데이터 입출력 구현

1. 논리 데이터 저장소 확인

(1) 논리 데이터 모델 검증

1) 논리 데이터 모델링

▼ 개념

- 데이터베이스 설계 프로세스의 기초 설계 단계, 비즈니스 정보의 구조와 규칙을 명확하게 표현
- 개념 모델로부터 업무 영역의 업무 데이터 및 규칙을 구체적으로 표현한 모델
- 개념 모델: 주제 영역과 데이터 집합 간의 관계를 정의. 엔티티와 관계 위주의 모델.

▼ 특성

- 정규화 - 모든 데이터를 정규화하여 모델링
- 포용성 - 모든 엔티티 타입, 속성, 관계, 프로세스 등을 포함
- 완전성 - 모든 규칙과 관계를 완전하고 정확히 표현
- 독립성 - 성능, 제약사항에 독립적인 모델. 특정 DBMS로부터 독립적인 성질

▼ 속성

- 개체 - 관리할 대상이 되는 **실체**
- 속성 - 관리할 정보의 구체적 **항목**
- 관계 - 개체 간의 대응 **관계**

▼ 개체-관계(E-R) 모델

- 현실 세계에 존재하는 데이터와 그들 간의 관계를 사람이 이해할 수 있는 형태로 명확히 표현하기 위해 사용하는 모델
- 요구사항으로부터 얻어낸 정보들을 **개체, 속성, 관계**로 기술한 모델
- 기호 - 개체(사각형), 속성(원), 관계(마름모)

▼ 정규화

▼ 개념

- 관계형 데이터베이스 설계에서 중복을 최소화하여 데이터를 구조화하는 프로세스

▼ 이상 현상 (Anomaly)

- 데이터 중복성으로 인해 릴레이션(테이블)을 조작할 때 발생하는 비합리적 현상
- 삽입 이상 - 정보 저장 시 해당 정보의 **불필요한 세부 정보를 입력**해야 하는 경우

EX) 학생 등록할 시 지도 교수가 정해지지 않으면 삽입 불가능

- 삭제 이상 - 정보 삭제 시 **원치 않는 다른 정보가 같이 삭제** 되는 경우

EX) 교수가 퇴사할 경우 학생 정보도 같이 삭제

- 갱신 이상 - 중복 데이터 중에서 **특정 부분만 수정되어** 중복된 값이 모순을 일으키는 경우

EX) 학생이 지도교수 변경 시 해당 교수의 정보가 없어진다

▼ 단계

1) 1차 정규화(1NF)

원자 값으로 구성

EX) 이메일 주소가 속성에 2개 이상 가지고 있는 경우 원자값이 아니므로 속성 1개만 가지도록 분리해서 저장

2) 2차 정규화(2NF)

부분 함수 종속 제거 (완전 함수적 종속 관계)

EX) <고객명, 서비스 이름, 서비스 가격, 서비스 이용 기간>을 한 테이블에 두는 것은 부분 함수 종속성

⇒ <고객명, 서비스 이름> → <서비스 이용 기간> / <서비스 이름> → <서비스 가격>

부분 관계인 <서비스 이름, 서비스 가격> 관계를 별도의 테이블로 분리하여 부분 함수 종속성 제거

3) 3차 정규화(3NF)

이행 함수 종속 제거, 속성에 종속적인 속성을 분리

A → B, B→C = A → C 관계

EX) <책번호, 출판사, 홈페이지>를 한 테이블에 두는 것은 이행 함수 종속성

책번호 → 출판사, 출판사 → 홈페이지 = 책번호 → 홈페이지

⇒ <책번호, 출판사> / <출판사, 홈페이지> 테이블로 분리하여 이행 함수 종속성 제거

4) 보이스-코드 정규화(BCNF)

결정자 함수 종속, 모든 결정자가 후보키.

EX) <학번, 과목명, 교수명>를 한 테이블에 두는 것은 <교수명>이 결정자이지만 후보키가 아니기 때문에 보이스-코드 정규화를 만족 못 함

<학번, 과목명> → <교수명>, <교수명> → <과목명> - <교수명>은 <과목명>에 영향 주지만, 한 테이블에 같이 존재하고 <교수명>은 키가 아닌 상황이므로 결정자인 <교수명>이 후보키가 아니다.

⇒ <교수명> → <과목명> 이기 때문에 <교수명, 과목명>로 분리, 교수명이 후보키 역할 하도록 함.

5) 4차 정규화(4NF)

다치(다중 값) 종속성 제거, 특정 속성 값에 따라 선택적인 속성을 분리

EX) <개발> 마다 <자격증> 값들이 열러 개 존재, 특정 <개발자>마다 <언어> 값들이 여러 개 존재 - 다치 종속 관계

⇒ <개발자, 자격증>, <개발자, 언어> 테이블로 분리

6) 5차 정규화(5NF)

조인 종속성 제거

4차 정규화 테이블에 대해 조인 연산을 수행하면 4차 정규화 수행 전 데이터와 다르게 되는 문제인 조인 종속성 발생

EX) <개발자, 자격증>, <개발자, 언어> 뿐 아니라 <자격증, 언어> 관계에 대한 테이블 추가하여

조인 했을 때 정확히 원래의 데이터로 복원 가능

2. 물리 데이터 저장소 설계

(1) 물리 데이터 모델 설계

1) 물리 데이터 모델링

▼ 개념

- 논리 모델을 적용하고자 하는 기술에 맞도록 상세화 해가는 과정
- DDL 이용해 데이터 모델 정의

▼ 변환 절차

1) 개체 → 테이블

- 테이블과 개체 명칭 동일
- 테이블 - 영문명
- 개체 - 한글명

2) 속성 → 컬럼

- 가독성 위해 짧은 컬럼 명칭
- 미리 정의된 표준에 의해 복합 단어를 사용할 경우 명명

3) UID → 기본키

- 개체의 UID에 해당하는 모든 속성에 대해 기본키로 선언
- NOT NULL, UNIQUE 등의 제약 조건 추가 정의
- 관계에 의한 외래키가 기본키에 포함될 수 있음

4) 관계 → 외래키

- 순환 관계에서 자신의 기본키는 외래키로 정의

5) 컬럼 유형과 길이 정의

- 유형 정의 및 데이터 최대 길이 파악하여 선정
- CHAR, VARCHAR2, NUMBER, DATE, BLOB, CLOB

6) 반 정규화 수행

- 시스템 성능 향상과 개발 및 운영의 단순화를 위해 데이터 모델을 통합하는 반 정규화 수행
- 중복 테이블 추가 - 특정 부분만 포함하는 테이블 추가, 집계 테이블 추가
- 테이블 조합 - 1:1, 1:M 관계 테이블 조합, 수퍼타입 / 서브타입 테이블 조합
- 테이블 분할 - 수직, 수평 분할
- 테이블 제거 - 테이블 재정의, 접근하지 않는 테이블 제거
- 컬럼 중복화 - 조인 성능 향상을 위한 중복 허용

(2) 물리 데이터 저장소 구성

1) 테이블 제약 조건 (Constraint) 설계

▼ 참조 무결성 제약 조건

- 릴레이션과 릴레이션 사이에 대해 **참조의 일관성을 보장**하기 위한 조건
- 두 개의 릴레이션이 기본키, 외래키를 통해 참조 관계 형성하는 경우, 참조하는 외래키의 값은 항상 참조되는 릴레이션에 기본키로 존재해야 함.

i) 제한(Restricted)

참조 무결성 원칙을 위배하는 연산을 거절하는 옵션

ii) 연쇄(Cascade)

참조되는 릴레이션에서 튜플(행)을 삭제하고, 참조되는 릴레이션에서 이 튜플을 **참조하는 튜플들도 함께 삭제**하는 옵션

iii) 널 값(Nullify)

참조되는 릴레이션에서 튜플을 삭제하고, 참조하는 릴레이션에서 해당 튜플을 참조하는 튜플들의 외래키에 NULL값을 넣는 옵션

만일 릴레이션을 정의할 때 참조하는 릴레이션에서 NULL값이 들어갈 애트리뷰트에 NOT NULL 명시 되어있으면 삭제 연산 거절

```
ALTER TABLE EMP ADD CONSTRAINT emp_dt_fk
FOREIGN KEY (deptno)
REFERENCES DEPT (deptno)
ON DELETE { RESTRICT | CASCADE | SET NULL };
```

2) 인덱스 (Index) 설계

인덱스 적용 기준, 컬럼 선정 등 고려하여 설계

▼ 개념

- 검색 연산의 최적화, 데이터 베이스 내 열에 대한 정보를 구성한 데이터 구조
- 전체 데이터 검색 없이 **필요한 정보에 대해 신속한 조회 가능**

▼ 적용 기준

- 인덱스 분포도 10~15% 이내인 경우 수식 참고

분포도 = (1 / (컬럼 값의 종류)) * 100

분포도 = (컬럼 값의 평균 Row 수) / (테이블의 총 Row 수) * 100

- 조회 및 출력 조건으로 사용 되는 컬럼인 경우 적용
- 인덱스 자동 생성 기본키와 Unique 키의 제약 조건 사용할 경우 적용

▼ 컬럼 선정

- 분포도가 좋은 컬럼은 단독적으로 생성
- **자주 조합되어 사용** 되는 컬럼은 **결합 인덱스**로 생성
- 결합 인덱스는 구성되는 컬럼 순서 선정(사용빈도, 유일성, 정렬 등)에 유의

- **수정이 빈번하지 않은 컬럼을 선정**

▼ 설계 시 고려 사항

- 지나치게 많은 인덱스는 오버헤드로 작용
- 인덱스는 추가적인 저장 공간이 필요
- 넓은 범위를 인덱스 처리 시 오히려 전체 처리보다 많은 오버헤드를 발생시킬 수 있음
- 인덱스와 테이블의 저장 공간을 적절히 분리되도록 설계

3) 뷰 (View) 설계

▼ 속성

- REPLACE - 뷰 이미 존재 → 재생성
- FORCE - 본 테이블 존재 관계 없이 뷰 생성
- NOFORCE - 기본 테이블 존재 → 뷰 생성
- WITH CHECK OPTION - 서브 쿼리 내의 조건을 만족하는 행만 변경
- WITH READ ONLY - DML 작업 불가

▼ 고려 사항

- 뷰 사용에 따라 수행속도에 문제가 발생할 수 있음
- 최적의 액세스 경로를 사용할 수 있도록 해야 함

4) 클러스터 (Cluster) 설계

▼ 적용 기준

- 인덱스 단점 해결한 기법, 분포도가 넓을수록 유리
- 액세스 기법이 아니라 액세스 효율 향상을 위한 물리적 저장 방법
- 분포도가 넓은 테이블의 클러스터링은 저장 공간의 절약 가능
- 대량의 범위를 자주 액세스하는 경우 적용
- 인덱스를 사용한 처리 부담이 되는 넓은 분포도에 활용

- 여러 테이블이 빈번하게 조인을 일으킬 때 활용

▼ 고려 사항

- 검색 효율은 높여주나 입력, 수정, 삭제 시는 부하가 증가
- UNION, DISTINCT, ORDER BY, GROUP BY가 빈번한 컬럼이면 검토 대상
- 수정이 자주 발생하지 않는 컬럼은 검토 대상
- 처리 범위가 넓어 문제 발생 경우, 단일 테이블 클러스터링 고려
- 조인이 많아 문제 발생 경우, 다중 테이블 클러스터링 고려

5) 파티션 (Partition) 설계

▼ 분류

▼ 레인지 파티셔닝

- 연속적인 숫자나 날짜를 기준으로 파티셔닝 기법
- 손쉬운 관리 기법을 제공하여 관리 시간의 단축 가능

▼ 해시 파티셔닝

- 파티션 키의 해시 함수 값에 의한 파티셔닝 기법
- 균등한 데이터 분할 가능, 질의 성능이 향상
- 파티션을 위한 범위가 없는 데이터에 적합

▼ 리스트 파티셔닝

- 특정 파티션에 저장 될 데이터에 대한 명시적 제어가 가능한 파티셔닝 기법
- 분포도가 비슷하고 데이터가 많은 SQL에서 컬럼의 조건이 많이 들어오는 경우 유용

▼ 컴포지트 파티셔닝

- 범위 분할에 이후 해시 함수를 적용하여 재분할 하는 파티셔닝 기법
- 큰 파티션에 대한 I/O 요청을 여러 파티션으로 분산

- 레인지 파티셔닝할 수 있는 컬럼이나, 파티션이 너무 커서 효과적으로 관리할 수 없을 때 유용

▼ 장점

- 성능 향상 - 데이터 액세스 범위를 줄여 성능 향상
- 가용성 향상 - 전체 데이터의 훼손 가능성이 감소 및 데이터 가용성 향상
- 백업 가능 - 분할 영역을 독립적으로 백업하고 복구 가능
- 경합 감소 - 입출력 성능 향상, 디스크 컨트롤러에 대한 경합 감소

6) 디스크 (Disk) 구성 설계

- 정확한 용량 산정하여 디스크 사용 효율 높임
- 업무량이 집중되 있는 디스크를 분리 설계
- 입출력 경합을 최소화하여 데이터의 접근 성능 향상
- 디스크 구성에 따라 테이블스페이스 개수와 사이즈 등을 결정
- 파티션 수행 테이블은 별도로 분류

3. 데이터 조작 프로시저 작성

(1) 데이터 조작 프로시저 개발

1) 프로시저

▼ 개념

- SQL을 이용해 생성된 데이터를 조작하는 프로그램
- 데이터베이스 내부에 저장 되고 일정한 조건이 되면 자동으로 수행
- 저장된 프로시저 - 배치 작업, 복잡한 트랜잭션을 수행하는 PL/SQL 문을 DB에 저장하는 기능을 제공하는 프로그램
- 저장된 함수 - 저장 프로시저와 용도는 비슷하나 실행 결과를 되돌려 받을 수 있는 프로그램
- 저장된 패키지 - 프로시저나 함수를 효율적으로 관리하기 위해 패키지 단위로 배포할 때 사용하는 프로그램

- 트리거 - 특정 테이블에 삽입, 수정, 삭제 등의 변경 이벤트 발생 시 DBMS에서 자동 실행 되도록 구현된 프로그램

▼ PL/SQL

▼ 개념

- 테이터 조작 언어
- Oracle 기반의 모든 프로시저 작성에 사용되며 표준SQL의 확장 기능이 우수

▼ 작성 절차

Java 환경의 경우 JDBC 통해 연결

1) 데이터 저장소 연결

i. 드라이버 로딩 - DB와 연결하기 위해 DBMS에서 제공하는 JAR파일 드라이버를 메모리에 적재

EX) orcle.jdbc.driver.OracleDriver;

ii. 데이터베이스 연결 - 해당 드라이버를 사용하여 데이터베이스를 연결

EX)

String url = "jdbc:oracle:thin:@localhost:1521:ORCL";

conn = DriverManager.getConnection(url, "scott", "tiger");

iii. 쿼리 전달 - 쿼리를 DB로 전달하기 위해 Statement, PreparedStatement 객체 생성

EX) pstmt = conn.prepareStatement(sql);

iiii. 결과 수신 - 전달된 쿼리의 수행으로 인한 반환 값 수신

EX) ResultSet rs = pstmt.executeQuery();

2) 데이터 저장소 정의

- 생성 - CREATE TABLE EMP ();

- 수정 - ALTER TABLE DEPT MODIFY (DEPT_NAME VARCHAR2(20));
- 삭제 - DROP TABLE DEPT;

3) 데이터 조작 프로시저 작성

```
CREATE OR REPLACE PROCEDURE NAME
( param1 data_type )
IS [AS]
선언부
BEGIN
실행부
EXCEPTION
예외처리부
END;
```

```
-- 프로시저 명
CREATE OR REPLACE PROCEDURE INPUT_EMP
( P_NAME IN VARCHAR2,
  P_EMPNO IN NUMBER )
IS
BEGIN
-- 기능
  INSERT INTO EMP (NAME, EMPNO)
      VALUES (P_NAME, P_EMPNO);
COMMIT;
END;
-- 프로시저 호출
EXEC INPUT_EMP('dobi', '123');
```

4) 데이터 검색 프로시저 작성

- 검색 조건에 맞는 데이터 조회

```
-- 프로시저 명
CREATE OR REPLACE PROCEDURE TEST_SEARCH_PROC
(
  P_REG_DATE IN VARCHAR2,
  P_ITEM_CODE IN VARCHAR2,
  P_COMPANY_CODE IN VARCHAR2
)
IS
BEGIN
-- 조회
  SELECT A.REG_DATE, B.ITEM_CODE, B. COMPANY_CODE
    FROM COMPANY A LEFT OUTER JOIN ITEM B
      ON A.COMPANY_CODE = B.COMPANY_CODE
```

```
    WHERE P_COMPANY_CODE = A.COMPANY_CODE
      AND P_REG_DATE = B.REG_DATE
      AND P_ITEM_CODE = B.ITEM_CODE
    RETURN;
END;
```

(2) 데이터 조작 프로시저 테스트

1) PL/SQL 테스트

i) DBMS_OUTPUT 패키지 활용

- 메시지를 버퍼에 저장하고 버퍼로부터 메시지를 읽어오기 위한 인터페이스 패키지 DBMS_OUTPUT을 코드에 포함

ii) 사례

```
CREATE OR REPLACE PROCEDURE TYPE_TEST
( p_empno IN emp.empno%TYPE )
IS
  -- %TYPE 데이터형 변수 선언
  v_empno emp.empno%TYPE;
  v_ename emp.ename%TYPE;
  v_sal emp.sal%TYPE;
BEGIN
  DBMS_OUTPUT.ENABLE;
  -- %TYPE 데이터형 변수 사용
  SELECT empno, ename, sal
    INTO v_empno, v_ename, v_sal
    FROM EMP
  WHERE empno = p_empno;

  -- 결과값 출력
  DBMS_OUTPUT.PUT_LINE('직원번호: ' || v_empno);
  DBMS_OUTPUT.PUT_LINE('직원이름: ' || v_ename);
  DBMS_OUTPUT.PUT_LINE('직원급여: ' || v_sal);
END;
```

- DMBS_OUPUT.DISABLE - 메시지 버퍼 내용삭제
- DBMS_OUTPUT.ENABLE - 메시지 버퍼 내용 할당
- DBMS_OUTPUT.PUT - 메시지의 마지막 라인 끝에 신규 라인 문자(EOL) 추가
- DBMS_OUTPUT.GET_LINE - 한 번 호출될 때마다 하나의 라인만을 읽어옴
- DBMS_OUTPUT.GET_LINES - 지정된 라인을 모두 읽음

iii) 실행 방법

- 출력하기 위한 SERVEROUTPUT을 ON시키고, 실행하고자 하는 PL/SQL 블록 또는 저장 객체 명 호출
- 오류 발생 시 'SHOW ERRORS' 명령어 통해 오류 내용 확인

```
SET SERVEROUTPUT ON
EXECUTE TYPE_TEST(7369);
```

2) 저장 객체 테스트

i) 저장된 함수

- 저장 객체를 테스트하기 위한 함수를 사용

```
-- 변수 선언
VAR salary NUMBER;
-- 함수 실행
EXECUTE: salary := update_sal(1004);
-- 변수 출력
PRINT salary;
```

ii) 저장된 프로시저

- 실행하기 전 프로시저 실행 후 변경될 이전의 값을 확인

```
SET SERVEROUTPUT ON;
EXECUTE update_sal(1004);
```

iii) 저장된 패키지

- 패키지 실행은 패키지명, 프로시저(함수)명으로 기술

```
SET SERVEROUTPUT ON;
EXEC emp_info.all_emp_info;
```

iiii) 트리거

DBMS_OUTPUT.PUT_LINE 출력하여 SET SERVEROUTPUT ON 실행

```
-- 트리거가 처리될 조건에 부합되는 SQL을 실행하여 데이터 처리결과 확인
SET SERVEROUTPUT ON;
```

4. 데이터 조작 프로시저 최적화

(1) 데이터 조작 프로시저 성능 개선

1) 쿼리 성능 개선(튜닝)

▼ 개념

- 데이터베이스에서 프로시저에 있는 SQL 실행 계획을 분석, 수정을 통해 최소의 시간으로 원하는 결과를 얻도록 프로시저를 수정하는 작업
- SQL 성능 개선을 통해 데이터 조작 프로시저의 성능 개선이 가능

▼ **개선 절차**

1. 문제 있는 SQL 식별

- 문제 있는 SQL을 식별하기 위해 애플리케이션의 성능을 관리 및 모니터링 도구인 APM 활용
- APM: 부하량, 접속자 파악 및 장애 진단 등을 목적으로 하는 성능 모니터링 도구

2. 옵티마이저 통계 확인

- 옵티마이저는 개발자가 작성한 SQL을 가자 빠르고 효율적으로 수행할 최적의 처리경로를 생성해주는 데이터베이스 핵심 모듈

3. SQL문 재구성

- 범위가 아닌 특정 값 지정으로 범위를 줄여 처리 속도를 빠르게 함
- 옵티마이저가 비정상적인 실행계획을 수립할 경우, 힌트로서 옵티마이저의 접근경로 및 조인 순서를 제어

4. 인덱스 재구성

- 성능에 중요한 액세스 경로를 고려하여 인덱스 생성
- 실행 계획을 검토하여 기존 인덱스의 열 순서를 변경/추가

5. 실행 계획 유지관리

- 데이터베이스 버전 업그레이드, 데이터 전환 등 시스템 환경의 변경 사항 발생 시에도 실행 계획이 유지되고 있는지 관리

2) 옵티마이저 통계 확인

▼ 개념

- SQL을 가장 빠르고 효율적으로 수행할 최적의 처리 경로를 생성해주는 DBMS 내부 핵심 엔진
- 옵티마이저가 생성한 SQL 처리 경로를 실행 계획이라고 부름

▼ 유형

1. RBO (Rule Based Optimizer)

- 개념 - 통계 정보가 없는 상태에서 사전 등록된 규칙에 따라 질의 실행 계획을 선택하는 옵티마이저
- 핵심 - 규칙(우선 순위) 기반
- 평가 기준 - 인덱스 구조, 연산자, 조건절 형태 등
- 장점 - 사용자가 원하는 처리 경로로 유도하기 쉬움

2. CBO (Cost Based Optimizer)

- 개념 - 통계 정보로부터 모든 접근 경로를 고려한 질의 실행 계획을 선택하는 옵티마이저
- 핵심 - 비용(수행 시간) 기반
- 평가 기준 - 레코드 개수, 블록 개수, 평균 행 길이, 컬럼 값의 수, 컬럼 값 분포, 인덱스 높이, 클러스터링 팩터 등
- 장점 - 옵티마이저의 이해도가 낮아도 성능 보장 가능(기본 설정)

▼ 역할

- 쿼리 변환 - SQL을 좀 더 일반적이고 표준화된 형태로 변환
- 비용 산정 - 쿼리 명령어 각 단계의 선택도(전체 대상 레코드 중 특정 조건에 의해 선택될 것으로 예상되는 레코드 비율), 카디널리티(튜플 개수), 비용을 계산. 실행 계획 전체에 대한 총비용 계산
- 계획 생성 - 하나의 쿼리를 수행 시 후보군이 될 만한 실행 계획들을 생성 해내는 역할

▼ 힌트 사용 (실행하려 하는 SQL문에 사전에 정보를 주어 SQL문 실행에 빠른 결과를 가져오는 효과 제공)

- SQL 성능 개선 핵심 부분. 옵티마이저의 실행 계획을 원하는 대로 변경 가능
- 옵티마이저가 항상 최선의 실행 계획을 수립할 수 없어 명시적인 힌트를 통해 실행 계획을 변경

```
SELECT /*+ RULE */ ENAME, SAL
  FROM EMP
  WHERE EMPNO > 9000;
```

3) SQL문 재구성

▼ SQL문 재구성 가이드

- 특정 값 지정

조건절의 '>' 또는 '<'가 아닌 **'='을 사용**

범위가 아닌 **특정 값 지정**으로 인한 범위 줄임

- 별도의 SQL 사용

다양한 작업에 대해 하나의 SQL문을 사용할 경우 각 작업에 최적화되지 않은 결과 발생

하나의 SQL문 사용 시 UNION ALL 연산자 사용

- 힌트 사용

옵티마이저가 비정상적인 실행 계획을 수립 시 힌트로서 액세스 경로 및 조인 순서를 제어

- HAVING 미사용

인덱스가 걸려있는 컬럼은 HAVING 사용 시 인덱스 미사용

- 인덱스만 질의 사용

가능한 인덱스만 이용해 질의를 수행하여 옵티마이저가 최적의 경로를 찾도록 유도

4) 인덱스 재구성

인덱스를 재구성하거나 새로 생성하여 성능 개선에 참고

▼ 인덱스 재구성 가이드

- 자주 쓰는 컬럼 선정 - 조건절에 항상 사용되거나, 자주 사용되는 컬럼 선정
- SORT 명령어 생략 - SORT 명령어를 생략하기 위한 컬럼 추가
- 분포도 고려 - 분포도가 좋은 컬럼은 단독으로 인덱스 생성
- 변경 적은 컬럼 선정 - 데이터의 변경이 적은 컬럼에 인덱스를 생성
- 결합 인덱스 사용 - 인덱스들이 자주 조합될 때 결합 인덱스 생성