능

- ALIAS나 테이블 명과 같은 접두사를 사용하여 SELECT 에 사용되는 칼럼을 논리적으로 명확하기 지정해주어야 함

- FROM 절에 테이블이 많이 사용될 경우 다소 복잡하게 보여 가독성이 떨어지는 단점

- 데이터 검증 조건 추가 가능

5. OUTER 조인

- JOIN 조건에서 동일한 값이 없는 행도 NULL 값으로 반환

- USING, ON 조건절 필수 사용

- LEFT/RIGHT/FULL (OUTER) JOIN

6. CROSS 조인

- 조인 조건이 없는 경우 생길 수 있는 모든 데이터의 조합

SELECT \* FROM EMP CROSS JOIN DEPT;

♣ FROM 절에 기술한 테이블 순서대로 조인하기 위한 힌트

oracle – ORDERED, MS-SQL – option(force order)

**제2절 집합 연산자**

- 2개 이상의 조회 결과를 하나의 결과로 반환

- SELECT 절의 칼럼 수가 같고 데이터 타입이 상호 호환 가능해야 함

- UNION, UNION ALL, INTERSECT, MINUS/EXCEPT

**제3절 계층형 조회**

- **계층형 조회** : 트리 형태의 구조를 탐색하면서 조회

SELECT MAX(LEVEL)

FROM

WHERE

START WITH 조건

CONNECT BY PRIOR [NOCYCLE] 조건

[ORDER SIBILINGS BY 칼럼명];

( START WITH : 시작 조건, 루트 데이터 지정

CONNECT BY : 조인 조건, 다음에 전개될 자식 데이터 지정

PRIOR 자식 = 부모 (순방향) : 부모데이터에서 자식데이터로 전개

PRIOR 부모 = 자식 (역방향) : 반대

CONNECT BY 키워드)

|  |  |
| --- | --- |
| 키워드 | 설명 |
| LEVEL | 검색 항목의 깊이 의미, 계층 구조에서 가장 상위 레벨은 1 |
| CONNECT\_BY\_ROOT | 현재 데이터의 루트 데이터 표시 |
| CONNECT\_BY\_ISLEAF | 해당 데이터가 리프면 1, 아니면 0 |
| SYS\_CONNECT\_BY\_PATH | 루트 데이터부터 현재까지의 경로를 표시  ex) SYS\_CONNECT\_BY\_PATH(칼럼, 경로분리자) |
| NOCYCLE | 순환구조 발생 지점까지만 전개 |
| CONNECT\_BY\_ISCYCLE | 순환구조 발생 지점을 표시 |

- ORDER SIBILING BY : 동일 LEVEL 사이에 정렬

- WHERE : 모든 전개 수행 후 지정된 조건을 만족하는 데이터만 추출

- MAX(LEVEL) : 최대 계층의 수를 구함

- LPAD(‘’,숫자) : 숫자만큼 공백 추가

**제4절 서브쿼리**

1. 서브쿼리

- SQL 문 안에 포함되어 있는 또 다른 SQL 문

- 서브쿼리는 메인 쿼리의 칼럼을 모두 사용 가능하지만 메인 쿼리는 서브쿼리의 칼럼 사

용 X

- 서브쿼리에서는 ORDER BY 절 사용 X

- 서브쿼리의 형태는 **FROM 구에 SELECT 문을 사용하는 인라인 뷰와 SELECT 문에 서브**

**쿼리를 사용하는 스칼라 서브쿼리** 등이 있음

- **보통 WHERE 구에 SELECT 문을 사용하면 서브쿼리**라고 함

2. 단일 행 서브쿼리와 다중 행 서브쿼리

- 단일 행 서브쿼리 : 반환하는 행 수가 한 개인 서브쿼리로 비교연산자 사용

- 다중 행 서브쿼리 : IN, ANY, ALL, EXIST 사용

|  |  |
| --- | --- |
| 다중 행 연산자 | 설명 |
| IN(서브쿼리) | 메인 쿼리의 비교조건이 서브쿼리의 결과 중 하나만 동일하면 참 |
| ALL(서브쿼리) | 메인 쿼리와 서브쿼리의 결과가 모두 동일하면 참  < : 서브쿼리의 최소값보다 작은 모든 건이 조건 만족  > : 서브쿼리의 최대값보다 큰 모든 건이 조건 만족 |
| ANY(서브쿼리) | 메인 쿼리의 비교조건이 서브쿼리의 결과 중 하나 이상 동일하면 참  < : 서브쿼리의 최대값보다 작은 모든 건이 조건 만족  > : 서브쿼리의 최소값보다 큰 모든 건이 조건 만족 |
| EXISTS(서브쿼리) | 메인 쿼리와 서브쿼리의 결과가 하나라도 존재하면 참 |

3. 다중 칼럼 서브쿼리

4. 연관 서브쿼리

- 서브쿼리 내에서 메인 쿼리 내의 칼럼을 사용

- EXISTS 서브쿼리는 항상 연관 서브쿼리로 사용

4. SELECT 절에 서브쿼리 사용 (스칼라 서브쿼리)

- 반드시 한 행과 한 칼럼만을 반환하는 서브쿼리

- 여러 행이 반환되면 오류 발생

5. FROM 절에서 서브쿼리 사용 (인라인 뷰)

- 인라인 뷰에는 ORDER BY 절 사용할 수 있음

- 인라인 뷰에서 먼저 정렬을 수행하고 정렬된 결과 중에서 일부 데이터를 추출하는 것을

TOP-N 쿼리라고 함

- TOP-N 쿼리 수행을 위해서는 정렬 작업과 정렬 결과 중에서 일부 데이터만을 추출하는

방법(ROWNUM)이 필요

6. HAVING 절에서 서브쿼리 사용

- 평균키가 삼성 블루윙즈팀의 평균키보다 작은 팀의 이름과 해당 팀의 평균키를 구하는

SQL 문

SELECT P.TEAM\_ID 팀코드, T.TEAM\_NAME 팀명, AVG(P.HEIGHT) 평균키

FROM PLAYER P, TEAM T

WHERE P.TEAM\_ID = T.TEAM\_ID

GROUP BY P.TEAM\_ID, T.TEAM\_NAME

HAVING AVG(P.HEIGHT) < (SELECT AVG(HEIGHT)

FROM PLAYER

WHERE TEAM\_ID=’K02’);

7. VIEW

**-** 뷰 사용의 장점

① 독립성 : 테이블 구조가 변경되어도 뷰를 사용하는 응용 프로그램은 변경하지 않아도

됨

② 편리성 : 복잡한 질의를 뷰로 생성함으로써 관련 질의를 단순하게 작성 또한 해당 형

태의 SQL 문을 자주 사용할 때 뷰를 이용하면 편리

③ 보안성 : 직원의 급여정보와 같이 숨기고 싶은 정보가 존재한다면 뷰를 생성할 때 해

당 칼럼을 빼고 생성함으로써 정보를 감출 수 있음

**제5절 그룹함수**

1. **ROLLUP**

- **GROUP BY 의 칼럼에 대해서 subtotal**을 만들어 줌

- GROUP BY 구에 칼럼이 두 개 이상 오면 순서에 따라서 결과가 달라짐

- GROUP BY ROLLUP (칼럼명1, 칼럼명2, 칼럼명3)

- ORDER BY 절로 정렬 가능

= GROUP BY (칼럼명1, 칼럼명2, 칼럼명3) + GROUP BY (칼럼명1, 칼럼명2) + GROUP BY (칼럼명1) + 총합계

- 괄호로 묶은 칼럼들의 경우 하나의 집합 칼럼으로 간주하여 괄호 내 각 칼럼별 집

계를 구하지 않음 예) GROUPING(칼럼명1,(칼럼명2,칼럼명3))

2. **GROUPING SETS**

- GROUP BY 에 나오는 칼럼의 순서와 관계없이 원하는 부분의 소계만 추출

- 괄호로 묶은 집합 별로 집계를 구할 수 있음

3. **CUBE**

- 제시한 칼럼에 대해서 결합 가능한 모든 집계를 계산

- 조합할 수 있는 경우의 수가 모두 조합

4. **GROUPING**

- 생성되는 합계 값을 구분하기 위해 만들어진 함수로 소계, 합계 등이 계산되면 1

그렇지 않으면 0 반환

- SELECT 절에 CASE/DECODE 를 이용해 원하는 문자열 지정 가능

CASE GROUPING(칼럼명) WHEN 1 THEN ‘~’ ELSE ‘~’ END

DECODE(GROUPING(칼럼명),1,’~’,’~”)

**제6절 윈도우 함수**

1. 윈도우 함수

- 행과 행 간의 관계를 정의하기 위해서 제공되는 함수

- 순위, 합계, 평균, 행 위치 등을 조작

SELECT WINDOW\_FUNCTION(ARGUMENTS(인수)

OVER (PARTITION BY 칼럼명

ORDER BY WINDOWING 절)

FROM 테이블명;

- ARGUMENTS(인수) : 0~N 개의 인수를 설정

- WINDOWING : 행 기준의 범위를 정함

ROWS는 물리적 결과의 행 수

RANGE는 논리적인 값에 의한 범위

|  |  |
| --- | --- |
| 구조 | 설명 |
| ROWS | 부분집합인 윈도우 크기를 물리적 단위로 행의 집합 지정 |
| RANGE | 논리적인 값에 의해 행 집합을 지정 |
| BETWEEN ~ AND | 윈도우의 시작과 끝의 위치 지정 |
| UNBOUNDED PRECEDING | 윈도우의 시작 위치가 첫 번째 행임을 의미 |
| UNBOUNDED  FOLLOWING | 윈도우 마지막 위치가 마지막 행임을 의미 |
| CURRENT ROW | 윈도우 시작 위치가 현재 행임을 의미, 누적 합계 |

2. 순위함수

- 특정 항목과 파티션에 대해 순위 계산

- 순위 관련 윈도우 함수

- ORDER BY 절과 함께 사용 가능

|  |  |
| --- | --- |
| 순위함수 | 설명 |
| RANK | 동일한 순위는 동일한 값 부여 예) 1 2 2 4 |
| DENSE\_RANK | 동일한 순위를 하나의 건수로 계산 예) 1 2 2 3 |
| ROW\_NUMBER | 동일한 순위에 대해서 고유의 순위를 부여 예) 1 2 3 4 |

RANK () OVER ([PARTITION BY] JOB ORDER BY SAL DESC)

3. 집계함수

- SUM/MAX/MIN : 파티션별 윈도우의 합/최댓값/최솟값을 구할 수 있음

SUM(칼럼명1) OVER (PARTITION BY 칼럼명2 [ORDER BY 칼럼명1] [WINDOWING절])

- AVG/COUNT : 파티션별 ROWS 윈도우를 이용해 원하는 조건에 맞는 데이터에 대한 통계값을 구할 수 있음

4. 행 순서 함수 (oracle만 지원)

- FIRST\_VALUE/LAST\_VALUE : 파티션별 윈도우에서 가장 먼저 나온 값/가장 나중에 나

온 값을 구함

- **LAG/LEAD** : 파티션별 윈도우에서 이전 몇 번째/이후 몇 번째 행의 값을 가져옴

LAG/LEAD(칼럼명,몇 번째,NULL 다른 값으로 변경) OVER (ORDER BY 칼럼명)

5. 비율 함수

|  |  |
| --- | --- |
| 비율 함수 | 설명 |
| CUME\_DIST | - 파티션 전체 건수에서 현재 행보다 작거나 같은 건수에 대한 **누적 백분율**을 조회  - 누적 분포상에 위치로 0 에서 1 사이의 값 |
| PERCENT\_RANK | 파티션에서 제일 먼저 나온 것을 0으로 제일 늦게 나온 것을 1로 하여 값이 아닌 행의 순서 별 백분율을 조회 |
| NTILE | 파티션별로 전체 건수를 ARGUMENT 값으로 N등분한 결과 조회  예) NTILE(나눌 수) OVER (ORDER BY 칼럼명) |
| RATIO\_TO\_REPORT | 파티션 내에 전체 SUM(칼럼)에 대한 행 별 칼럼 값의 백분율을 소수점까지 조회 |

**제7절 DCL**

1. **GRANT**

- 데이터베이스 사용자에게 권한 부여

GRANT 권한 ON 테이블명 TO 사용자 [WITH GRANT OPTION]

(권한 : SELECT, INSERT, UPDATE, DELETE, PEFERENCES, ALTER, INDEX, ALL)

- WITH GRANT OPTION

|  |  |
| --- | --- |
| GRANT 옵션 | 설명 |
| WITH GRANT OPTION | - 특정 사용자에게 권한을 부여할 수 있는 권한 부여 |
| WITH ADMIN OPTION | - 테이블에 대한 모든 권한 부여 |

2. **REVOKE**

- 데이터베이스 사용자에게 부여된 권한을 회수

REVOKE 권한 ON 테이블명 FROM 사용자

**제2장 SQL 최적화의 원리**

**제1절 옵티마이저와 실행계획**

1. 옵티마이저

- SQL을 어떻게 실행할 것인지에 대한 실행계획을 수립하고 실행하는 데이터베이스 관리 시스템 소프트웨어

- 동일한 결과가 나오는 SQL도 어떻게 실행하느냐에 따라서 성능이 달라지므로 옵티마이저의 실행계획은 SQL 성능에 중요한 역할

- 옵티마이저는 데이터 딕셔너리에 있는 오브젝트 통계, 시스템 통계 등의 정보를 사용해서 예상되는 비용을 산정

- 여러 개의 실행계획 중에서 최저비용을 가지고 있는 하나를 선택하여 실행

- DESC PLAN\_TABLE; 을 통해 실행계획을 확인할 수 있음 (엑세스 기법, 질의처리 예상비용, 조인순서)

- SQL 처리 흐름도 : 실행계획을 시각화해서 표현한 것

2. 옵티마이저 엔진

|  |  |
| --- | --- |
| 엔진 | 설명 |
| Query Transformer | - SQL문을 효율적으로 실행하기 위해 옵티마이저가 변환  - SQL이 변환되어도 결과는 동일 |
| Estimator | 통계정보를 사용해서 SQL 실행비용 계산 |
| Plan Generatio | SQL을 실행할 실행계획 수립 |

3. 규칙기반 옵티마이저

- 우선순위를 기준으로 실행계획을 수립

|  |  |
| --- | --- |
| 우선순위 | 설명 |
| 1 | **ROWID를 사용한 단일 행**인 경우 |
| 2 | 클러스터 조인에 의한 단일 행인 경우 |
| 3 | 유일하거나 기본키를 가진 해시 클러스터 키에 의한 단일 행인 경우 |
| 4 | 유일하거나 기본키에 의한 단일 행인 경우 |
| 5 | 클러스터 조인인 경우 |
| 6 | 해시 클러스터 조인인 경우 |
| 7 | 인덱스 클러스터 키인 경우 |
| 8 | 복합 칼럼 인덱스인 경우 |
| 9 | 단일 칼럼 인덱스인 경우 |
| 10 | 인덱스가 구성된 칼럼에서 제한된 범위를 검색하는 경우 |
| 11 | 인덱스가 구성된 칼럼에서 무제한 범위를 검색하는 경우 |
| 12 | 정렬-병합(Sort Merge) 조인인 경우 |
| 13 | 인덱스가 구성된 칼럼에서 MAX 혹은 MIN을 구하는 경우 |
| 14 | 인덱스가 구성된 칼럼에서 ORDER BY 를 실행하는 경우 |
| 15 | 전체 테이블을 스캔 하는 경우 |

- 힌트 : /\*+ RULE \*/

3. 비용기반 옵티마이저

- 오브젝트 통계 및 시스템 통계를 사용해서 총비용 계산

- 통계 정보가 부적절한 경우 성능 저하 발생

- 총비용이란 SQL문을 실행하기 위해서 예상되는 소요시간 혹은 자원의 사용량 의미

- 통계 정보, DBMS 버전, DBMS 설정 정보 등의 차이로 동일 SQL 문에도 서로 다른 실행 계획이 생성될 수 있음

- 다양한 한계들로 인해 실행계획의 예측 및 제어가 어렵다는 단점

♣ View Merging

- 옵티마이저가 SQL 실행을 최적화하기 위해서 SQL 문을 가공하는 작업

- 집합연산자, connect by절, ROWNUM, SELECT 구에 집계합수, 분석함수를 사용하는 경우 View Merging 불가능

**제2절 인덱스**

1. 인덱스

- 원하는 데이터를 빠르게 검색할 수 있는 방법 제공

- 인덱스는 인덱스 키로 정렬되어 있기 때문에 원하는 데이터를 빠르게 조회

- 오름차순 및 내림차순 탐색 가능

- 하나의 테이블에 여러 개의 인덱스를 생성할 수 있고 하나의 인덱스는 여러 개의 칼럼으로 구성될 수 있음

- 테이블 생성시 기본키는 자동으로 인덱스가 만들어지고 이름은 SYS

CREATE INDEX 인덱스명 ON 테이블명(칼럼명);

- INDEX 검색 → ROWID를 사용해서 테이블 조회 → 조인

- 조인 시 먼저 조회되는 테이블을 Outer Table 그 다음에 조회되는 테이블을 Inner

Table이라 함

- UPDATE 작업에는 부하가 없을 수도 있음

2. 트리기반 인덱스

- DBMS 에서 가장 일반적인 인덱스는 B-트리 인덱스

- 인덱스의 구조는 Root Block, Branch Block, Leaf Block 로 구성되고 Root Block 는 인덱스 트리에서 가장 상위에 있는 노드를 의미하며 Branch Block 은 다음 단계의 주소를 가지고 있는 포인터로 되어있음

- Leaf Block 은 인덱스를 구성하는 칼럼의 데이터(인덱스 키)와 해당 데이터를 가지고 있는 행의 위치를 가리키는 레코드 식별자(ROWID)로 구성

- 동등 검색과 범위 검색 모두 적합

- 일반적으로 테이블 내의 데이터 중 10% 이하의 데이터를 검색할 때 유리

3. SQL server의 클러스터형 인덱스

- 인덱스의 리프페이지가 곧 데이터 페이지

- 리프페이지의 모든 로우는 인덱스 키 칼럼 순으로 물리적으로 정렬되어 저장됨

- oracle 의 IOT 와 유사

3. 인덱스 스캔

① 인덱스 유일 스캔 : 인덱스의 키 값이 중복되지 않는 경우 해당 인덱스를 사용할 때

발생

② 인덱스 범위 스캔 : SELECT문에서 특정 범위를 조회하는 WHERE문(LIKE, BETWEEN)

을 사용할 경우, 인덱스의 Leaf Block의 특정 범위를 스캔

③ 인덱스 전체 스캔 : 인덱스에서 검색되는 인덱스 키가 많은 경우에 Leaf Block을

처음부터 끝까지 전체를 읽어 들임

♣ 인덱스를 사용할 수 없는 조건 (null 이나 형변환이 일어난 경우)

- NVL (Key, 0)

- to\_char (vintagedate, ’yyyymmdd’) = sysdate

- name || ‘’ = ‘lim’

♣ 전체 스캔 시 High Water Mark의 의미

- 테이블을 읽을 때 HWM 이하까지만 전체 스캔

- HWM은 테이블에 데이터가 저장된 블록에서 최상위 위치를 의미

**제3절 조인 수행 원리**

1. **Nested Loop 조인**

- 중첩된 반복문과 유사한 방식으로 조인을 수행

- 하나의 테이블에서 데이터를 먼저 찾고 다음 테이블을 조인하는 방식

- 외부 테이블의 크기가 작은 것을 먼저 찾는 것이 중요

- RANDOM ACCESS(외부 테이블에서 내부 테이블의 인덱스를 찾는 것)의 양을 줄여야 성능 향상

- OLTP의 목록 처리 업무에 많이 사용됨 (DW는 Sort Merge나 Hash에서 많이 사용)

- 조인 칼럼에 적당한 인덱스가 있어서 자연조인이 효율적일 때 유용

- 유니크 인덱스를 활용하여 수행시간이 적게 걸리는 소량 테이블을 온라인 조회하는 경우 유용

2. **Sort Merge 조인**

- 두 개의 테이블을 SORT\_AREA라는 메모리 공간에 모두 로딩하고 SORT 수행

- 두 개의 테이블에 대해 SORT가 완료되면 두 개의 테이블 병합(Merge)

- 주로 스캔 방식으로 데이터를 읽음

- 랜덤 액세스로 NL 조인에서 부담이 되던 넓은 범위의 데이터를 처리

- 정렬이 발생하므로 데이터 양이 많아 지면 성능이 떨어짐

- 정렬 데이터의 양이 너무 많은 경우 정렬은 임시 영역에서 수행

- 기본키와 외래키 관계에서 외래키에 인덱스가 없을 때 옵티마이저가 sort merge 조인

- 조인 조건의 인덱스의 영향을 받지 않음

3. **Hash 조인**

- 조인을 수행할 테이블의 조인 칼럼을 기준으로 해시함수를 수행하여 서로 동일한 해시 값을 갖는 것들 사이에서 실제 값이 같은지를 비교하면서 조인 수행

- NL 조인의 랜덤 액세스 문제와 Sort Merge 조인의 문제점인 정렬 작업의 부담의 해결을 위한 대안으로 등장

- 동등 조인에서만 사용 가능

- 결과 행의 수가 적은 테이블을 선행 테이블로 사용하는 것이 좋음

- 두 개의 테이블 중에서 작은 테이블을 HASH 메모리에 로딩하고 두 개의 테이블의 조인 키를 사용해서 테이블 생성

- 해시함수를 사용해서 주소를 계산하고 해당 주소를 사용해서 테이블을 조인하므로 CPU 연산이 많음