데이터 모델링 이해

■ 데이터 모델링 단계

■ 개념적 모델링
- 전사적 관점
- 추상화 수준이 가장 높음
- 업무 측면에서 모델링

■ 논리적 모델링
- 정규화를 통해 재사용성을 높임

■ 물리적 모델링
- 테이블, 인덱스 등 생성
- 성능, 보안, 가용성 등을 고려하여 데이터베이스 구축

■ 3층 스키마 구조

■ 외부 스키마
- 사용자 관점 (집주인)
- 관련 데이터베이스의 뷰
- 응용 프로그램이 접근하는 데이터베이스 정의

■ 개념 스키마
- 설계자 관점 = 조직 전체 관점(관리인)
- 통합 데이터베이스 구조

■ 내부 스키마
- 개발자 관점 (건설 업체)
- 물리적 저장 구조

■ 엔터티 특징

- 유일한 식별자
- 2개 이상의 인스턴스
- 2개 이상의 속성
- 다른 엔터티와 최소 한 개 이상 관계
- 업무에서 관리되어야 하는 집합

■ 엔터티 종류

■ 유형 / 무형 (= 물리적 형태 존재 여부)

- 유형 엔터티 : 물리적 형태가 있음
- 개념 엔터티 : 물리적 형태가 없음
- 사건 엔터티 : 비즈니스 프로세스에서 생성

■ 발생 시점

- 기본(key) 엔터티 : 독립적 생성
- 중심(main) 엔터티 : 기본 엔터티로부터 발생되고 행위 엔터티를 생성
- 행위 엔터티 : 2개 이상의 엔터티로부터 발생 – 지속적으로 정보 추가됨 (데이터 양 多)

■ 속성 특징

- 업무에서 관리되는 정보
- 하나의 값만 가짐
- 주식별자에 함수적 종속

■ 속성 종류

■ 분해 여부 : 단일 속성 / 복합 속성 / 다중값 속성(엔터티로 분해)

■ 특성 : 기본 속성 / 설계 속성 / 파생 속성

■ 식별자 종류

■ 대표성 여부 (유일성&최소성)

- 주 식별자 : 엔터티 내에서 구분자 역할을 하며 타 엔터티와 참조 관계를 연결할 수 있음
- 보조 식별자 : 엔터티 내에서 구분자 역할을 하지만 대표성을 가지지 못해 참조 관계 연결 불가능

■ 생성 여부

- 내부 식별자 : 엔터티 내부에서 스스로 생성됨 ex) 부서코드, 주문번호, 종목코드
- 외부 식별자 : 타 엔터티와의 관계로 타 엔터티로부터 받아옴 ex) 계좌 엔터티에 회원아이디

■ 속성의 수

- 단일 식별자 : 하나의 속성으로 구성됨
- 복합 식별자 : 둘 이상의 속성으로 구성됨

■ 대체 여부

- 본질 식별자 : 업무에 의해 생성됨
- 인조 식별자 : 원조 식별자가 복잡한 구성을 가지고 있을 때 인위적으로 생성됨

■ 정규화

- 모델의 유연성 향상시킴
- 제1정규화 : 속성의 원자성 확보
- 제2정규화 : 부분 함수 종속성 제거
- 제3정규화 : 이행 함수 종속성
- BCNF : 후보키가 기본키를 종속시키면 분해

■ 반정규화

- 데이터 중복 허용
- 조인을 줄이는 방법
- 조회 속도 향상

■ 테이블 반정규화

병합	1:1 관계 테이블 병합	1:1 관계를 통합하여 성능 향상
	1:M 관계 테이블 병합	1:M 관계를 통합하여 성능 향상
	슈퍼/서브타입 테이블 병합	슈퍼/서브 관계를 통합하여 성능 향상
분할	수직 분할	컬럼 단위의 테이블을 1:1로 분리하여 성능 향상
	수평 분할	행 단위로 집중 발생되는 트랜잭션을 분석하여 행 단위로 테이블 분리
추가	중복 테이블 추가	다른 업무이거나 서버가 다른 경우 동일한 테이블 구조를 중복하여 원격 조인을 제거하고 성능 향상
	통계 테이블 추가	집계 함수를 미리 수행하여 계산해 둠으로써 조회 성능 향상
	이력 테이블 추가	이력 테이블 중에서 마스터 테이블에 존재하는 레코드를 중복하여 이력 테이블에 추가하는 방법

■ 컬럼 반정규화

중복 컬럼 추가	조인을 감소시키기 위해 중복된 컬럼 추가
파생 컬럼 추가	계산량을 감소시키기 위해 미리 값을 계산하여 컬럼 추가
이력 테이블 컬럼 추가	대량의 이력 데이터를 처리할 때 기능성 컬럼 추가 (최근값 여부, 시작일자)
기본키에 의한 컬럼 추가	복합 의미를 갖는 단일 기본키에서 특정 값을 별도로 조회하는 경우 이미 기본키 안에 데이터가 존재하지만 성능 향상을 위해 일반 속성으로 포함하는 방법
응용 시스템 오작동을 위한 컬럼 추가	업무적으로 의미가 없지만 사용자가 잘못 처리하여 원래 값으로 복구하기 원하는 경우 이전 데이터를 임시적으로 중복하여 보관하는 기법

■ **분산 데이터베이스 투명성**

: 분할 / 위치 / 지역 사상 / 중복 / 장애 / 병행

■ **분산 데이터베이스 장단점**

■ 장점

- 신뢰성과 가용성
- 빠른 응답
- 용량 확장 쉬움
- 지역 자치성
- 통신 비용 절감

■ 단점

- 관리, 통제 어려움
- 보안 관리 어려움
- 무결성 관리 어려움
- 설계 복잡
- 개발, 처리 비용 증대
- 불규칙한 응답 속도

SQL 기본 및 활용

■ NULL 관련 함수

- NVL(col1,col2): col1 값이 null이면 col2 값을 반환 (ISNULL 함수)
- NVL2(col1,col2,col3) : col1 값이 null이 아니면 col2값을, null이면 col3 값을 반환
- NULLIF(col1,col2) : col1 값이 col2 값과 같으면 null을, 같지 않으면 col1 값을 반환
- COALESCE(col1,col2,col3,…) : null이 아닌 최초의 인자 값 반환

■ SELECT문 실행 순서

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

■ 문자열 함수

- SUBSTR(str,m,n) : str에서 m번째 위치부터 n개 반환
- CONCAT(str1,str2) : str1과 str2 결합 [(oracle) || , (MS-SQL) +]
- TRIM(str) : str의 공백 제거

■ 숫자형 함수

- MOD(num1,num2) : num1을 num2로 나눈 나머지 반환 (%)
- CEIL(num) : num보다 크거나 같은 최소 정수 반환
- FLOOR(num) : num보다 작거나 같은 최대 정수 반환
- ROUND(num,m) : num의 소수점 m자리에서 반올림
- TRUNC(num,m) : num의 소수점 m자리에서 버림

■ DECODE문 구조

DECODE(col,val,result1,result2)

■ CASE문 구조

CASE [expression] WHEN condition THEN result1 ELSE result2 END

■ ROWNUM

- 조회되는 행 수를 제한할 때 사용
- 데이터를 출력할 때 부여되는 순번

ex) ROWNUM = 1 (O)　　ROWNUM 〉 0 (O)　　ROWNUM 〈= 3 (O)　ROWNUM = 2 (X)

■ DROP / TRUNCATE / DELETE 비교

DROP	- DDL - Rollback 불가능 = Auto Commit - 테이블이 사용한 storage 모두 release - 테이블 정의 완전히 삭제 - 로그 X
TRUNCATE	- DDL - Rollback 불가능 = Auto Commit - 테이블이 사용한 storage 중 최초 테이블 생성 시 할당된 storage만 남기고 release - 테이블을 최초 생성된 초기 상태로 초기화 - 로그 X - 데이터를 빠르게 삭제함
DELETE	- DML - Commit 이전 Rollback 가능 = 사용자 Commit - 사용했던 storage는 release 되지 않음 - 데이터만 삭제 - 로그 O

■ 계층형 조회 (CONNECT BY)

- START WITH : 시작 조건 (해당 데이터 출력)
- CONNECT BY PRIOR : 조인 조건
- PRIOR 위치가 중요

✓ LEVEL 키워드를 사용하면 Root가 1, 다음 노드는 2

■ 서브쿼리

- SELECT문에 사용 : 스칼라 서브쿼리
- FROM구에 사용 : 인라인 뷰
- WHERE구에 사용 : 서브쿼리
- 연관 서브쿼리 : 서브쿼리 내에서 메인 쿼리 내의 컬럼 사용

■ 그룹 함수

■ ROLLUP

- 계층 구조이기 때문에 인수의 순서가 바뀌면 결과도 달라짐
- 계층 간 정렬 가능

ex) GROUP BY ROLLUP(DEPTNO) : 부서별 합계 & 전체 합계

ex) GROUP BY ROLLUP(DEPTNO, JOB) : 부서별 합계 & 부서-직업별 합계 & 전체 합계

ex) GROUP BY ROLLUP(DEPTNO) = GROUP BY GROUPING SETS(DEPTNO,())

■ GROUPING SETS

- GROUP BY에 나오는 컬럼의 순서와 관계없이 개별적으로 모두 처리함

ex) GROUP BY GROUPING SETS(DEPTNO, JOB) : 부서별 합계, 직업별 합계

■ CUBE

- 결합 가능한 모든 집계 계산

ex) GROUP BY CUBE(DEPTNO, JOB) : 부서별 합계 & 직업별 합계 & 부서-직업별 합계 & 전체 합계

= GROUP BY GROUPING SETS(DEPTNO, JOB, (DEPTNO,JOB), ())

■ 윈도우 함수

- 행과 행 간의 관계 정의
- 순위, 합계, 평균, 행 위치 등 조작
- Group by 구문과 Window function은 병행하여 사용할 수 없음

■ 순위 함수

- RANK() : 동일 순위에 동일 값 부여
- DENSE_RANK() : 동일 순위를 하나의 건수로 계산
- ROW_NUMBER() : 동일 순위에 고유의 순위 부여

■ 행 순서 관련 함수

- FIRST_VALUE / LAST_VALUE : 가장 처음에/나중에 나오는 값 (MIN/MAX)
- LAG : 이전 행을 가지고 온다
- LEAD : 특정 위치의 행을 가지고 온다 (default = 1 = 다음 행)

■ 비율 관련 함수

- CUME_DIST : 누적 백분율 조회
- PERCENT_RANK : 행 순서별 백분율 조회
- NTILE(n) : 파티션 별로 전체 건수를 n등분한 결과 조회
- RATIO_TO_REPORT : 파티션 내에 전체 SUM에 대한 행 별 컬럼값의 백분율을 소수점까지 조회

■ 테이블 파티션

- 대용량의 테이블을 여러 개의 데이터 파일에 분리해서 저장
- 입력, 수정, 삭제, 조회 성능 향상
- Range Partition : 값의 범위를 기준으로 분할
- List Partition : 특정 값을 기준으로 분할
- Hash Partition : 해시 함수를 사용해서 분할
- Composite Partition : 여러 개의 파티션 기법을 조합해서 분할

■ 옵티마이저 조인

■ Nested Loop Join

- 하나의 테이블에서 데이터를 먼저 찾고 그 다음 테이블을 조인하는 방식
- 선행 테이블의 크기가 작은 것을 먼저 찾음
- RANDOM ACCESS 발생 -〉 성능 지연 발생
- 선행 테이블의 조건을 만족하는 건수만큼 반복 수행
- 조인 컬럼에 인덱스가 존재해야함

■ Sort Merge Join

- 두 개의 테이블을 메모리 공간에 로딩하고 SORT 수행
- 정렬이 완료되면 두 개의 테이블 병합
- 데이터 양이 많아지면 성능 저하
- 기본키와 외래키 관계에서 외래키에 인덱스가 없을 때

■ Hash Join

- 두 개의 테이블 중에서 작은 테이블을 메모리에 로딩
- 두 개의 테이블의 조인 키를 사용해서 해시 테이블 생성
- CPU 연산을 많이 함
- EQUI 조인만 사용 가능한 방법
- 조인 컬럼의 인덱스가 존재하지 않아도 사용 가능
- 정렬 작업이 없어 대량 배치작업에 유리함

■ Hash Join 절차

1) 선행 테이블에서 조건을 만족하는 레코드 필터링
2) 선행 테이블의 조인 키를 기준으로 해시 함수를 적용하여 해시 테이블 생성
3) 1,2번 작업을 선행 테이블에서 조건을 만족하는 모든 행을 수행
4) 후행 테이블에서 주어진 조건을 만족하는 행 필터링
5) 필터링한 행을 대상으로 후행 테이블의 조인키를 기준으로 해시 함수를 적용하여
   선행 테이블에서의 해시 함수 반환값과 같은 값을 반환하는 행을 찾음

■ 프로시저와 트리거

프로시저	트리거
CREATE PROCEDURE	CREATE TRIGGER
생성하면 소스코드와 실행코드가 생성됨	생성하면 소스코드와 실행코드가 생성됨
EXECUTE 명령어로 실행	생성 후 자동 실행
COMMIT, ROLLBACK 실행 가능	COMMIT, ROLLBACK 실행 불가능

✓ CHAR(10)으로 컬럼을 생성하고 8개의 문자를 입력하면 나머지 2개는 공백으로 입력됨
VARCHAR는 가변길이 문자열 타입으로 입력한 크기만큼 할당됨 (oracle은 varchar2)

✓ count(*) : null을 포함한 행수 계산 count(col) : null을 제외한 행 수 계산

✓ WHERE 컬럼명 LIKE '%@_%' ESCAPE '@'

✓ FULL OUTR JOIN = LEFT OUTER JOIN UNION RIGHT OUTER JOIN

✓ SELF JOIN : 하나의 테이블에서 두 개의 컬럼이 연관 관계를 가지고 있는 경우 사용

✓ NATURAL JOIN : 두 테이블 간에 동일한 컬럼 이름을 가진 것 모두 출력 (on절 생략 가능)
동일한 컬럼이 두 개 이상일 경우 JOIN~USING 문장 사용
Alias 사용 불가능
두 테이블에 같은 이름이 있기 때문에 SELECT 테이블명.컬럼명 불가능

✓ 서브쿼리는 메인쿼리 컬럼 사용 가능
메인쿼리는 서브쿼리 컬럼 사용 불가능

✓ SQL server는 null 값을 가장 작은 값으로 취급하고 ORACLE은 가장 큰 값으로 취급함

✓ ORACLE은 DDL이 수행되면 묵시적으로 자동 COMMIT 수행됨.
✓ SQL Server는 DDL 문장도 같이 Rollback 수행됨.

✓ ORACLE에서 Column Header의 Alias명이 없는 컬럼명은 대문자로 바뀜. SQL Server는 그대로.

✓ 조회는 일반적으로 인덱스가 있는 것이 유리함

✓ DML 작업은 테이블과 인덱스를 함께 변경하므로 느려지는 단점이 있음

✓ 인덱스 트리 구조 : Root Block, Branch Block, Leaf Block

✓ Row Chaining : 하나의 행을 여러 블록에 걸쳐서 저장 => 조회 성능 감소

✓ 옵티마이저 : SQL 실행 계획 수립

✓ 규칙 기반 옵티마이저는 적절한 인덱스가 존재하면 항상 인덱스를 사용함
✓ 규칙 기반 옵티마이저에서 제일 낮은 우선순위는 전체 테이블 스캔

✓ 비용 기반 옵티마이저는 비용 계산을 위해 다양한 통계정보 사용

✓ PL / SQL 특징
- 변수와 상수 등을 사용하여 일반 SQL 문장을 실행할 때 WHERE절의 조건으로 대입 가능
- Procedure, User Defined Function, Trigger 객체를 PL/SQL로 작성 가능
- Procedure 내부에 작성된 절차적 코드는 PL/SQL 엔진이 처리
- 일반적인 SQL 문장은 SQL 실행기가 처리
- PL/SQL문의 기본 구조로 DECLARE, BEGIN~END 문은 필수, EXCEPTION 문은 선택
- Block 구조로 되어 있어 각 기능별로 모듈화 가능
- 응용 프로그램의 성능을 향상시킴
- 여러 SQL 문장을 Block으로 묶고 한 번에 Block 전부를 서버로 보내기 때문에 통신량을 줄일 수 있음

✓ 데이터의 무결성을 보장하기 위한 방법 : 애플리케이션, Trigger, 제약조건

* Lock은 병행성(동시성) 제어 기법