

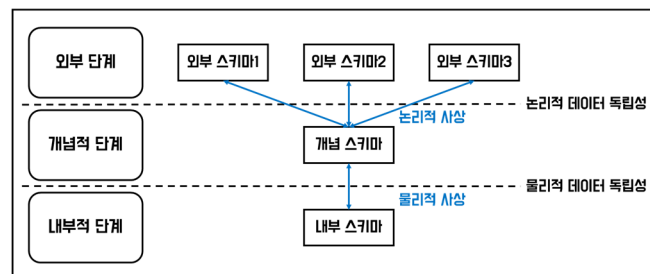
1과목 데이터 모델링의 이해

1장 데이터 모델링의 이해

1절 데이터 모델의 이해

- 모델링: 현실세계를 단순화하여 표현하는 것
 - ✓ 특징
 - 추상화: 일정한 형식에 맞춰 표현함
 - 단순화: 제한된 표기법이나 언어로 표현함
 - 명확성: 이해가 쉽게 표현함
 - ✓ 관점
 - 데이터 관점: 업무와 데이터 및 데이터 사이의 관계
 - 프로세스 관점: 진행되고 있거나 진행되어야 하는 업무
 - 상관 관점: 데이터에 대한 업무 처리 방식의 영향
- 데이터 모델링: 정보 시스템 구축을 위한 데이터 관점의 업무 분석 기법
 - ✓ 목적: 1) 정보에 대한 표기법을 통일하여 업무 내용 분석 정확도 증대 2) 데이터 모델을 기초로 DB 생성
 - ✓ 기능: 1) 가시화 2) 명세화 3) 구조화된 틀 제공 4) 문서화 5) 다양한 관점 제공 6) 구체화
 - ✓ 중요성
 - 파급효과(Leverage)
 - 간결한 표현(Conciseness): 정보 요구사항과 한계를 간결하게 표현하는 도구
 - 데이터 품질
 - 유일성: 데이터 중복 저장 방지
 - 유연성: 데이터 정의와 데이터 사용 프로세스 분리
 - 일관성
 - ✓ 이해관계자: 1) 개발자 2) DBA 3) 모델러 4) 현업업무전문가, 완성된 모델을 정확히 해석할 수 있어야 함
- 데이터 모델링 3단계
 - ① 개념적 모델링: 엔터티와 속성을 도출하고 ERD를 작성함, 업무 중심적이고 포괄적인 수준의 모델링
 - ② 논리적 모델링: 식별자를 도출하고 속성과 관계 등을 정의함, 정규화를 수행하여 데이터 모델의 독립성과 재사용성 확보, 논리 데이터 모델은 데이터 모델링 완료 상태
 - ③ 물리적 모델링: DB를 구축함, 성능 및 보안 등 물리적인 성격 고려

※ 프로젝트 생명주기(Life Cycle): 계획 > 분석 > 설계 > 개발 > 테스트 > 전환/이행 단계로 구성됨, 1) 계획과 분석 단계는 개념적 모델링 2) 분석 단계는 논리적 모델링 3) 설계 단계는 물리적 모델링에 해당
- DB의 3단계 구조: 데이터 독립성 확보를 목표로 함



- ✓ DB 독립성의 필요성: 데이터의 중복성과 데이터 복잡도 증가로 인한 1) 유지보수 비용 증가 2) 요구사항 대응 저하
- ✓ 3층 스키마(3-level Schema)
 - 외부 스키마: 각 사용자 단계의 개인적 DB 스키마, 사용자 관점, 응용 프로그램이 접근하는 DB를 정의함
 - 개념 스키마: 조직 전체의 통합된 DB 스키마, 설계자 관점 *데이터 모델링의 지향점*
 - 내부 스키마: 물리적으로 데이터가 저장되는 방법을 표현하는 스키마, 개발자 관점, 물리적 저장 구조임
- ✓ 데이터 독립성

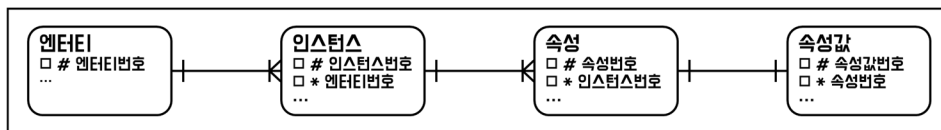
- 논리적 독립성: 외부 스키마가 개념 스키마의 변화에 무관함, 논리적 사상 없음
 - 물리적 독립성: 개념 스키마가 내부 스키마의 변화에 무관함, 물리적 사상 없음
5. 데이터 모델링 3요소: 엔터티, 관계, 속성
 6. ERD(Entity Relationship Diagram): 1) 엔터티는 사각형 2) 관계는 마름모 3) 속성은 타원형으로 표현, 현실의 데이터 모두 표현 가능
 - ① 엔터티 도출
 - ② 엔터티 배치
 - ③ 엔터티 간 관계 설정
 - ④ 관계명 기술
 - ⑤ 관계차수 표현: 1:1, 1:N, M:N
 - ⑥ 관계선택사양 표현: 필수, 선택
 7. 좋은 모델링의 요건: 1) 완전성 2) 중복 배제 3) 업무 규칙 4) 데이터 재사용 5) 의사소통 6) 통합성

2절 엔터티

1. 정의: 업무에서 관리해야 하는 데이터의 집합, 명사형, 인스턴스의 집합
2. 특징: 1) 업무에서 필요로 함 2) 유일한 식별자를 가짐 3) 2개 이상의 인스턴스를 포함함 4) 업무 프로세스에 이용됨 5) 속성을 가짐 6) 관계를 가짐
3. 종류
 - ✓ 유무형에 따른 분류
 - 유형 엔터티: 물리적 형태가 있고 지속적으로 활용되는 엔터티
 - 개념 엔터티: 물리적 형태가 없는 엔터티
 - ✓ 발생시점에 따른 분류
 - 기본 엔터티(Key Entity): 독립적으로 생성되는 엔터티
 - 중심 엔터티(Main Entity): 기본 엔터티와 행위 엔터티의 중간에 존재하는 엔터티
 - 행위 엔터티(Active Entity, 사건 엔터티): 2개 이상의 부모 엔터티로부터 발생함, 비즈니스 프로세스를 실행하면서 생성되는 엔터티, 지속적으로 정보가 추가되고 변경되어 데이터양이 가장 많음
4. 명명 규칙: 1) 현업업무에서 사용하는 용어 2) 약어 지양 3) 단수 명사 4) 유일성 보장 5) 명확성

3절 속성

1. 정의: 엔터티가 가지는 최소 의미 단위, 인스턴스의 구성요소
2. 엔터티와 인스턴스 및 속성과 속성값 간의 관계



3. 속성 표기법: IE 표기법, Barker 표기법
4. 특징: 1) 업무에서 필요하고 관리하고자 하는 정보 2) 주식별자에 함수적으로 종속됨 3) 속성값 하나만 가짐 (하나 이상의 속성값이면 정규화 필요)
5. 종류
 - ✓ 특성에 따른 분류
 - 기본 속성: 비즈니스 프로세스에서 도출되는 본래의 속성
 - 설계 속성: 데이터 모델링 과정에서 업무 규칙화를 위해 발생하는 속성
 - 파생 속성: 다른 속성에 의해 만들어지는 속성 (↔ 저장 속성은 유도 속성을 생성하는 데 사용되는 속성)
 - ✓ 분해 가능 여부에 따른 분류
 - 단일 속성: 하나의 의미
 - 복합 속성: 여러 의미, 단일 속성으로 분해 가능
 - ex) 주소
 - 단일값 속성: 하나의 값
 - 다중값 속성: 여러 값, 엔터티로 분해 가능

- ✓ 엔터티 구성방식에 따른 분류
 - 기본키 속성: 엔터티를 식별할 수 있는 속성
 - 외래키 속성: 다른 엔터티와의 관계에서 포함된 속성
 - 일반 속성: 엔터티에 포함되고 PK나 FK 속성이 아닌 속성

6. 도메인: 속성이 가질 수 있는 값의 범위

4절 관계

1. 정의: 엔터티 간의 논리적인 관련성, 동사형
2. 관계의 패어링: 인스턴스 간 개별적 관계
3. 관계 표기법: 1) 관계명 2) 관계차수 3) 관계선택사양
 - ✓ 관계차수(Cardinality): 관계 내 튜플의 전체 개수, 1은 직선 多는 삼발로 표시
 - M:N 관계: 관계형 DB에서 M:N 관계의 조인은 카테시안 곱 발생
 - ✓ 관계선택사양(Optionality): 필수는 I 선택은 O로 표시
4. 종류
 - ✓ ERD 기준: 표기구분 안함
 - 존재 관계: 엔터티 간의 상태
 - 행위 관계: 엔터티 간에 발생하는 행위
 - ✓ UML(Unified Modeling Language) 기준
 - 연관 관계(Association): 실선 표기
 - 의존 관계(Dependency): 점선 표기
 - ✓ 식별자에 따른 분류
 - 식별 관계: 부모 엔터티의 식별자를 자식 엔터티에서 주식별자로 사용
 - ※ 약한 엔터티: 부모 엔터티에 종속되어 존재 (↔ 강한 엔터티는 독립적으로 존재함)
 - 비식별 관계: 부모 엔터티의 식별자를 자식 엔터티에서 일반 컬럼으로 참조 사용, 약한 종속 관계
 - ※ 식별 관계만으로 연결되면 주식별자 수가 많아질 수밖에 없으므로 1) 관계 강약 분석 2) 자식 엔터티의 독립 PK 필요성 3) SQL 복잡성과 개발 생산성 고려 필요
5. 관계 읽기: 각각의/하나의 > 기준 엔터티 > 관계차수 > 대상 엔터티 > 관계선택사양 > 관계명

5절 식별자

1. 정의: 엔터티를 대표할 수 있는 유일성을 만족하는 속성
2. 특징: 1) 유일성 2) 최소성 3) 불변성 4) 존재성
3. 종류
 - ✓ 대표성 여부에 따른 분류
 - 주식별자: 대표성을 만족하는 식별자
 - 보조 식별자: 유일성과 최소성만 만족하는 식별자, 참조 관계 연결에 사용할 수 없음
 - ※ DB 키의 종류
 - 기본키(PK; Primary Key): 엔터티를 대표하는 키, 후보키 중 선정됨
 - 후보키: 유일성과 최소성을 만족하는 키
 - 슈퍼키: 유일성만 만족하는 키
 - 대체키: 기본키를 제외한 나머지 후보키
 - 외래키(FK; Foreign Key): 여러 테이블의 기본 키 필드, 참조 무결성(Referential Integrity)을 확인하기 위해 사용됨 (허용된 데이터 값만 저장하기 위함)
 - ✓ 생성 여부에 따른 분류
 - 내부 식별자: 자연스럽게 존재하는 식별자 (~ 본질식별자)
 - 외부 식별자: 다른 엔터티와의 관계를 통해 생성되는 식별자
 - ✓ 속성 수에 따른 분류
 - 단일 식별자: 하나의 속성

- 복합 식별자: 여러 속성
- ✓ 대체 여부에 따른 분류
 - 본질 식별자: 대체될 수 없는 식별자
 - 인조 식별자: 인위적으로 만들어지는 대체가능한 식별자 (순서번호(Sequence Number)를 사용하여 생성된 식별자), 1) 후보 식별자 중 주식별자로 선정할 것이 없거나 2) 주식별자가 너무 많은 칼럼으로 구성되어 있을 때 사용

4. 주식별자 도출 기준: 업무에서 자주 이용되는 속성, 이름 명명 지양, 복합 식별자 지양

2장 데이터 모델과 성능

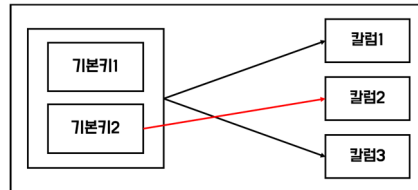
1절 성능 데이터 모델링의 개요

1. 성능 데이터 모델링: DB 성능향상을 위한 사항이 데이터 모델링에 반영되도록 하는 것
2. 수행 시점: 분석/설계 단계, 성능 데이터 모델링 시점이 늦어질수록 재업무 비용이 증가함
3. 고려 사항: 정규화 수행, DB 용량 산정과 트랜잭션 유형 파악을 통한 반정규화 수행, *정규화는 무조건 해야 됨*

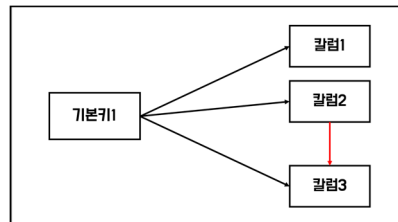
2절 정규화와 성능

1. **정규화(Normalization): 데이터 분해 과정, 이상현상(anomaly) 제거**
 - ✓ **정규형(NF; Normal Form):** 정규화로 도출된 데이터 모델이 갖춰야 할 특성
2. 함수적 종속성(FD; Functional Dependency): 결정자와 종속자의 관계, 결정자의 값으로 종속자의 값을 알 수 있음
 - ✓ 다치 종속(MVD; Multivalued Dependency): 여러 칼럼이 동일한 결정자의 종속자일 때
3. 정규화 이론: 1) 1차 2차 3차 보이스코드 정규화는 함수적 종속성에 근거 2) 4차 정규화는 다치 종속을 제거 3) 5차 정규화는 조인에 의한 이상현상을 제거하여 정규화를 수행함
 - ✓ **1차 정규화:** 속성의 원자성 확보, 다중값 속성을 분리함
 - ✓ **2차 정규화:** 부분 함수 종속성 제거, 일부 기본키에만 종속된 속성을 분리함, 기본키가 하나의 칼럼일 때 생략 가능

- 부분 함수 종속성



- ✓ **3차 정규화:** 이행 함수 종속성 제거, 서로 종속관계가 있는 일반속성을 분리함, 주식별자와 관련성이 가장 낮음
- 이행 함수 종속성



- ✓ **보이스코드 정규화(BCNF; Boyce-Codd Normal Form):** 후보키가 기본키 속성 중 일부에 함수적 종속일 때 다수의 주식별자를 분리함
 - ✓ **4차 정규화, 5차 정규화:** 다치 종속 분리, 결합 종속 분리
4. **정규화와 성능:** 정규화는 입출력 데이터의 양을 줄여 성능을 향상시킴
- ✓ **정규화로 인한 성능 향상:** 입력/수정/삭제 시 성능은 항상 향상됨
 - 유연성 증가: High Cohesion & Loose Coupling 원칙에 충실해짐
 - 재활용 가능성 증가: 개념이 세분화됨
 - 데이터 중복 최소화
 - ✓ **정규화로 인한 성능 저하:** 조회 시 처리 조건에 따라 성능 저하가 발생할 수도 있음
 - 데이터 조회 시 조인을 유발하여 CPU와 메모리를 많이 사용하게 됨
 - 반정규화로 해결 가능
 - 조인이 발생하더라도 인덱스를 사용하여 조인 연산을 수행하면 성능 상 단점이 거의 없고, 정규화를 통해

필요한 인덱스의 수를 줄일 수 있음

- 정규화를 통해 소량의 테이블이 생성된다면 성능 상 유리할 수 있음

3절 반정규화와 성능

1. 반정규화(Denormalization): 데이터 중복을 허용하여 조인을 줄이는 DB 성능 향상 방법, 데이터의 무결성을 희생하고 조회 성능 향상
2. 절차
 - ① 반정규화 대상 조사: 데이터 처리 범위 및 통계성 등 조사
 - ② 다른 방법 검토: 1) 뷰 2) 클러스터링 3) 인덱스 4) 애플리케이션
 - ③ 반정규화 적용: 정규화 수행 후 반정규화 수행
3. 기법
 - ✓ 테이블 반정규화
 - 테이블 병합
 - 1:1 관계 테이블 병합
 - 1:N 관계 테이블 병합: 많은 데이터 중복 발생
 - 슈퍼타입/서브타입 테이블 병합
 - 테이블 분할: 1) 수직분할 2) 수평분할
 - 테이블 추가
 - 중복 테이블: 업무나 서버가 다를 때 중복 테이블 생성 (원격조인 제거)
 - 통계 테이블
 - 이력 테이블
 - 부분 테이블: 자주 이용하는 칼럼으로 구성된 테이블 생성
 - ✓ 칼럼 반정규화
 - 중복 칼럼 추가
 - 파생 칼럼 추가: 필요한 값 미리 계산한 칼럼 추가
 - 이력 테이블 칼럼 추가
 - PK에 의한 칼럼 추가: PK의 종속자를 일반속성으로 생성
 - 응용 시스템의 오작동을 위한 칼럼 추가
 - ✓ 관계 반정규화: 데이터 무결성 보장 가능
 - 중복 관계 추가

4절 대용량 데이터에 따른 성능: 테이블 반정규화 중 테이블 분할 관련

1. 블록: 테이블의 데이터 저장 단위
 2. 대량 데이터 발생으로 인한 현상: 블록 I/O 횟수 증가 → 디스크 I/O 가능성 상승 (디스크 I/O 시 성능 저하)
 - ✓ 로우 체이닝(Row Chaining): 행 길이가 너무 길어 여러 블록에 걸쳐 저장되는 현상
 - ✓ 로우 마이그레이션(Row Migration): 수정된 데이터가 해당 블록이 아닌 다른 블록의 빈 공간에 저장되는 현상
 3. 테이블 분할: 반정규화 기법
 - ✓ 수직분할: 칼럼 단위로 테이블을 분할하여 I/O를 감소시킴, 너무 많은 수의 칼럼이 있는 경우 사용
 - ✓ 수평분할: 행 단위로 테이블을 분할하여 I/O를 감소시킴
 4. 파티셔닝(Partitioning): 테이블 수평분할 기법, 논리적으로는 하나의 테이블이지만 물리적으로 여러 데이터 파일에 분산 저장, 데이터 조회 범위를 줄여 성능 향상
 - ✓ Range Partition: 데이터 값의 범위를 기준으로 분할
 - ✓ List Partition: 특정한 값을 기준으로 분할
 - ✓ Hash Partition: 해시 함수를 적용하여 분할, DBMS가 알아서 분할 관리, 데이터 위치를 알 수 없음
 - ✓ Composite Partition: 여러 파티션 기법을 복합적으로 사용하여 분할
- ※ 파티션 인덱스(Partition Index)
- Global Index, Local Index: 여러 파티션에서 단일 인덱스 사용, 파티션 별로 각자 인덱스 사용

- Prefixed Index, Non-Prefixed Index: 파티션키와 인덱스키 동일, 파티션키와 인덱스키 구분

5절 DB 구조와 성능

1. 슈퍼타입/서브타입 데이터 모델 변환을 통한 성능 향상

- ✓ 슈퍼타입/서브타입 데이터 모델: 속성을 할당하여 배치하는 수평 분할된 형태의 모델 (공통 속성은 슈퍼타입으로 모델링하고 차이가 있는 속성은 서브타입으로 구분됨), 변환을 통해 1) 정확하게 업무를 표현할 수 있고 2) 물리적 모델링 시 선택의 폭을 넓힐 수 있음
- ✓ 변환 기준: 데이터 양, 트랜잭션 유형
- ✓ 변환 기술
 - 1:1 타입(OneToOne type): 개별로 처리하는 트랜잭션에 대해 개별 테이블 구성, 슈퍼타입과 서브타입 각각 필요한 속성과 유형에 적합한 데이터만 가지도록 분리하여 1:1 관계를 갖도록 함
 - 슈퍼/서브 타입(Plus type): 슈퍼타입과 서브타입을 공통으로 처리하는 트랜잭션에 대해 슈퍼타입과 서브타입 각각의 테이블 구성
 - All in One 타입(Single type): 일괄 처리하는 트랜잭션에 대해 단일 테이블 구성

	1:1 타입	슈퍼/서브 타입	All in One 타입
특징	개별 테이블 유지	슈퍼/서브 타입 테이블 구성	단일 테이블 구성
트랜잭션 유형	개별 처리	슈퍼/서브 타입 공통 처리	일괄 처리
확장성	좋음 (테이블 추가 용이)	보통	나쁨
조인 성능	나쁨 (조인 많이 필요)	나쁨 (조인 많이 필요)	좋음
I/O 성능	좋음	좋음	나쁨 (항상 전체 데이터 조회)
관리용이성	나쁨	나쁨	좋음

2. PK/FK 칼럼 순서 조절을 통한 성능 향상: 등호 조건이나 BETWEEN 조건이 걸리는 칼럼을 앞으로 이동 (여러 조건이 있을 경우 등호 조건이 걸리는 칼럼을 선행로 이동)
3. 인덱스 특성을 고려한 PK/FK DB 성능 향상: 물리적인 테이블에 FK 제약을 걸어 인덱스를 생성

6절 분산 DB 데이터에 따른 성능

1. 분산 DB: 분산된 DB를 하나의 가상 시스템으로 사용할 수 있도록 한 DB, 물리적 사이트는 분산되어 있으나 논리적으로 동일한 시스템, 과거에는 위치 중심이었으나 현재는 업무 필요에 따라 분산 설계

- ✓ 설계 방식
 - 상향식: 지역 스키마 작성 후 전역 스키마 작성
 - 하향식: 전역 스키마 작성 후 지역사상 스키마 작성
- ✓ 장단점
 - 1) 신뢰성과 가용성 증가 2) 빠른 응답 속도와 통신비용 절감 3) 용량 확장 용이
 - 1) 관리 및 통제 어려움 2) 데이터 무결성 관리 어려움 3) S/W 개발 비용 및 처리 비용 증가 4) 불규칙한 응답 속도

2. 분산 DB의 투명성 분위/중장병행

- ✓ 분할 투명성: 하나의 논리적 관계가 분할되어 각 단편의 사본이 여러 사이트에 저장됨
- ✓ 위치 투명성: 사용하려는 데이터 저장 장소가 명시되지 않아도 됨
- ✓ 지역사상 투명성: 지역 DBMS와 물리적 DB 사이의 사상이 보장됨
- ✓ 중복 투명성: DB 객체 중복 여부를 몰라도 됨
- ✓ 장애 투명성: 구성요소(DBMS, 컴퓨터)의 장애에 무관하게 트랜잭션의 원자성이 유지됨
- ✓ 병행 투명성: 다수의 트랜잭션을 동시 수행했을 때 결과의 일관성이 유지됨 *병렬 아님*

3. 분산 DB 적용 기법

- ✓ 테이블 위치 분산: 설계된 테이블의 위치를 분산함

- ✓ 테이블 분할 분산(Table Fragmentation): 테이블을 쪼개서 분산함
 - 1) 수평분할 2) 수직분할
- ✓ 테이블 복제 분산(Table Replication): 동일한 테이블을 다른 지역이나 서버에서 동시 생성함, 원격지 조인을 내부 조인으로 변경하여 성능 향상
 - 1) 부분복제 2) 광역복제
- ✓ 테이블 요약 분산(Table Summarization)
 - 분석요약: 사이트 별 요약정보를 본사에서 통합하여 전체 요약정보 산출
 - 통합요약: 사이트 별 정보를 본사에서 통합하여 전체 요약정보 산출

2과목 SQL 기본 및 활용

1장 SQL 기본

1절 관계형 DB 개요

1. **DB**: 데이터를 일정한 형태로 저장해 놓은 것, DBMS를 이용하여 효율적인 데이터 관리와 데이터 손상 복구 가능
 - ✓ 종류
 - 계층형 DB: 트리 형태의 자료구조에 데이터 저장, 1:N 관계 표현
 - 네트워크형 DB: 오너와 멤버 형태로 데이터 저장, M:N 관계 표현
 - 관계형 DB: 릴레이션에 데이터 저장, 집합 연산과 관계 연산 가능
2. **관계형 DB**(RDB; Relational Database): 1) 정규화를 통해 이상현상 및 중복 데이터 제거 2) 동시성 관리와 병행 제어를 통해 데이터 동시 조작 가능
 - ✓ 집합 연산
 - 합집합(Union)
 - 차집합(Difference)
 - 교집합(Intersection)
 - 곱집합(Cartesian Product): 각 릴레이션에 존재하는 모든 데이터를 조합
 - ✓ 관계 연산
 - **선택 연산**(Selection): 조건에 맞는 행(튜플) 조회
 - **투영 연산**(Projection): 조건에 맞는 칼럼(속성) 조회
 - 결합 연산(Join): 공통 속성을 사용하여 새로운 릴레이션 생성
 - 나누기 연산(Division): 공통요소를 추출하고 분모 릴레이션의 속성을 삭제한 후 중복된 행 제거
3. **SQL**(Structured Query Language): RDB에서 사용하는 언어, 데이터 조회 및 신규 데이터 입력/수정/삭제 기능 제공
 - ✓ 종류
 - **DML**(Data Manipulation Language, 데이터 조작어) *ISUD*
 - SELECT: 데이터 조회 명령어
 - INSERT, UPDATE, DELETE: 데이터 변형 명령어
 - **DDL**(Data Definition Language, 데이터 정의어): 데이터 구조 관련 명령어 *CA_D*
 - CREATE, ALTER DROP
 - **DCL**(Data Control Language, 데이터 제어어): DB 접근 권한 부여 및 회수 명령어
 - GRANT, REVOKE
 - **TCL**(Transaction Control Language, 트랜잭션 제어어): DML로 조작한 결과를 논리적인 작업단위 별로 제어
 - COMMIT, ROLLBACK
4. **테이블(Table)**: RDB의 기본 단위, 데이터를 저장하는 객체, 칼럼과 행의 2차원 구조

2절 DDL

1. 데이터 타입 (앞은 Oracle 뒤는 SQL Server)
 - ✓ CHAR(L) : 고정 길이 문자열, 할당된 변수 값의 길이가 L 이하일 때 차이는 공백으로 채워짐
 - ✓ VARCHAR2(L), VARCHAR(L) : 가변 길이 문자열, 할당되는 변수 값의 길이의 최대값이 L임, 문자열은 가능한 최대 길이로 설정
 - ✓ **NUMBER(L,D)** : 숫자형 (L은 전체 자리 수 D는 소수점 자리 수)
 - SQL Server은 NUMERIC DECIMAL FLOAT REAL 등
 - ✓ DATE, DATETIME : 날짜형, 데이터 크기 지정이 필요하지 않음
2. **CREATE TABLE** SQL>> CREATE TABLE 테이블명 (칼럼명 데이터타입 제약조건, ...);
 - ✓ 테이블 및 칼럼 명명 규칙
 - 1) 알파벳 2) 숫자 3) '_'(언더바) 4) '\$'(달러) 5) '#'(샵) 사용
 - 대소문자 구분하지 않음

- 테이블명은 단수형 권고
- ✓ 제약조건: 데이터 무결성 유지가 목적, 복제 테이블에는 기존 테이블 제약조건 중 NOT NULL만 적용
- **PRIMARY KEY**: 테이블 당 하나의 기본키만 정의 가능, 기본키 생성시 DBMS가 자동으로 인덱스를 생성함, NULL 불가
- **FOREIGN KEY**: 으로 다른 테이블의 기본키를 외래키로 지정, 참조 무결성 제약조건
SQL>> ALTER TABLE 테이블명 ADD CONSTRAINT 칼럼명 FOREIGN KEY (칼럼명) REFERENCES 테이블명(칼럼명);
- **UNIQUE KEY**: 행 데이터를 식별하기 위해 생성함, NULL 가능
- **DEFAULT**: 'DEFAULT 값'으로 기본값 설정
- **NOT NULL**
※ NULL: 아직 정의되지 않은 값 또는 현재 데이터를 입력하지 못하는 값, NULL과의 1) 수치연산은 NULL 2) 비교연산은 FALSE 출력
- **CHECK**: 입력값의 종류 및 범위 제한
- ✓ DESCRIBE 테이블명, sp_help 'dbo.테이블명': 테이블 구조 확인, 'DESCRIBE 테이블명'이 ANSI/ISO 표준

3. ALTER TABLE: 테이블의 칼럼 관련 변경 명령어

- ✓ 칼럼 추가 SQL>> ALTER TABLE 테이블명 **ADD** (칼럼명 데이터타입);
 - 마지막 칼럼으로 추가됨 (칼럼 위치 지정 불가)
 - ✓ 칼럼 삭제 SQL>> ALTER TABLE 테이블명 **DROP COLUMN** 칼럼명;
 - 삭제 후 복구 불가
 - ✓ 칼럼 설정 변경 SQL>> ALTER TABLE 테이블명 **MODIFY** (칼럼명 데이터타입 제약조건);
 - 1) NULL만 있거나 2) 행이 없는 경우에만 칼럼의 크기 축소 가능
 - NULL만 있을 때는 데이터 유형도 변경 가능
 - NULL이 없으면 NOT NULL 제약조건 추가 가능
 - 기본값 변경 작업 이후 발생하는 데이터에 대해서만 기본값이 변경됨
 - ✓ 칼럼명 변경 SQL>> ALTER TABLE 테이블명 **RENAME COLUMN** 칼럼명;
 - ANSI/ISO 표준에 명시된 기능 아님
 - ✓ 제약조건 추가 SQL>> ALTER TABLE 테이블명 **ADD CONSTRAINT** 제약조건;
 - ✓ 제약조건 제거 SQL>> ALTER TABLE 테이블명 **DROP CONSTRAINT** 제약조건;
- ### 4. RENAME TABLE
- SQL>> **RENAME** 테이블명 **TO** 테이블명; (ANSI/ISO 표준)
- ✓ 'ALTER TABLE 테이블명 RENAME TO 테이블명'으로도 가능함
- ### 5. DROP TABLE
- SQL>> **DROP TABLE** 테이블명;
- ✓ 테이블의 데이터와 구조 삭제, 복구 불가
 - ✓ CASCADE CONSTRAINT 옵션으로 관련 테이블의 참조 제약조건도 삭제하여 참조 무결성을 준수할 수 있음 (CREATE TABLE에서 ON DELETE CASCADE 옵션으로도 동일 기능 실현 가능)
- ### 6. TRUNCATE TABLE
- SQL>> **TRUNCATE TABLE** 테이블명;
- ✓ 테이블의 전체 데이터 삭제 (↔ DROP TABLE은 테이블 자체를 제거함)
 - ✓ 로그를 기록하지 않기 때문에 ROLLBACK 불가

3절 DML

1. INSERT: 데이터 입력
SQL>> **INSERT INTO** 테이블명 (칼럼명, ...) **VALUES** (필드값, ...);
SQL>> **INSERT INTO** 테이블명 **VALUES** (필드값, ...);
 2. UPDATE: 데이터 수정
SQL>> **UPDATE** 테이블명 **SET** 칼럼명=필드값;
 3. DELETE: 데이터 삭제
SQL>> **DELETE FROM** 테이블명 **WHERE** 조건절;
SQL>> **DELETE FROM** 테이블명;
- ✓ **DELETE**로 데이터를 삭제해도 테이블 용량은 초기화되지 않음 (↔ **TRUNCATE**로 삭제하면 초기화됨)
 - ✓ ↔ DROP은 객체 삭제 명령어

4. SELECT

- ✓ 칼럼 별 데이터 선택 SQL>> SELECT 칼럼명 FROM 테이블명;
 - ✓ 데이터 중복 없이 선택 SQL>> SELECT DISTINCT 칼럼명 FROM 테이블명;
 - ✓ 전체 칼럼의 데이터 선택 SQL>> SELECT * FROM 테이블명;
- ※ 앨리어스(Alias)
- SELECT 칼럼명 AS "별명" : 출력되는 칼럼명 설정
 - FROM 테이블명 별명 : 쿼리 내에서 사용할 테이블명 설정, 칼럼명이 중복될 경우 SELECT절에서 앨리어스 필수
5. 문자열의 합성 연산자: '+'(플러스), CONCAT 함수로도 2개 문자열 합성 가능, Oracle에서는 '||'(수직선 2개)도 가능
6. DUAL : Oracle의 기본 더미 테이블, 연산 수행을 위해 사용됨

4절 TCL

1. 트랜잭션: DB의 논리적 연산 단위, 하나 이상의 SQL문을 포함함
 - ✓ 특성 ACID
 - 원자성(Atomicity): 전부 실행되거나 전혀 실행되지 않음 (All or Nothing)
 - 일관성(Consistency): 트랜잭션으로 인한 DB 상태의 모순이 없음
 - 고립성(Isolation): 부분적인 실행 결과에 다른 트랜잭션이 접근할 수 없음, LOCKING으로 고립성 보장
 - 영속성(Durability): 트랜잭션의 결과는 영구적으로 저장됨
2. TCL: 데이터 무결성 보장을 목적으로 함, 1) 영구 변경 전 확인과 2) 연관 작업 동시처리 가능
 - ✓ Oracle은 1) SQL 문장을 실행하면 트랜잭션이 시작되고 2) TCL을 실행하면 트랜잭션이 종료됨
 - ✓ DDL을 실행하면 자동 커밋 (DML 이후 커밋 없이 DDL을 실행해도 자동 커밋)
 - ✓ DB를 정상적으로 종료하면 자동 커밋, 애플리케이션 등의 이상으로 DB 접속이 단절되면 자동 롤백
3. COMMIT: 데이터를 DB에 영구적으로 반영하는 명령어, 커밋 시 트랜잭션이 완료되어 LOCKING이 해제됨, SQL Server은 기본적으로 자동 커밋
 - ✓ COMMIT 전
 - 데이터 변경이 메모리 버퍼에만 영향을 받았기 때문에 복구 가능 (NOLOGGING 옵션 사용 시 버퍼 캐시의 기록을 생략하여 입력 성능이 향상됨)
 - 사용자는 SELECT절로 결과를 확인할 수 있으나 다른 사용자는 현재 결과를 볼 수 없음
 - 변경된 행에 LOCKING이 설정되어 다른 사용자가 변경할 수 없음 (LOCKING이 안 걸린 상태일 때 여러 사용자가 데이터를 변경하면 상관없음)
 - ✓ COMMIT 후
 - 변경 사항이 DB에 반영되고 이전 데이터는 복구 불가
 - 모든 사용자가 결과를 볼 수 있음
 - LOCKING이 해제되어 다른 사용자가 행을 조작할 수 있음
4. ROLLBACK: 트랜잭션 시작 이전의 상태로 되돌리는 명령어, COMMIT 이전 상태로 돌려줌, ROLLBACK 시 LOCKING이 해제됨
 - ✓ SAVEPOINT: 트랜잭션 일부만 롤백 할 수 있도록 중간상태를 저장하는 명령어, 'ROLLBACK TO 저장점명' 시 해당 저장점 상태로 돌려줌, 동일한 저장점명이 있으면 나중 저장점이 유효함
 - ✓ SQL Server에서는 'BEGIN TRAN'으로 명시해야 가능함

5절 WHERE절

1. WHERE SQL>> SELECT 칼럼명 FROM 테이블명 WHERE 조건절;
2. 연산자
 - ✓ 종류
 - 비교 연산자: =, >, >=, <, <=
 - 비교 대상 데이터 타입에 따라 자동으로 형 변환되는 경우도 있음
 - 부정 비교 연산자: 'NOT 칼럼명 비교연산자'와 동일

- 부등호: !=, ^=, <> (ISO 표준)
- SQL 연산자 입력값을 비교하여 논리값 출력
 - BETWEEN A AND B : A와 B 사이값
 - IN (리스트) : 리스트 내의 값
 - LIKE '문자열' : 문자열의 형태와 일치하는 값
 - ※ 와일드카드: 1) '%' (퍼센트)는 0개 이상의 문자 2) '_' (언더바)는 1개의 단일 문자
 - IS NULL : NULL은 등호로 판단 불가 어떤 상황에서도
 - NOT BETWEEN A AND B, NOT IN (리스트), IS NOT NULL
- 논리 연산자: AND, OR, NOT
- ✓ 우선순위: 부정 연산자 > 비교 연산자 > 논리 연산자
 - ① '()' (괄호)
 - ② NOT
 - ③ 비교 연산자 및 SQL 연산자
 - ④ AND
 - ⑤ OR
- ✓ 문자열 비교방법
 - CHAR vs CHAR : 첫 서로 다른 문자열 값으로 비교 (뒤 순서가 더 큰 값), 길이가 다를 때 공백을 추가하여 길이 맞춤 (공백 수만 다르면 같은 값)
 - CHAR vs VARCHAR : 첫 서로 다른 문자열 값으로 비교, 길이가 다르면 길이가 긴 값이 크다고 판단, VARCHAR의 공백도 문자로 판단, TRIM 함수로 VARCHAR의 공백 제거하고 판단할 수 있음
 - CHAR vs 상수 : 상수를 변수 타입으로 바꿔 비교

3. 부분 범위 처리

- ✓ ROWNUM (Oracle): SQL 처리 결과 집합의 각 행에 임시로 부여되는 번호, 조건절 내에서 행의 개수를 제한하는 목적으로 사용함
- ✓ TOP (SQL Server): 출력 행의 수 제한 함수, 'TOP (N)'로 N개 행 출력, 개수 대신 비율로도 제한 가능
 - ※ ORDER BY절이 없으면 ROWNUM과 TOP의 기능이 같음

6절 함수

1. 단일 행 함수: 1) SELECT절 2) WHERE절 3) ORDER BY절에 사용 가능, 각 행에 개별적으로 작용, 여러 인자를 입력해도 단 하나의 결과만 출력
 - ✓ 문자형 함수: 문자열 입력 시 문자열이나 숫자 반환
 - LOWER, UPPER, LENGTH
 - CONCAT : 문자열 결합
 - SUBSTR : 문자열 부분 추출
 - LTRIM, RTRIM, TRIM : 왼쪽 공백 제거, 오른쪽 공백 제거, 양쪽 공백 제거
 - ASCII : 아스키 코드값 출력
 - ✓ 숫자형 함수
 - ABS, SIGN : 절대값, 부호 (1, 0, -1 중 출력)
 - MOD : 나머지, 연산자 '%'로 대체 가능함
 - ROUND, CEIL, FLOOR : 반올림, 올림, 버림 ('함수(E,N)'으로 소수점 이후 N번째 자리까지 출력)
 - TRUNC : 숫자형 부분 추출
 - ✓ 날짜형 함수
 - SYSDATE : 현재 시각 출력 (년, 월, 일, 시, 분, 초)
 - EXTRACT : 날짜형 부분 추출 SQL>> SELECT EXTRACT(부분 FROM SYSDATE) FROM DUAL;
 - ±숫자, ±숫자/24 : 일자 연산, 시간 연산
 - NEXT_DAY : 지정된 요일 첫 날짜 출력

- ✓ 변환형 함수: 데이터 타입 변환, 명시적 형 변환 방식
 - **TO_NUMBER, TO_CHAR, TO_DATE** (Oracle): 문자열을 숫자로, 숫자나 날짜를 문자열로, 문자열을 날짜로
 - CAST, CONVERT (SQL Server)
- ✓ NULL 관련 함수
 - **NVL(칼럼,값)** : NULL 값 변환
 - **NVL2(칼럼,값,값)** : NULL이면 앞의 값 아니면 뒤의 값 출력
 - **NULLIF(값,값)** : 같으면 NULL 다르면 첫 값 출력
 - **COALESCE(값,값,...)** : NULL이 아닌 첫 값 출력
 - **ISNULL(칼럼,값)** : NULL이면 값으로 대치 아니면 칼럼 값 출력

2. 데이터 변환

- ✓ 명시적 형 변환: 변환형 함수를 이용하여 데이터 타입 변환
- ✓ 암시적 형 변환: DBMS가 자동으로 데이터 타입 변환

3. 조건문: IF-THEN-ELSE 형태

- ✓ CASE WHEN 조건절1 THEN 출력값1 ... ELSE 기본값 END : ELSE 생략 시 NULL 출력
 - ※ 'CASE WHEN NULL THEN 출력값 ELSE 기본값'은 조건이 없으므로 모든 행에서 기본값 출력 (일반적으로 'WHEN 칼럼 IS NULL'로 수정 필요)
- ✓ **DECODE** (칼럼, 기준값1, 출력값1, ..., 기본값) : Oracle 함수, 기준값n이면 출력값n 출력

7절 GROUP BY, HAVING절

- 집계 함수(Aggregate Function): 그룹별 결과 출력, 다중 행 함수 중 하나, GROUP BY절이 없으면 그룹핑 대상이 존재하지 않아 에러 발생, WHERE절에 사용 불가, 공집합에서도 연산 수행
 - ✓ ALL, DISTINCT : 전체 출력, 중복 제외 출력
 - ✓ **SUM, AVG, MAX, MIN, VARIAN, STDDEV** : NULL 제외하고 연산 (↔ 숫자 연산은 NULL 출력)
 - ✓ COUNT : 행 수 출력
 - **COUNT(*)** : NULL 포함
 - **COUNT(표현식)** : NULL 제외
- GROUP BY: 그룹핑 기준 설정, 앨리어스 사용 불가
- HAVING: GROUP BY절에 의한 집계 데이터에 출력 조건을 걸 (↔ WHERE절은 SELECT절에 조건을 걸기 때문에 제외된 데이터가 GROUP BY 대상이 아님), 일반적으로 GROUP BY 뒤에 위치함

8절 ORDER BY절

- ORDER BY: 특정 칼럼을 기준으로 정렬, 기본 정렬기준은 오름차순
 - ✓ 1) 칼럼명 2) 앨리어스 3) 칼럼의 SELECT절에서 순서로 칼럼 지정 가능, SELECT절에 없는 칼럼도 지정 가능, GROUP BY절이 있으면 GROUP BY 대상 칼럼만 지정 가능
 - ✓ Oracle은 NULL을 최대값으로 판단함 *회장님은 상사가 없음* (↔ SQL Server은 최소값으로 판단함)
- SELECT문 실행 순서
 - ✓ 테이블에서 출력 대상이 아닌 것은 제거하고 그룹핑해서 그룹핑된 값이 조건에 맞는 데이터를 계산 및 출력하고 정렬함

SELECT 칼럼명 AS "별명"

⑤ 계산 및 출력하고

FROM 테이블명

① 테이블에서

WHERE 조건식

② 출력 대상이 아닌 것은 제거하고

GROUP BY 칼럼/표현식

③ 그룹핑해서

HAVING 조건식

④ 그룹핑된 값이 조건에 맞는

데이터를

ORDER BY 칼럼/표현식

⑥ 정렬함

9절 조인

- 조인: 여러 테이블을 연결 또는 결합하여 데이터를 출력하는 것, 일반적으로 PK나 FK의 연관성에 의해 성립

2. 등가 조인: 두 테이블의 칼럼 값이 정확히 일치하는 경우, 대부분 PK와 FK 관계를 기반으로 함
SQL>> SELECT 칼럼s FROM 테이블1 A, 테이블2 B WHERE A.칼럼명=B.칼럼명;
✓ SELECT 대상 칼럼이 두 테이블 모두에 있는 경우 앨리어스를 지정해야 함 (양쪽 앨리어스 모두 무관)
3. 비등가 조인: 두 테이블의 칼럼 값이 정확하게 일치하지 않는 경우, 부등호나 BETWEEN 연산자를 통해 조인

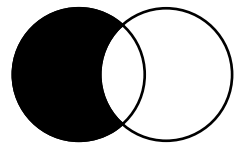
2장 SQL 활용

1절 표준조인

1. SQL에서의 연산

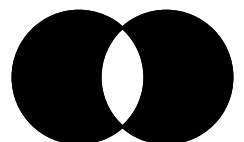
집합 연산	UNION	UNION	합집합
	INTERSECTION	INTERSECT	교집합
	DIFFERENCE	MINUS (오라클) EXCEPT (SQL Server)	차집합
	PRODUCT	CROSS JOIN	곱집합 (생길 수 있는 모든 데이터 조합)
관계 연산	SELECT	WHERE절	조건에 맞는 행 조회
	PROJECT	SELECT절	조건에 맞는 칼럼 조회
	JOIN	여러 JOIN	
	DIVIDE	없음	공통요소를 추출하고 분모 릴레이션의 속성을 삭제한 후 중복된 행 제거

2. ANSI/ISO SQL의 조인 형태: INNER JOIN, NATURAL JOIN, CROSS JOIN, OUTER JOIN
3. NATURAL JOIN: 같은 이름을 가진 칼럼 전체에 대한 등가 조인, USING 조건절이나 ON 조건절 사용 불가, 같은 데이터 유형 칼럼만 조인 가능, 앨리어스나 테이블명 사용 불가
SQL>> SELECT 칼럼s FROM 테이블1 NATURAL JOIN 테이블2;
4. INNER JOIN: 행에 동일한 값이 있는 칼럼 조인, JOIN의 디폴트 옵션, USING 조건절이나 ON 조건절 필수, CROSS JOIN이나 OUTER JOIN과 동시 사용 불가, 두 테이블에 동일 이름 칼럼이 있을 경우 SELECT절에 앨리어스 필수
SQL>> SELECT 칼럼s FROM 테이블1 A, 테이블2 B WHERE A.칼럼=B.칼럼;
SQL>> SELECT 칼럼s FROM 테이블1 A INNER JOIN 테이블2 B ON A.칼럼=B.칼럼; (ANSI/ISO 표준)
✓ USING 조건절: 같은 이름을 가진 칼럼 중 등가 조인 대상 칼럼 선택, SQL Server에서는 지원하지 않음, 조건절에 앨리어스나 테이블명 불가
SQL>> SELECT 칼럼s FROM 테이블1 A JOIN 테이블2 B USING (칼럼명);
✓ ON 조건절: 다른 이름을 가진 칼럼 간 조인 가능 (앨리어스나 테이블명 필수), 괄호는 의무사항 아님
SQL>> SELECT 칼럼s FROM 테이블1 A JOIN 테이블2 B ON (A.칼럼=B.칼럼);
5. CROSS JOIN: 가능한 모든 조합으로 조인
SQL>> SELECT 칼럼 FROM 테이블1, 테이블2; (조인 조건이 없을 때 발생 ↔ NATURAL JOIN은 명시해야 됨)
6. OUTER JOIN: 조인 조건에서 행에 동일한 값이 없는 칼럼 조인, USING 조건절이나 ON 조건절 필수
✓ LEFT OUTER JOIN: 좌측 테이블 데이터 조회 후 우측 테이블 조인 대상 데이터 조회
SQL>> SELECT 칼럼s FROM 테이블1 A, 테이블2 B A.칼럼=B.칼럼(+);
SQL>> SELECT 칼럼s FROM 테이블1 A
LEFT OUTER JOIN 테이블2 B
ON (A.칼럼=B.칼럼);
✓ RIGHT OUTER JOIN *오른쪽 결과가 더 큼*
✓ FULL OUTER JOIN: LEFT와 RIGHT OUTER JOIN 포함
SQL>> SELECT 칼럼s FROM 테이블1 A
FULL OUTER JOIN 테이블2 B
ON (A.칼럼=B.칼럼);



결과:

FT	ST
D1	D1'
D2	
	D3'

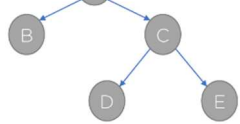


2절 집합 연산자

1. 집합 연산자: 조인 없이 여러 테이블의 관련 데이터를 조회하는 연산자

2. UNION: 합집합, 칼럼 수와 데이터 타입이 모두 동일한 테이블 간 연산만 가능
SQL>> SELECT 칼럼명 FROM 테이블명 A WHERE 조건절 UNION SELECT 테이블명 WHERE 조건절;
✓ UNION ALL: 중복된 행도 전부 출력하는 합집합, 정렬 안함 (↔ UNION은 정렬을 유발함), 집합 연산자에 속함
SQL>> SELECT 칼럼명 FROM 테이블명 A WHERE 조건절 UNION ALL SELECT 테이블명 WHERE 조건절;
3. INTERSECT: 교집합
SQL>> SELECT 칼럼명 FROM 테이블명 A WHERE 조건절 INTERSECT SELECT 테이블명 WHERE 조건절;
4. MINUS, EXCEPT: 차집합
SQL>> SELECT 칼럼명 FROM 테이블명 A WHERE 조건절 MINUS SELECT 테이블명 WHERE 조건절;

3절 계층형 질의와 셀프 조인

1. 계층형 질의(Hierarchical Query): 계층형 데이터를 조회하기 위해 사용함, Oracle에서 지원함

 - ✓ 계층형 데이터: 엔터티를 순환관계 데이터 모델로 설계할 때 발생함
 - ✓ CONNECT BY : 트리 형태의 구조로 쿼리 수행 (루트 노드부터 하위 노드의 쿼리를 실행함) *상사 이름과 사람 이름을 조인하여 상사 밑에 넣기*
 - START WITH : 시작 조건 지정
 - CONNECT BY PRIOR : 조인 조건 지정
 - LEVEL : 검색 항목의 깊이, 최상위 계층의 레벨은 1
 - CONNECT_BY_ROOT : 최상위 계층 값 표시
 - CONNECT_BY_ISLEAF : 최하위 계층 값 표시
 - SYS_CONNECT_BY_PATH : 계층 구조의 전개 경로 표시
 - CONNECT BY절의 루프 알고리즘 키워드
 - NOCYCLE : 순환구조의 발생지점까지만 전개
 - CONNECT_BY_ISCYLE : 순환구조 발생지점 표시 (부모 노드와 자식 노드가 같을 때 1 아니면 0 출력)
 - ✓ LPAD : 계층형 조회 결과를 명확히 하기 위해 사용 (LEVEL 값을 이용하여 결과 데이터 정렬)
2. SQL Server 계층형 질의: CTE(Common Table Expression)로 재귀 호출
3. 셀프 조인: 한 테이블 내에서 두 칼럼이 연관 관계가 있는 경우, 앨리어스 필수

4절 서브쿼리

1. 서브쿼리: 하나의 SQL문 안의 SQL문
2. 종류
 - ✓ 동작 방식에 따른 분류
 - 비연관 서브쿼리: 메인쿼리 칼럼을 가지고 있지 않는 서브쿼리, 메인쿼리에 값을 제공하기 위한 목적으로 주로 사용함
 - Access Subquery: 제공자 역할
 - Filter Subquery: 확인자 역할
 - Early Filter Subquery: 데이터 필터링 역할
 - 연관 서브쿼리(Associative Subquery): 메인쿼리의 결과를 조건이 맞는지 확인하기 위한 목적으로 주로 사용함
 - ✓ 반환 데이터 형태에 따른 분류
 - 단일 행 서브쿼리: 실행 결과가 1건 이하인 서브쿼리, 단일 행 비교 연산자와 함께 사용
 - 다중 행 서브쿼리: 실행 결과가 여러 건인 서브쿼리, 다중 행 비교 연산자와 함께 사용
 - ※ 다중 행 비교 연산자
 - IN : 서브쿼리의 결과 중 하나의 값이라도 동일하다는 조건
 - ANY : 서브쿼리의 결과 중 하나의 값이라도 만족한다는 조건
 - ALL : 서브쿼리의 모든 결과값을 만족한다는 조건
 - EXISTS : 서브쿼리의 결과를 만족하는 값이 존재하는지 여부를 확인하는 조건, 'WHERE EXISTS

(SELECT ~)' (항상 연관 서브쿼리로 사용)

- 다중 칼럼 서브쿼리: 실행 결과로 여러 칼럼 반환, 주로 메인쿼리의 조건과 비교하기 위해 사용 (비교하고자 하는 칼럼의 개수와 위치가 동일해야 함)

3. 스칼라 서브쿼리: 값 하나를 반환하는 서브쿼리, SELECT절에 사용하는 서브쿼리

4. 뷰: 가상의 테이블, FROM절에 사용하는 뷰는 인라인 뷰(Inline View)라고 함

✓ 장점

- 독립성: 테이블 구조 변경 자동 반영
- 편리성: 쿼리를 단순하게 작성할 수 있음, 자주 사용하는 SQL문의 형태를 뷰로 생성하여 사용할 수 있음
- 보안성: 뷰를 생성할 때 칼럼을 제외할 수 있음

5. WITH: 서브쿼리를 이용하여 뷰로 사용할 수 있는 구문

SQL>> WITH 뷰명 AS (SELECT ~)

5절 그룹 함수

1. ANSI/ISO 표준 데이터 분석 함수: 집계 함수, 그룹 함수, 윈도우 함수

2. 그룹 함수(Group Function): 합계 계산 함수, NULL을 빼고 집계함 (~ 집계 함수), 결과값 없는 행은 출력 안함

- ✓ ROLLUP : GROUP BY로 묶인 칼럼의 소계 계산, 계층 구조로 GROUP BY의 칼럼 순서가 바뀌면 결과 값 바뀜
- ✓ CUBE : 조합 가능한 모든 값에 대해 다차원 집계
- ✓ GROUPING SETS : 특정 항목에 대한 소계 계산, GROUP BY의 칼럼 순서와 무관하게 개별적으로 처리함

표현식	출력값
GROUP BY ROLLUP (E1,E2)	E1과 E2별 소계 / E1별 소계 / 총합계
GROUP BY CUBE (E1,E2)	E1과 E2별 소계 / E1별 소계 / E2별 소계 / 총합계
GROUP BY GROUPING SETS (E1,E2)	E1별 소계 / E2별 소계

※ 'GROUP BY CUBE (E1,E2)'와 'GROUP BY GROUPING SETS (E1,E2,(E1,E2),())'는 동일한 결과 출력

3. GROUPING : 그룹 함수에서 생성되는 합계를 구분해주는 함수, 소계나 합계가 계산되면 1 아니면 0 반환

6절 윈도우 함수

1. 윈도우 함수(Window Function): 여러 행 간의 관계 정의 함수, 중첩 불가

✓ 순위 함수

- RANK : 중복 순위 포함
- DENSE_RANK : 중복 순위 무시 (중간 순위를 비우지 않음)
- ROW_NUMBER : 단순히 행 번호 표시, 값에 무관하게 고유한 순위 부여

✓ 일반집계 함수: SUM, MAX, MIN, AVG, COUNT

✓ 행 순서 함수

- FIRST_VALUE, LAST_VALUE : 첫 값, 끝 값
- LAG, LEAD : 이전 행, 이후 행 (Oracle) 랑킷

※ 'LEAD(E,A)'는 E에서 A번째 행의 값을 호출하는 형태로도 쓰임 (A의 기본값은 1)

✓ 비율 관련 함수

- PERCENT_RANK() : 백분율 순서
- CUME_DIST() : 현재 행 이하 값을 포함한 누적 백분율
- NTILE(A) : 전체 데이터 A등분
- RATIO_TO_REPORT : 총합계에 대한 값의 백분율

2. 윈도우 함수 문법

SQL>> SELECT 윈도우함수(A) OVER (PARTITION BY 칼럼 ORDER BY 칼럼 윈도우절) FROM 테이블명;

✓ PARTITION BY : 그룹핑 기준

✓ ORDER BY : 순위 지정 기준

✓ 윈도우절 : 함수의 대상이 되는 행 범위 지정

- BETWEEN A AND B : 구간 지정

➤ N PRECEDING, N FOLLOWING : N번째 앞 행, N번째 뒤 행

- UNBOUNDED PRECEDING, UNBOUNDED FOLLOWING : 첫 행, 끝 행
- CURRENT ROW : 현재 행
- ROWS, RANGE : 행 지정, 값의 범위 지정

7절 DCL

1. DCL: 유저를 생성하거나 권한을 제어하는 명령어, 보안을 위해 필요함
 - ✓ GRANT: 권한 부여
SQL>> GRANT 권한 ON 오브젝트 TO 유저명;
 - ✓ REVOKE: 권한 제거
SQL>> REVOKE 권한 ON 오브젝트 TO 유저명;
2. 권한(Privileges)
 - ✓ SELECT, INSERT, UPDATE, DELETE, ALTER, ALL : DML 관련 권한
 - ✓ REFERENCES : 지정된 테이블을 참조하는 제약조건을 생성하는 권한
 - ✓ INDEX : 지정된 테이블에서 인덱스를 생성하는 권한
3. Oracle의 유저
 - ✓ SCOTT: 테스트용 샘플 유저
 - ✓ SYS: DBA 권한이 부여된 최상위 유저
 - ✓ SYSTEM: DB의 모든 시스템 권한이 부여된 DBA
4. ROLE: 권한의 집합, 권한을 일일이 부여하지 않고 ROLE로 편리하게 여러 권한을 부여할 수 있음
 - ✓ Oracle의 ROLE

ROLE	권한	
CONNECT	CREATE SESSION	
RESOURCE	CREATE CLUSTER	CREATE TRIGGER
	CREATE PROCEDURE	CREATE OPERATOR
	CREATE TYPE	CREATE TABLE
	CREATE SEQUENCE	CREATE INDEXTYPE

8절 절차형 SQL

1. 절차형 SQL: 일반적인 개발언어처럼 절차지향적인 프로그램을 작성할 수 있도록 제공하는 기능
 - ✓ SQL문의 연속적인 실행 및 조건에 따른 분기처리를 이용하여 특정 기능을 수행하는 저장 모듈 생성 가능
 - ✓ PL/SQL (Oracle)
 - 블록 구조: 블록 내에 1) DML 2) 쿼리 3) IF나 LOOP 등을 사용할 수 있음
 - Declare(선언부): 블록에서 사용할 변수나 인수에 대한 정의
 - Begin(실행부): 처리할 SQL문 정의
 - Exception(예외 처리부): 블록에서 발생한 예외 처리 로직 정의, 유일한 선택 항목
 - ✓ T-SQL (SQL Server)
2. 프로시저(Procedure)
3. 사용자 정의 함수: 절차형 SQL을 로직과 함께 DB 내에 저장해 놓은 명령문 집합, RETURN을 통해 반드시 하나의 값 반환 (↔ 프로시저)
4. 트리거(Trieger): DML문이 수행되었을 때 자동으로 동작하는 프로그램 (↔ 프로시저는 EXECUTE로 실행함), DCL와 TCL 실행 불가 (↔ 프로시저는 사용 가능함)

3장 SQL 최적화 기본 원리

1절 옵티마이저와 실행계획

1. 옵티마이저: SQL문에 대한 최적의 실행방법을 결정하여 실행 계획 도출, SQL문에 대한 파싱 후 실행됨, *내비게이션*
 - ※ SQL문 실행 순서
 - ① 파싱(Parsing): SQL 문법 검사 및 구문 분석 작업
 - ② 실행(Execution): 옵티마이저의 실행 계획에 따라

③ 인출(Fetch): 데이터를 읽어 전송

✓ 옵티마이저 엔진

- 질의 변환기(Query Transformer): 작성된 SQL문을 처리하기 용이한 형태로 변환하는 모듈
- 비용 예측기(Estimator): 생성된 계획의 비용을 예측하는 모듈
- 대안계획 생성기(Plan Generator): 동일한 결과를 생성하는 다양한 대안 계획을 생성하는 모듈, 1) 연산 적용 순서 2) 연산 방법 3) 조인 순서의 변경을 통해 대안 계획 생성

✓ 종류

- 규칙기반 옵티마이저: 우선순위 규칙에 따라 실행계획 생성, 인덱스가 있으면 반드시 인덱스 사용
- 비용기반 옵티마이저: 처리 비용이 가장 적은 실행계획 선택, 데이터 디셔너리(Data Dictionary)의 통계정보나 DBMS의 차이로 같은 쿼리도 다른 실행계획이 생성될 수 있음, 실행계획의 예측 및 제어가 어려움

2. SQL 처리 흐름도: SQL문의 처리절차를 시각적으로 표현한 도표

3. 실행계획: 1) 객체 2) 조인 방법 및 순서 3) 액세스 패턴 등의 정보 출력

✓ DESC PLAN_TABLE; : 실행 계획 확인

✓ 해독 순서: ←로 찾다가 2줄 이상의 동일 레벨을 만나면 ↓로 해독

2절 인덱스 기본

1. 인덱스: 검색 조건에 부합하는 데이터를 효과적으로 검색할 수 있도록 돕는 기능, 인덱스키로 정렬되어 있어 조회 속도가 빠름, DML 작업 효율은 저하함

✓ 트리기반 인덱스: DBMS에서 사용하는 가장 일반적인 인덱스, 1) 루트 블록(Root Block) 2) 브랜치 블록(Branch Block) 3) 리프 블록(Leaf Block)으로 구성됨

- 포인터(Pointer): 루트 블록과 브랜치 블록의 키 값, 하위 블록 키 값의 범위 정보
- 리프 블록은 1) 인덱스키 2) ROWID로 구성됨, Doubly Linked List 형태라서 양방향 탐색 가능

※ ROWID: Oracle에서 데이터를 구분할 수 있는 유일한 값, 데이터를 입력하면 자동으로 생성됨, 데이터가 어떤 데이터 파일의 어느 블록에 속해 있는지 알려줌

- 오브젝트 번호: 해당 데이터가 속하는 오브젝트 번호, 오브젝트 별로 유일한 값을 가짐
- 상대 파일 번호: 테이블스페이스 내 데이터 파일의 순번
- 블록 번호: 데이터 파일 내 데이터가 속해 있는 블록의 순번
- 데이터 번호: 데이터 블록에 데이터가 저장되어 있는 순번

✓ 클러스터형 인덱스 (SQL Server): 인덱스의 리프 페이지가 데이터를 포함함, 리프 페이지의 모든 로우가 인덱스키 칼럼 순서대로 물리적으로 정렬되어 있음

✓ CREATE INDEX 인덱스명 테이블명 ON 테이블명 (칼럼명, ...) : 인덱스 생성

※ 인덱스키가 변환되면 사용 불가 ex) NVL(인덱스키,값), TO_타입(인덱스키), 인덱스키||값

2. 인덱스 스캔 효율화: 랜덤 액세스 최소화 (인덱스 스캔 후 추가 정보를 가져오기 위한 랜덤 액세스는 DBMS 성능 부하를 유발함)

※ 인덱스 칼럼의 순서는 랜덤 액세스와 무관함

3. 스캔 방법

- ✓ 전체 테이블 스캔(Full Table Scan): 테이블의 모든 데이터를 읽으며 데이터 추출, 읽은 블록의 재사용성을 낮다고 판단하여 메모리 버퍼에서 제거함, 1) SQL문에 조건이 없거나 2) SQL문 조건 관련 인덱스가 없거나 3) 전체 테이블 스캔을 하도록 강제로 힌트를 지정하거나 4) 옵티마이저가 유리하다고 판단하는 경우 수행, 많은 데이터를 조회할 때 유리함
- ✓ 인덱스 스캔(Index Scan): 인덱스를 구성하는 칼럼의 값을 기반으로 데이터 추출, 인덱스를 읽어 ROWID를 찾고 해당 데이터를 찾기 위해 테이블을 읽음, 일반적으로 인덱스 칼럼 순서로 정렬되어 출력됨, 적은 데이터를 조회할 때 유리함, 1) 랜덤 액세스에 의한 부하가 발생할 수 있고 2) 중복 스캔 비효율이 발생함
- 인덱스 범위 스캔(Index Range Scan): 특정 범위에 인덱스 스캔 적용
 - 인덱스 역순 범위 스캔: 리프 블록의 Doubly Linked List 저장 방식을 활용하여 인덱스를 역순으로 스캔, 결과 집합이 내림차순으로 정렬됨

- 인덱스 유일 스캔(Index Unique Scan): 인덱스키가 중복되지 않을 때 단 한 건의 데이터 추출, 등호 조건으로 조회함, 검색 속도가 가장 빠름
- 인덱스 전체 스캔(Index Full Scan): 리프 블록을 모두 읽으며 데이터 추출
 - 인덱스 고속 전체 스캔: 물리적으로 저장된 순서대로 인덱스 리프 블록 스캔
 - 인덱스 스킵 스캔: 인덱스 선두 칼럼이 조건절에 없어도 활용함, 상위 블록에서 읽은 칼럼 값 정보를 이용해 조건에 맞는 데이터를 포함할 가능성이 있는 리프 블록만 접근
- 4. IOT(Index-Organized Table): 인덱스키가 붙은 칼럼으로 구성된 테이블, 인덱스가 원래 테이블을 참조하지 않음, 클러스터형 인덱스와 유사함

3절 조인 수행 원리

1. 조인 순서: 항상 두 테이블을 조인함
 - ✓ 선행 테이블(First Table, Outer Table, Driving Table, Build Input)
 - ✓ 후행 테이블(Second Table, Inner Table, Driven Table, Probe Input): 선행 테이블로부터 입력값을 받아 처리함, 후행 테이블에 걸리는 조인 조건이 성능에 큰 영향을 미침
2. 조인 방식: NL 조인 > 소트 머지 조인 > 해시 조인 순서로 발전됨
 - ✓ NL 조인(Nested Loop Join): 선행 테이블의 데이터 하나씩 순차적으로 조인 (중첩 반복문과 유사함), 선행 테이블 처리 범위가 성능을 결정함 (~ 해시 조인, ↔ 소트 머지 조인은 순서에 무관함), 랜덤 액세스 위주이므로 대용량 데이터 처리 시 불리 *유니크 인덱스를 이용하여 소량 테이블 조인할 때 유리함*
 - 절차
 - ① 선행 테이블에서 조건을 만족하는 행을 찾음
 - ② 후행 테이블에 선행 테이블의 조인키가 존재하는지 확인함
 - ③ 후행 테이블 인덱스에 선행 테이블의 조인키가 존재하는지 확인함
 - ④ 인덱스에서 추출한 ROWID로 후행 테이블을 액세스함
 - 조인 결과를 하나씩 바로 출력하여 OLTP 환경에 적합함
 - ✓ 소트 머지 조인(Sort Merge Join): 두 테이블을 개별적으로 스캔한 후 조인 (↔ NL 조인은 선행 테이블을 랜덤 액세스 방식으로 조회하며 조인), 대용량 데이터 처리 시 디스크에서 정렬이 진행되므로 성능상 불리, 인덱스 유무가 성능에 큰 영향을 주지 않음 (↔ NL 조인은 인덱스 구성에 크게 영향을 받음)
 - ✓ 해시 조인(Hash Join): 조인 칼럼을 기준으로 동일한 해시 값을 갖는 데이터의 실제 값을 비교하며 조인, 두 테이블의 데이터 차이가 클 때 유리, 1) NL 조인의 랜덤 액세스와 2) 소트 머지 조인의 정렬 작업 부담 해결, 등가 조인에서만 사용할 수 있음, 해시 메모리에서 해시 테이블을 생성하므로 선행 테이블이 작을 때 유리 *테이블이 커서 소트 부하가 심할 때 유리함*
 - OLAP 환경에 적합함