과목1
1장 데이터 모델링의 이해

제1절 데이터 모델의 이해

1. 모델링
- 다양한 현상을 추상화, 단순화하여 일정한 표기법에 의해 표현하는 것
- 모델이란 현실 세계의 추상화 된 반영

2. 모델링 특징
- **추상화** : 일정한 형식에 맞춰서 표현
- **단순화** : 제한된 표기법이나 언어로 표현
- **명확화**(=정확화) : 애매모호함을 제거하여 이해가 쉽게 표현

3. 모델링의 관점
- 데이터 관점 (what) : 업무와 데이터 및 데이터 사이의 관계를 모델링
- 프로세스 관점 (how) : 업무가 실제로 하고있는 일, 해야 하는 일 모델링
- 데이터와 프로세스의 상관 관점 (interaction) : 데이터에 대한 업무 처리 방식의 영향을 모델링

* 데이터 모델링의 중요성과 유의점
(1) 중요성 : 파급효과, 간결한 표현, 데이터 품질 유지
(2) 유의점
① 중복 : 여러 장소에 같은 정보 저장 X
② 비유연성 : 데이터의 정의를 데이터 사용 프로세스와 분리
③ 비일관성 : 모델링 할때 데이터 간 상호 연관 관계 명확히 정의

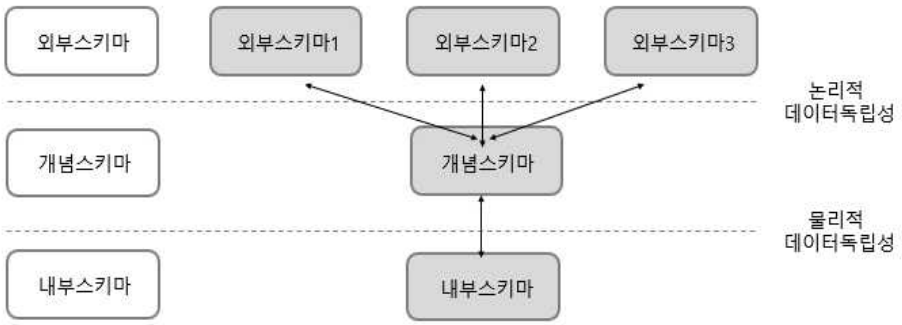
4. 데이터 모델링의 3단계
- 개념적 모델링 (계획/분석) : ERD 도출, 업무중심, 포괄적인 수준의 모델링
- 논리적 모델링 (분석) : 테이블 도출, (key, 속성, 관계)를 표현, 재사용성, 정규화 수행
- 물리적 모델링 (설계) : DB구축, 물리적 성격, 개념적보다 구체적

5. 데이터 독립성
- 데이터의 구조가 변경되어도 응용 프로그램이 변경될 필요가 없음
- 논리적 독립성 + 물리적 독립성으로 실현됨
- 필요성 : 데이터 중복성과 복잡도가 증가 → 요구사항 대응 저하, 유지보수 비용 증가

* 데이터베이스 스키마 (Schema)
- 데이터 모델링 대상 (**틀**)
- 데이터베이스 구조, 데이터 타입, 제약조건에 대한 명세
- 데이터베이스 설계 단계에서 명시되며 자주 변경되지 않음

* 데이터베이스의 3단계 구조
- 외부 스키마 : 응용프로그램 관점에서의 요구사항, 사용자 관점, DB 정의
- 개념 스키마 : 외부 스키마가 필요로 하는 데이터 모두 모아놓은 것. 설계자 관점
(=DB에 저장되는 데이터와 사용자 관계 표현 → 모든 사용자 관점 통합, 조직 전체 통합)
- 내부 스키마 : 데이터베이스가 물리적으로 저장된 형식, 개발자 관점

⇨ 데이터 모델링은 통합 관점의 개념 스키마를 만들어 가는 과정



* 사상 (Mapping)
- 상호 독립적인 개념을 연결시켜주는 다리
- 논리적 사상 : 외부화면 및 사용자 인터페이스 스키마 구조는 개념스키마와 연결
- 물리적 사상 : 개념스키마 구조와 물리적 저장된 구조(테이블 스페이스)와 연결

* 논리적 독립성
- 논리적 사상 (외부적-개념적) 을 통해 논리적 독립성 보장
- 개념 스키마가 변경되어도 외부 스키마에는 영향 X
- 논리적 구조가 변경되어도 응용 프로그램에는 영향이 없음, 통합구조 변경 가능

* 물리적 독립성
- 물리적 사상 (개념적-내부적) 을 통해 물리적 독립성 보장
- 내부스키마가 변경되어도 외부/개념 스키마는 영향 X
- 응용프로그램과 개념스키마에 영향 없이 저장장치 구조변경 가능, 물리적 구조

6. 데이터 모델링 3요소 (개체, 속성, 관계)
- 업무가 관여하는 **어떤 것** (Thing) : 엔터티타입, 엔터티 / 엔터티, 인스턴스, 어커런스
- 어떤 것이 가지는 **성격** (Attributes) : 속성 / 속성값
- 어떤 것들 간의 **관계** (Relationships) : 관계 / 페어링

* 데이터베이스 인스턴스 (Instance)
- 특정 시점에 데이터베이스에 실제로 저장되어 있는 데이터의 **값**

개념	복수/집합개념 타입/클래스	개별/단수개념 어커런스/인스턴스
어떤 것	엔터티 타입	엔터티
	엔터티	인스턴스/어커런스
연관	관계	페어링
성격	속성	속성값

7. ERD (Entity Relationship Diagram)
- 데이터 모델 표기법 : 엔터티를 사각형, 관계를 마름모, 속성을 타원형으로 표현

8. ERD 표기법을 이용하여 모델링 하는 방법
① 엔터티를 그린 후 적절하게 배치
② 엔터티 간 관계 설정
- 식별자 관계를 우선 설정함
  (식별자 관계 : 부모로부터 상속받은 FK(외래키)가 자식의 PK(기본키)의 일부가 되는 관계)
- 가급적 Cycle 관계도 발생하지 않아야 한다
③ 관계명 기술 (양 방향)
- 현재형 사용, 지나치게 포괄적인 단어는 지양
- 실제 프로젝트에서는 크게 고려 X
④ 관계차수, 관계의 참여도, 선택성 표시

9. 좋은 모델링의 요건
- **완전성** : 업무에서 필요로 하는 모든 데이터가 데이터 모델에 정의되어야 한다
- **중복배제** : 동일한 사실은 반드시 한번만 기록
- **업무규칙** : 업무규칙이 데이터 모델에 표현되어야 한다
- **데이터 재사용** : 회사 전체 관점에서 공통 데이터 도출, 전 영역 사용할 수 있도록 설계
- **통합성** : 동일한 데이터는 조직의 전체에서 한번만 정의되고 이를 참조, 활용

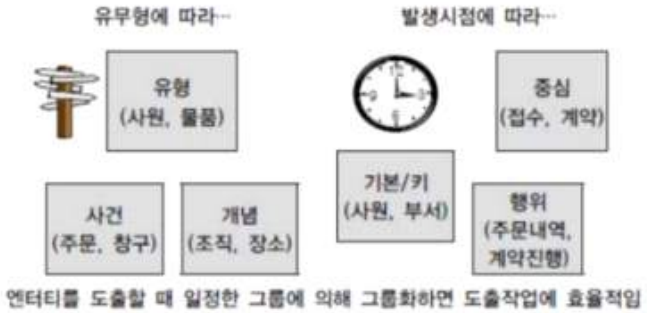
제2절 엔터티

1. 엔터티
- 업무에 필요한 정보를 저장하고 관리하기 위한 집합적인 것 (실체, 객체)
- 엔터티는 인스턴스의 집합 (인스턴스는 엔터티 안의 데이터)

엔터티-인스턴스의 예

엔터티	인스턴스
과 목	수 학
	영 어
강 사	이춘식
	조시형
사 건	2010-001
	2010-002

2. 엔터티의 분류



* 유형, 무형에 따른 분류
① 유형(Tangible) 엔터티
- 물리적인 형태가 있고 안정적이며 지속적 활용 ex) 교수, 강의실, 학생
② 개념(Conceptual) 엔터티
- 물리적인 형태는 존재하지 않으나 관리해야 할 개념적 정보 ex) 수업, 보험상품
③ 사건(Event) 엔터티
- 업무 수행 과정에서 발생, 비교적 발생량 많음 (통계자료에 이용) ex) 수강신청, 주문

* 발생시점에 따른 엔터티 (기본 / 중심 / 행위)
① 기본(key) 엔터티 : 독립적으로 생성되는 엔터티
② 중심(main) 엔터티 : 기본 엔터티와 행위 엔터티 중간에 존재하는 엔터티
③ 행위(active) 엔터티 : 2개 이상의 부모 엔터티로부터 발생, 비즈니스 프로세스를 실행하면서 생성되는 엔터티, 지속적 정보가 추가되고 변경되어 데이터양이 가장 많음

3. 엔터티의 특징
- 업무에서 필요로 하는 정보 포함
- 유일한 식별자를 가짐 (식별자에 의해 식별이 가능하도록, 관계엔터티 예외)
- 2개 이상의 인스턴스를 포함함 (인스턴트의 집합)
- 업무 프로세스에 이용됨
- 속성 없이 엔터티의 이름만 존재할 수 없음 (속성이 포함)
- 다른 엔터티와 최소 1개 이상의 관계가 존재 (통계성, 코드성, 내부필요 엔터티 예외)

4. 엔터티의 명명
- 엔터티 생성 의미대로, 실제 업무에서 사용하는 용어를 사용
- 약어를 사용 X, 단수명사 사용
- 이름이 동일한 엔터티가 중복으로 존재 X

제3절 속성

1. 속성의 정의
- 사물의 특징 또는 본질적인 성질 (속성이 없다면 실체를 생각할 수 없다)
- 인스턴스에 대해 의미상 더 분리되지 않는 최소의 데이터 단위
- 엔터티에 속한 인스턴스들의 성격을 구체적으로 나타냄
- 엔터티, 인스턴스, 속성, 속성값의 대응

* **엔터티, 인스턴스 속성의 관계**
- 1개의 엔터티 : 2개 이상의 인스턴스 집합 가짐
- 1개의 인스턴스 : 2개 이상의 속성을 가짐
- 1개의 속성 : 1개의 속성값을 가짐

2. 속성의 특징
- 해당 업무에서 필요하고 관리해야 하는 정보
- 모든 속성은 정해진 주식별자에 함수적으로 종속되어야 함
- 하나의 속성은 한 개의 값만을 가짐 (다중값인 경우 해당 속성을 별로의 엔터티로 분리)

3. 속성의 명명
- 현업에서 사용하는 이름을 부여
- 약어 사용은 가급적 X
- 수식어/소유격 X, 서술식 속성명 X, 명사형 속성명 O
- 전체 데이터 모델에서 유일성 확보

4. (속성의) **도메인**
- 각 속성이 가질 수 있는 값의 범위
  ex) 학점 : 0.0~4.5 사이의 실수 / 주소 : 길이 20자리 이내 문자열
- 엔터티 내에서 속성에 대한 데이터타입과 크기, 제약사항을 지정

5. 속성의 분류

1) 특성에 따른 분류
- 기본 속성 : 가장 일반적인 속성으로, 원래의 업무로부터 유래된 속성
- 설계 속성 : 데이터 모델링을 위해 새로 만든 속성(코드, 일련번호)
- 파생 속성 : 다른 속성들로부터 유도된 속성(통계, 계산된 값)

2) 분해 가능 여부에 따른 분류
- 단일 속성 : 하나의 의미
- 복합 속성 : 여러 의미, 단일 속성으로 분해 가능
- 단일값 속성 : 속성 1개 한 개의 값
- 다중값 속성 : 속성 1개에 여러 값, 엔터티로 분해 가능

3) 엔터티 구성방식에 따른 분류
- 기본키 속성 (PK, Primary Key) : 엔터티의 인스턴스를 구별할 수 있는 속성
- 외래키 속성 (FK, Foreign Key) : 타 엔터티의 PK를 참조하는 속성
- 일반 속성 : 엔터티에 포함되고 PK나 FK속성이 아닌 속성

제4절 관계

1. 관계와 페어링(Pairing)
- 관계 : 엔터티의 인스턴스 사이의 논리적인 연관성으로서 존재의 형태로서나 행위로서 서로에게 연관성이 부여된 상태
- 페어링 : 엔터티 내 인스턴스간 개별적 관계 → 이것의 집합을 관계로 표현
- 인스턴스의 집합 ⇨ **엔터티 페어링의 집합** ⇨ **관계**

2. 관계의 분류

1) ERD : 존재에 의한 관계 / 행위에 의한 관계 (구분 없이 단일화된 표기법 사용)

2) UML : 연관 관계 / 의존 관계 (실선과 점선 표기법으로 구분)

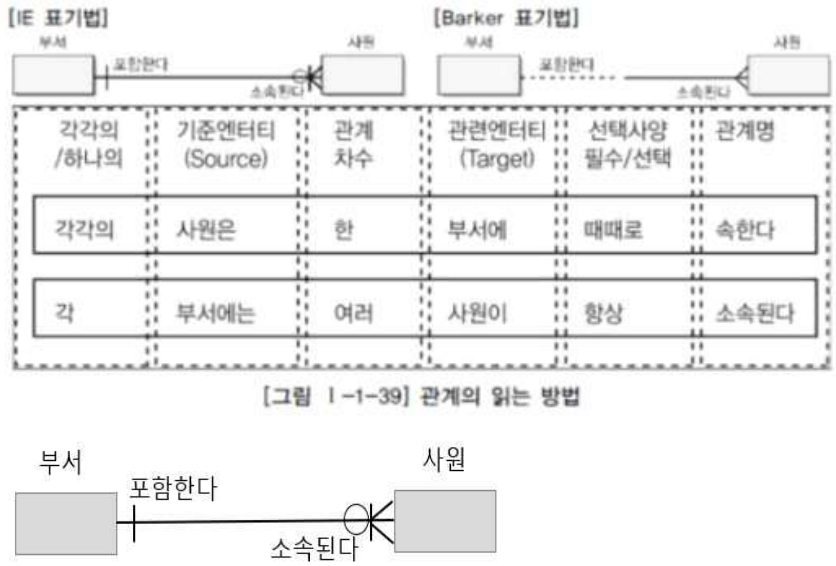
3. 관계의 표기법

① 관계명 : 엔터티가 관계에 참여하는 형태. 각 관계는 2개의 관계명 및 관점을 가짐

② 관계차수(Cardinality) : 1:1, 1:M, M:M (관계 엔터티 이용)

③ 관계선택사양 : 필수참여(Mandatory), 선택참여(Optional) (필수는 I 선택은 O로 표시)

4. 관계 읽기



[그림 I-1-39] 관계의 읽는 방법

제5절 식별자

1. 식별자의 개념

- 하나의 엔터티에 구성되어있는 여러 개 속성 중 엔터티를 대표할 수 있는 속성을 의미
- 하나의 엔터티는 반드시 하나의 유일한 식별자가 존재

2. **식별자의 분류**

분류	식별자	설명
대표성	주식별자	- 엔터티 내에서 각 인스턴스를 구분가능 - 타 엔터티와 참조관계를 연결가능(primary key 에 해당)
	보조식별자	- 엔터티 내에서 각 인스턴스를 구분가능 - 대표성을 갖지 못해 참조관계 연결에 사용되지 않음(candidate key에 해당)
목적	내부식별자	- 자연스럽게 존재하는 식별자 (본질식별자)
	외부식별자	- 관계를 통해 유입되는 타 엔터티의 식별자(foreign key에 해당) - 주식별자 속성 또는 일반 속성으로 포함될 수 있음
속성 수	단일식별자	- 하나의 속성으로 구성된 식별자
	복합식별자	- 둘 이상의 속성으로 구성된 식별자
본질	본질식별자	- 업무에 의해 만들어지는 식별자
	인조식별자	- 원조식별자가 복잡한 구성을 가지고 있기 때문에 인위적으로 만든 식별자

3. 식별자의 특징

- **유일성** : 주식별자에 의해 엔터티 내의 모든 인스턴스들을 유일하게 구분함
- **최소성** : 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되야 함
- **불변성** : 주식별자가 지정되면 그 식별자의 값은 변하지 않아야 함
- **존재성** : 주식별자가 지정되면 반드시 데이터 값이 존재해야 함 (Null X)

4. 주식별자 도출 기준

- 유일성을 갖는 속성 중 해당 업무에서 자주 이용되는 속성을 지정
- 명칭, 내역 등과 같이 이름으로 기술되는 것들은 가능하면 주식별자로 지정 X
  → 구분자가 존재하지 않을 경우 새로운 식별자 생성 (일련번호, 코드 등)
- 복합식별자를 구성할 경우 너무 많은 속성이 포함되지 않아야 함
  → 주식별자 개수가 많을 경우 새로운 인조식별자를 생성

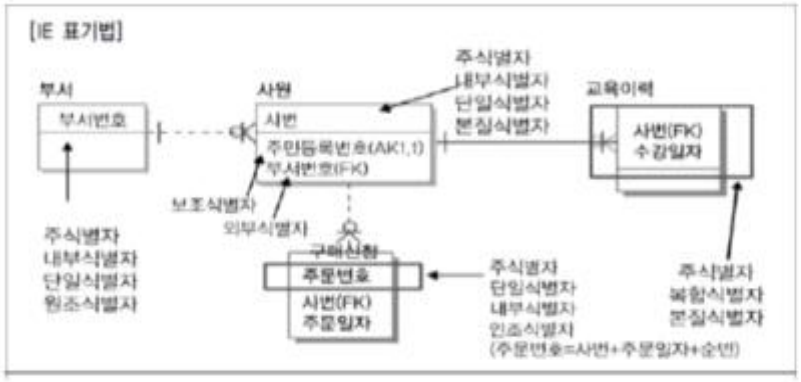
5. 식별자 관계와 비식별자 관계

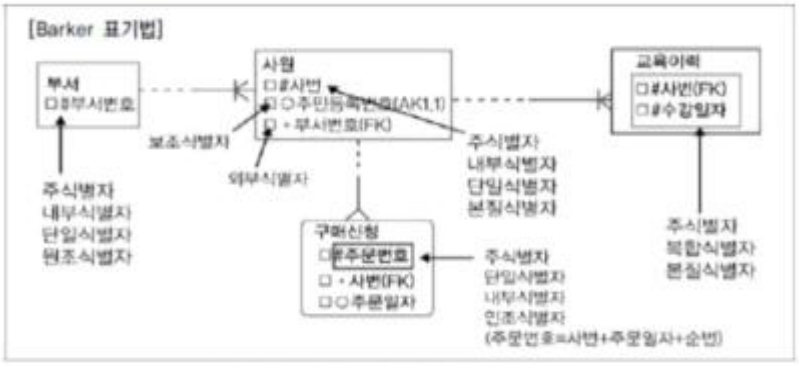
① 식별자 관계 (실선)

- 자식이 부모의 기본키(PK)를 상속받아 기본키로 사용하는 경우
- 강한 연결관계, 부모에 종속, Null X, 1:1 or 1:N 관계
- 문제점 : 자식의 주식별자 속성이 지속적으로 증가할 수 있음 → 복잡, 오류가능성

② 비식별자 관계 (점선)

- 부모로부터 속성을 받아 일반 속성으로 사용하는 경우 (약한 종속)
- 문제점 : 부모까지 조인, 불필요 현상 발생 → SQL 구문 길어져 성능 저하





2장 데이터 모델과 성능

제1절 성능 데이터 모델링의 개요

1. 성능 데이터 모델링 정의
- 데이터베이스 성능을 고려하여 데이터 모델링을 수행하는 것
- 정규화, 반정규화, 테이블 통합 및 분할, 조인 구조, PK / FK 설정 등

2. 수행 시점
- 빠를수록 좋음
- 분석/설계 단계에서 성능 모델링 수행 best → 재업무 비용 최소화

3. 성능 데이터 모델링 고려사항
- 정규화를 정확하게 수행 : 주요 관심사별로 테이블을 분산시킴
- 데이터베이스 용량산정 수행 : 각 엔터티에 어느 정도의 트랜잭션이 들어오는지 파악
- 데이터베이스에 발생되는 트랜잭션의 유형 파악 : CRUD 매트릭스 활용
- 용량과 트랜잭션의 유형에 따라 반정규화 수행 : 테이블, 속성, 관계 변경
- 이력모델의 조정, 인덱스를 고려한 PK/FK의 순서 조정, 슈퍼/서브타입 조정 등 수행
- 성능관점에서 데이터 모델 검증

제2절 정규화와 성능

1. 정규화 (Normalization)
- 정의 : 데이터 분해 과정, 이상현상 제거
- 목적 : 삽입/삭제/갱신 이상현상 방지
- 함수적 종속성에 기반한 정규화 수행 필요

2. **함수적 종속성** (FD, Fuctional Dependency)
- 데이터들이 어떤 기준값에 의해 종속되는 현상을 지칭, 결정자와 종속자의 관계, 결정자의 값으로 종속자의 값을 알 수 있음
- 결정자 ex) 학번, 주민등록번호
- 종속자 ex) 이름, 혈액형, 출생지, 주소
  (학번은 이름과 혈액형을 함수적으로 결정, 이름과 혈액형은 학번에 함수적으로 종속된다)

3. 종류
- 정규형 (NF : Normal Form) : 정규화로 도출된 데이터 모델이 갖춰야 할 특성
① 1NF (1 Normal Form) : 모든 값이 원자값을 가짐
② 2NF : 부분함수종속 제거
③ 3NF : 이행함수종속 제거 (식별자가 아닌 속성(주식별자의 일부 or 일반속성)이 결정자 역할하는 함수 종속 제거)

4. 정규화의 효과
- 성능 = 조회, 입력/수정/삭제 2가지로 분류
- 데이터 중복 감소 → 성능 향상
- 데이터가 관심사별로 묶임 → 성능 향상
- 조회 질의에서 조인이 많이 발생 → 성능 저하
⇨ 입력/수정/삭제의 경우 성능 향상
but ! 조회의 경우 처리조건에 따라 향상 or 저하

제3절 반정규화와 성능

1. 반정규화 (=역정규화 =Denormalization)
- 정의 : 정규화된 엔터티, 속성, 관계에 대해 성능 향상을 목적으로 중복, 통합, 분리를 수행하는 데이터 모델링 기법 (cf 비정규화 : 정규화를 수행하지 않음)

2. 특징
- 테이블, 칼럼, 관계의 반정규화를 종합적으로 고려 (일반적으로 속성(칼럼)의 중복 시도)
- 과도한 반정규화는 데이터 무결성을 침해

3. 반정규화 사전절차
1) 반정규화 대상 조사 : 데이터 처리 범위 및 통계성 등 조사
2) 다른 방법 검토 : 뷰, 클러스터링, 인덱스, 애플리케이션
3) 반정규화 적용 : 정규화 수행 후 반정규화 수행

4. 반정규화 기법
1) 테이블 반정규화
- 테이블 병합 : 관계 병합, 슈퍼/서브타입 병합 (one to one, plus, single type)
- 테이블 분할 : 수직, 수평 분할
- 테이블 추가 : 중복 테이블 / 통계 테이블 / 이력 테이블 / 부분 테이블 추가

2) 칼럼 반정규화
- 중복칼럼 추가 : 조인 횟수를 감소시키기 위해 다른 테이블의 칼럼 중복 칼럼 저장
- 파생칼럼 추가 : 값의 계산으로 인한 성능 저하 예방, 예상값을 미리 계산해서 중복 칼럼 저장 (Derived 칼럼)
- 이력테이블 칼럼 추가 : 기능성 칼럼, 대량 이력 데이터 처리의 성능 향상을 위해 종료 여부, 최근값 여부 등의 칼럼 추가로 저장
- PK의 의미적 분리를 위한 칼럼 추가 : PK가 복합 의미를 갖는 경우 단일 속성을 구성시 발생, 구성 요소 값의 조회 성능 향상을 위해 일반 속성을 추가
- 데이터 복구를 위한 칼럼 추가 : 사용자의 실수 또는 응용프로그램 오류로 인해 데이터가 잘못 처리된 경우 원래 값으로 복구 위해 이전 데이터를 임시로 중복 저장

3) 관계 반정규화
- 중복관계 추가 : 조인으로 정보 조회가 가능 but 조인 경로 단축을 위해 중복관계 추가

* 테이블과 칼럼의 반정규화는 데이터 무결성에 영향을 미침
* 관계의 반정규화는 데이터 무결성 보장 가능, 데이터처리 성능 향상

제4절 대량 데이터에 따른 성능

1. 성능 저하 원인
· 하나의 테이블에 데이터 대량집중 : 테이블 구조 너무 커져 효율성 ↓, 디스크 I/O ↑
· 하나의 테이블에 여러개의 컬럼 존재 : 디스크 점유량↑, 데이터 읽는 I/O량 ↑
· 대량의 데이터가 처리되는 테이블 : SQL 문장에서 데이터 처리 위한 I/O량 ↑, 인덱스 구성
· 대량의 데이터가 하나의 테이블에 존재 : 인덱스의 크기 ↑ 성능 저하
· 컬럼이 많아지는 경우 : 로우 체이닝, 로우 마이그레이션 발생

2. 해결 방안
- 한 테이블에 많은 칼럼 → 수직분할
- 대량 데이터 저장 문제 → 파티셔닝, PK에 의한 테이블을 분할

* 대량 데이터 발생에 따른 테이블 분할
- 수직분할 : 컬럼 단위로 분할하여 I/O 경감
- 수평분할 : 로우 단위로 분할하여 I/O 경감

3. 대량 데이터 발생으로 인한 현상
· 블록: 테이블의 데이터 저장 단위
· 블록 I/O 횟수 증가, 디스크 I/O 가능성 상승, 디스크 I/O 성능 저하

- 로우 체이닝 (Row Chaining) : 행(Row) 길이가 길어 데이터 블록 하나에 데이터를 모두 저장하지 않고 두 개 이상의 블록에 걸쳐 하나의 로우를 저장하는 형태
- 로우 마이그레이션 (Row Migration) : 수정된 데이터가 해당 블록에 저장하지 못하고 다른 블록의 빈 공간에 저장되는 현상

4. 파티셔닝 (Partitioning)
- 테이블 수평 분할 기법, 논리적으로는 하나의 테이블이지만 물리적으로 여러 데이터 파일에 분산 저장, 데이터 조회 범위를 줄여 성능 향상

- Range Partition : 범위로 분할 (고객번호 1~1000, 1001~2000 등)
- List Partition : 특정한 값을 기준으로 분할 (지역 : 서울, 부산 등)
- Hash Partition : 해시 함수를 적용하여 분할, 데이터 위치 알 수 없음

* 해시 함수 : 임의 길이의 데이터를 짧은 길이의 데이터로 매핑하는 함수
- Composite Partition : 여러 파티션 기법을 복합적으로 사용하여 분할

제5절 데이터베이스 구조와 성능

1. 슈퍼/서브타입 데이터 모델
- 논리적 데이터 모델에서 주로 이용 (분석단계에서 많이 쓰임)
- 물리적 데이터 모델로 설계 시 문제 발생 (적당한 노하우 X → 1:1 또는 All in one 타입이 되어버려 성능 저하)
- 슈퍼타입 : 공통부분을 슈퍼타입으로 모델링
- 서브타입 : 공통으로부터 상속받아 다른 엔터티와 차이가 있는 속성만 모델링

2. 데이터베이스 성능 저하 원인 3가지
① Union 연산에 의해 성능 저하
- 트랜잭션 : 전체를 일괄처리, 테이블 : 개별로 유지

② 조인에 의해 성능 저하
- 트랜잭션 : 슈퍼+서브타입 공통 처리, 테이블 : 개별로 유지

③ 불필요하게 많은 데이터 집적
- 트랜잭션 : 서브타입만 개별로 처리, 테이블 : 하나로 통합

3. 슈퍼/서브타입 데이터 모델 변환을 통한 성능 향상
- 변환기준 : 데이터 양, 트랜잭션 유형
- 데이터 소량 : 데이터 처리 유연성 고려하여 가급적 1:1 관계 유지
- 데이터 대량 : 3가지 변환 방법 (개별 테이블, 슈퍼+서브타입 테이블, 하나의 테이블)

① 1:1 타입 (One to one type)
: 개별로 처리하는 트랜잭션에 대해 개별 테이블 구성하여 1:1 관계 가짐
② 슈퍼/서브 타입 (Plus type)
: 슈퍼+서브 공통으로 처리하는 트랜잭션에 대해 슈퍼/서브 각각 테이블 구성
③ All in One 타입 (Single type)
: 전체를 하나로 묶어 트랜잭션이 발생, 단일 테이블 구성

구분	One to one type	plus type	Single type
특징	개별 테이블 유지	슈퍼+서브타입 테이블	하나의 통합 테이블
확장성	우수함	보통	나쁨
조인 필요 수	많음	보통	적음
I/O 성능저하	양호	양호	나쁨
관리 용이성	나쁨	나쁨	좋음
적합 트랜잭션 유형	개별 테이블로 접근이 많은 경우	슈퍼+서브 형식 데이터 처리가 많은 경우	전체에 대한 일괄 처리가 많은 경우

⇨ 쪼개질수록 확장성↑/ Disk, I/O 성능↑/ 조인 성능↓/ 관리 용이성↓

4. PK/FK 칼럼 순서 및 성능
- 일반적인 프로젝트에선 PK/FK 칼럼 순서의 중요성을 인지하지 못해 데이터 모델링 되어있는 상태로 DDL을 생성하여 성능이 저하되는 경우가 빈번

① 인덱스 중요성 : 데이터 조작 시 가장 효과적으로 처리될 수 있는 접근 경로 제공
* 인덱스의 특징 : 여러개의 속성이 하나의 인덱스로 구성되어 있을 때, 앞쪽에 위치한 속성의 값이 비교자로 있어야 인덱스가 좋은 효율을 나타낼 수 있다. 앞쪽에 위치한 속성값이 가급적 '=' 아니면 최소한의 범위인 'BETWEEN'이 들어와야 됨

② PK/FK 설계 중요성 : 데이터 접근 시 접근경로 제공, 설계단계 마지막에 칼럼 순서 조정
③ PK 순서의 중요성 : 물리적 모델링 단계에서 스스로 생성된 PK 외에 상속되는 PK 순서도 중요
④ FK 순서의 중요성 : 조인을 할 수 있는 수단이 됨(=경로), 조회 조건 고려해서 반드시 인덱스 생성

5. PK 순서를 조정하지 않으면 성능 저하 되는 이유
- 조회 조건(WHERE)에 따라 인덱스를 처리하는 범위가 달라짐
- PK의 순서를 인덱스 특징에 맞게 생성하지 않고 자동으로 생성하면, 테이블에 접근하는 트랜잭션이 인덱스 범위를 넓게 하거나 풀 스캔(full scan)을 유발

6. 물리적 테이블에 FK 제약이 걸려있지 않은 경우 인덱스 미생성으로 생긴 성능 저하
- 물리적으로 두 테이블 사이 FK 참조 무결성 관계를 걸어 상속받은 FK에 인덱스 생성

7. 인덱스 엑세스 범위 좁히는 가장 좋은 방법
- PK가 여러 개일 때, Where절에 사용하는 조건용 칼럼들이 우선순위가 되어야 함
- '=', EQUAL 조건, 동등 조건에 있는 칼럼이 제일 앞으로
- BETWEEN, IN 범위 조건에 있는 칼럼이 그 다음 순위
- 나머지 PK는 그 뒤에 아무렇게나

제6절 분산 데이터베이스와 성능

1. 분산 데이터베이스의 개념
- 물리적으로 분산된 데이터베이스를 하나의 논리적 시스템으로 사용
- 빠른 네트워크 환경을 이용하여 데이터베이스를 여러 지역에서 노드로 위치시켜 사용성과 성능을 극대화하는 데이터베이스

데이터베이스 분산 설계는 다음과 같은 경우에 적용하면 효과적이다.

- 성능이 중요한 사이트에 적용해야 한다.
- 공통코드, 기준정보, 마스터 데이터 등에 대해 분산환경을 구성하면 성능이 좋아진다.
- 실시간 동기화가 요구되지 않을 때 좋다. 거의 실시간(Near Real Time)의 업무적인 특징을 가지고 있을 때도 분산 환경을 구성할 수 있다
- 특정 서버에 부하가 집중이 될 때 부하를 분산할 때도 좋다.
- 백업 사이트(Disaster Recovery Site)를 구성할 때 간단하게 분산기능을 적용하여 구성할 수 있다.

2. 분산 데이터베이스 설계 방식
- 상향식 : 지역 스키마 작성 후 전역 스키마 작성
- 하향식 : 전역 스키마 작성 후 지역사상 스키마 작성

3. 분산 데이터베이스의 장/단점

장점	단점
- 지역 자치성 증가 - 점증적 시스템 용량 확장 가능 - 신뢰성 / 가용성 / 효용성 / 융통성 - 빠른 응답 속도, 통신비용 절감 - 데이터의 가용성과 신뢰성 증가 - 시스템 규모의 적절한 조절 가능 - 각 지역 사용자의 요구 수용 가능	- 소프트웨어 개발 비용 증가 - 오류의 잠재성 증대 - 처리 비용의 증대 - 설계, 관리의 복잡성 및 비용증가 - 불규칙한 응답 속도 - 통제의 어려움 - 데이터 무결성 유지의 어려움

4. 분산 DB의 투명성
- 분할 투명성 : 하나의 논리적 관계가 분할되어 각 단편의 사본이 여러 사이트에 저장
- 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 : 구성 요소(DBMS, 컴퓨터)의 장애에 무관하게 트랜잭션의 원자성이 유지됨
- 병행 투명성 : 다수의 트랜잭션을 동시 수행했을 때 결과의 일관성이 유지됨

5. 분산 데이터베이스의 적용 기법

① 테이블 위치 분산 : 설계된 테이블의 위치를 분산
- 테이블 구조 변경 X
- 테이블이 다른 데이터베이스에 중복으로 생성 X
- 정보를 이용하는 형태가 각 위치별로 차이가 있을 경우 사용
- 테이블 위치를 파악할 수 있는 도식화된 위치별 데이터베이스 문서 필요

② 테이블 분할 분산 (수평, 수직분할) : 테이블을 수평이나 수직으로 분할하여 분산
- 수평 분할 : 특정 칼럼의 값 기준으로 로우 단위 분리, 칼럼 분리 X
- 수직 분할 : 칼럼을 기준으로 칼럼 단위로 분리, 로우 분리 X

③ 테이블 복제 분산 (부분, 광역복제)
- 동일한 테이블을 다른 지역이나 서버에서 동시 생성, 원격지 조인을 내부조인으로 변경하여 성능 향상. 프로젝트에서 많이 사용되는 데이터베이스 분산 기법

* 부분복제
- 마스터 데이터베이스에서 테이블의 일부 내용만 다른 지역이나 서버에 위치
- 본사는 통합 테이블 관리, 각 지사에서는 지사에 해당하는 로우만 관리
- 실제로는 지사에서 먼저 데이터 발생 → 본사에서 전체 통합

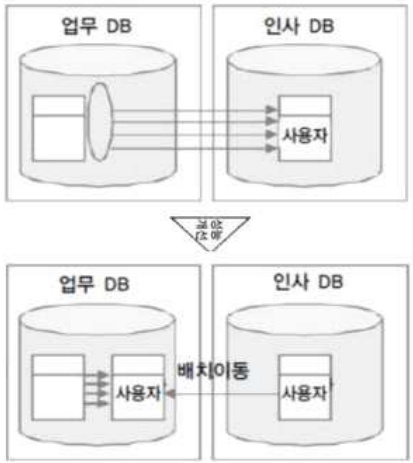
* 광역복제
- 통합된 테이블을 본사에 가지고 있으며 각 지사에 본사와 동일한 데이터 분배
- 동일한 테이블을 여러 곳에 복제하여 관리
- 본사에서 데이터의 입력, 수정, 삭제 발생 → 지사에서 이를 반영

④ 테이블 요약 분산 (분석, 통합요약)
- 유사한 내용의 데이터를 서로 다른 관점/수준에서 요약하여 분산 관리

* 분석요약 (rollup replication)
- 각 지사별 동일한 주제의 정보를 본사에서 통합하여 전체 요약 정보 산출

* 통합요약 (consolidation replication)
- 각 지사별로 존재하는 다른 내용 정보를 본사에 통합, 다시 전체의 요약 산출

[그림 Ⅰ-2-55] 업무 특성에 따른 분산환경 구성



과목2
1장 SQL 기본

제1절 관계형 데이터베이스 개요

1. 데이터베이스
- 데이터를 일정한 형태로 저장해 놓은 것
- 계층형 DB : 트리 형태의 자료구조에 데이터 저장 (1:N)
- 네트워크형 DB : 오너와 멤버 형태로 데이터 저장 (M:N)
- 관계형 DB : 집합 연산과 관계 연산 가능

2. 관계형 데이터베이스 (RDB : Relational Database)
- 파일 시스템 단점 : 동시에 삽입/수정/삭제 불가능해 데이터 관리가 어렵고 복사본 파일을 만들어 사용할 경우 데이터의 불일치성이 발생
- RDB 장점 : 정규화를 통해 이상 현상과 중복제거, 데이터 무결성 보장, 데이터 회복/복구 가능, 병행 제어, 동시성 관리를 통해 데이터 공유, 데이터 표현 방법 등 체계화

3. SQL 문장들의 종류
- SQL : 관계형 데이터베이스에서 데이터 정의/조작/제어를 위해 사용하는 언어

명령어의 종류	명령어	설명
데이터 조작어 (DML: Data Manipulation Language)	SELECT	데이터베이스에 들어 있는 데이터를 조회하거나 검색하기 위한 명령어를 말하는 것으로 RETRIEVE 라고도 한다.
	INSERT UPDATE DELETE	데이터베이스의 테이블에 들어 있는 데이터에 변형을 가하는 종류의 명령어들을 말한다. 예를 들어 데이터를 테이블에 새로운 행을 집어넣거나, 원하지 않는 데이터를 삭제하거나 수정하는 것들의 명령어들을 DML이라고 부른다.
데이터 정의어 (DDL: Data Definition Language)	CREATE ALTER DROP RENAME	테이블과 같은 데이터 구조를 정의하는데 사용되는 명령어들로 그러한 구조를 생성하거나 변경하거나 삭제하거나 이름을 바꾸는 데이터 구조와 관련된 명령어들을 DDL이라고 부른다.
데이터 제어어 (DCL: Data Control Language)	GRANT REVOKE	데이터베이스에 접근하고 객체들을 사용하도록 권한을 주고 회수하는 명령어를 DCL이라고 부른다.
트랜잭션 제어어 (TCL: Transaction Control Language)	COMMIT ROLLBACK	논리적인 작업의 단위를 묶어서 DML에 의해 조작된 결과를 작업단위(트랜잭션) 별로 제어하는 명령어를 말한다.

4. 테이블 : RDB의 기본 단위, 데이터를 저장하는 객체, 칼럼과 행의 2차원 구조

용어	설명
테이블 (Table)	행과 칼럼의 2차원 구조를 가진 데이터의 저장 장소이며, 데이터베이스의 가장 기본적인 개념
칼럼/열 (Column)	2차원 구조를 가진 테이블에서 세로 방향으로 이루어진 하나하나의 특정 속성 (더이상 나눌 수 없는 특성)
행 (Row)	2차원 구조를 가진 테이블에서 가로 방향으로 이루어진 연결된 데이터

5. ERD (Entitiy Relationship Diagram)
- 관계의 의미를 직관적으로 표현할 수 있는 수단
- 구성요소 : 엔터티(Entity), 관계(Relationship), 속성(Attribute) 3가지
- 표기법 : IE(Information Engineering) 표기법, Barker(Case Method) 표기법

제2절 DDL (CREATE, ALTER, DROP, RENAME)

1. 데이터의 유형

① 숫자 타입
- ANSI / ISO 기준 : Numeric, Decimal, Dec, Small Int, Integer, Int, Big int, Float, Real, Double Precision
- SQL Server / Sybase : 작은 정수, 정수, 큰 정수, 실수 등 + Money, Small Money
- Oracle : 숫자형 타입에 대해서 Number 한 가지 타입만 지원
- 벤더에서 ANSI/ISO 표준을 사용할 땐 기능을 중심으로 구현, 표준과 다른 용어 사용 허용

② 문자열 유형
- CHAR(고정길이) 과 VARCHAR(가변길이) 의 차이
- CHAR에서 문자열 비교 = 공백을 채워서 비교
- VARCHAR에서 비교 = 맨 처음부터 한 문자씩 비교 (공백도 하나의 문자로 취급)

③ 테이블의 칼럼이 가지고 있는 대표적인 4가지 유형

[표 Ⅱ-1-7] 자주 쓰이는 데이터 유형

데이터 유형	설명
CHARACTER(s)	- 고정 길이 문자열 정보 (Oracle, SQL Server 모두 CHAR로 표현) - s는 기본 길이 1바이트, 최대 길이 Oracle 2,000바이트, SQL Server 8,000바이트 - s만큼 최대 길이를 갖고 고정 길이를 가지고 있으므로 할당된 변수 값의 길이가 s보다 작을 경우에는 그 차이 길이만큼 공간으로 채워진다.
VARCHAR(s)	- CHARACTER VARYING의 약자로 가변 길이 문자열 정보 (Oracle은 VARCHAR2로 표현, SQL Server는 VARCHAR로 표현) - s는 최소 길이 1바이트, 최대 길이 Oracle 4,000바이트, SQL Server 8,000바이트 - s만큼의 최대 길이를 갖지만 가변 길이로 조정이 되기 때문에 할당된 변수값의 바이트만 적용된다. (Limit 개념)
NUMERIC	- 정수, 실수 등 숫자 정보 (Oracle은 NUMBER로, SQL Server는 10가지 이상의 숫자 타입을 가지고 있음) - Oracle은 처음에 전체 자리 수를 지정하고, 그 다음 소수 부분의 자리 수를 지정한다. 예를 들어, 정수 부분이 6자리이고 소수점 부분이 2자리인 경우에는 'NUMBER(8,2)'와 같이 된다.
DATETIME	- 날짜와 시각 정보 (Oracle은 DATE로 표현, SQL Server는 DATETIME으로 표현) - Oracle은 1초 단위, SQL Server는 3.33ms(millisecond) 단위 관리

2. CREATE TABLE – 테이블 생성

```
CREATE TABLE 테이블 이름
(칼럼명1 DATATYPE [DEFAULT 형식],
칼럼명1 DATATYPE [DEFAULT 형식],
칼럼명1 DATATYPE [DEFAULT 형식] );

CREATE TABLE PLAYER
( PLAYER_ID CHAR(7) NOT NULL, PLAYER_NAME VARCHAR(20) NOT NULL,
TEAM_ID CHAR(3) NOT NULL, E_PLAYER_NAME VARCHAR(40),
NICKNAME VARCHAR(30), JOIN_YYYY CHAR(4), POSITION VARCHAR(10),
BACK_NO TINYINT, NATION VARCHAR(20), BIRTH_DATE DATE, SOLAR CHAR(1),
HEIGHT SMALLINT, WEIGHT SMALLINT,
CONSTRAINT PLAYER_PK PRIMARY KEY (PLAYER_ID),
CONSTRAINT PLAYER_FK FOREIGN KEY (TEAM_ID) REFERENCES TEAM(TEAM_ID) );
```

- 명명 규칙 : 알파벳 → 숫자 → '_'(언더바) → '$'(달러) → '#'(샵)
- 테이블 생성시 대/소문자 구분은 하지 않는다. 기본적으로 테이블이나 칼럼명 = 대문자로
- DATETIME 데이터 유형에는 별도로 크기를 지정하지 않는다.
- 문자 데이터 유형은 반드시 가질 수 있는 최대 길이를 표시
- 칼럼과 칼럼의 구분은 콤마로 하되, 마지막 칼럼은 콤마 X
- 칼럼에 대한 제약조건이 있으면 CONSTRAINT를 이용하여 추가

3. 제약조건 (CONSTRAINT) – 데이터 무결성 유지 방법, 사용자가 원하는 조건의 데이터만 유지

- **PK (Primary Key)** : 한 테이블에 하나만 지정 가능 → 자동으로 unique 인덱스 생성, null 입력 불가 (기본키 제약 = 고유키 & not null 제약)

- **UNIQUE** : NULL 가능, 행을 고유하게 식별하기 위한 고유키
- **NOT NULL** : NULL 값 입력 금지
- **CHECK** : 데이터 무결성을 유지하기 위한 테이블의 특정 칼럼에 설정하는 제약, 입력할 수 있는 값의 범위 등을 제한, TRUE or FALSE 논리식을 지정
- **FK (Foreign Key)** : 참조 무결성 옵션 선택 가능, 여러개 가능

- **NULL** : 아직 정의되지 않은 값, 데이터 입력하지 못하는 경우
알 수 없는 값, 0과 공백(" ")은 NULL이 아님
NULL은 IS NULL, IS NOT NULL로만 비교 가능 (<>NULL !=NULL 사용 불가)

구분	설명
PRIMARY KEY (기본키)	테이블에 저장된 행 데이터를 고유하게 식별하기 위한 기본키를 정의한다. 하나의 테이블에 하나의 기본키 제약만 정의할 수 있다. 기본키 제약을 정의하면 DBMS는 자동으로 UNIQUE 인덱스를 생성하며, 기본키를 구성하는 칼럼에는 NULL을 입력할 수 없다. 결국 '기본키 제약 = 고유키 제약 & NOT NULL 제약'이 된다.
UNIQUE KEY (고유키)	테이블에 저장된 행 데이터를 고유하게 식별하기 위한 고유키를 정의한다. 단, NULL은 고유키 제약의 대상이 아니므로, NULL 값을 가진 행이 여러 개가 있더라도 고유키 제약 위반이 되지 않는다.
NOT NULL	NULL 값의 입력을 금지한다. 디폴트 상태에서는 모든 칼럼에서 NULL을 허가하고 있지만, 이 제약을 지정함으로써 해당 칼럼은 입력 필수가 된다. NOT NULL을 CHECK의 일부분으로 이해할 수도 있다.
CHECK	입력할 수 있는 값의 범위 등을 제한한다. CHECK 제약으로는 TRUE or FALSE로 평가할 수 있는 논리식을 지정한다.
FOREIGN KEY (외래키)	관계형 데이터베이스에서 테이블 간의 관계를 정의하기 위해 기본키를 다른 테이블의 외래키로 복사하는 경우 외래키가 생성된다. 외래키 지정시 참조 무결성 제약 옵션을 선택할 수 있다.

4. 생성된 테이블 구조 확인
[Oracle] DESCRIBE 테이블명; (간략히 DESC 테이블명;)
[SQL Server] sp_help 'dbo.테이블명'

5. SELECT 문장을 통한 테이블 생성 사례
① CREATE TABLE AS (CTAS)
- CREATE TABLE AS 문장을 사용하는 경우 테이블의 구조를 복사하므로 따로 작성 X
- 기존 테이블 제약조건 중 NOT NULL 제약 조건만 새로운 테이블에 복제됨
- 다른 제약조건(기본키, 고유키, 외래키, CHECK 등)은 복사가 되지 않아서 다시 적용해야 한다
- 계약 조건 추가하기 위해 ALTER TABLE 기능 사용
[Oracle] CREATE TABLE 테이블명B AS SELECT * FROM 테이블명A;
[예제] 선수(PLAYER) 테이블과 같은 내용으로 TEAM_TEMP라는 복사 테이블을 만들어 본다.
Oracle CREATE TABLE TEAM_TEMP AS SELECT * FROM TEAM;

② SQL Server에서는 Select ~ Into ~를 활용하여 같은 결과
[SQL SERVER] SELECT * INTO 테이블명B FROM 테이블명A;
SQL Server SELECT * INTO TEAM_TEMP FROM TEAM; (1개 행이 영향을 받음)

6. ALTER TABLE – 칼럼을 추가/삭제하거나 제약조건을 추가/삭제

```
ALTER TABLE 테이블이름 ADD 속성_이름 데이터타입 [DEFAULT]; //추가
ALTER TABLE 테이블이름 ALTER 속성_이름 [SET DEFAULT]; //속성명변경
ALTER TABLE 테이블이름 DROP 속성_이름 [CASCADE | RESTRICT]; //속성 삭제

ALTER TABLE PLAYER ADD ADDRESS VARCHAR(80);
ALTER TABLE PLAYER DROP COLUMN ADDRESS;
ALTER TABLE TEAM_TEMP ALTER COLUMN ORIG_YYYY VARCAHR(8) NOT NULL;
```

① ALTER TABLE + ADD COLUMN : 칼럼 추가
② ALTER TABLE + DROP COLUMN : 칼럼 삭제

③ ALTER TABLE + MODIFY : 칼럼 속성 변경
[Oracle] ALTER TABLE 테이블명 MODIFY (칼럼명 데이터유형 DEFAULT NOT NULL);
[SQL Server] ALTER TABLE 테이블명 ALTER COLUMN 칼럼명 데이터유형 DEFAULT NOT NULL;

④ ALTER TABLE + RENAME COLUMN A TO B : 컬럼명 변경
⑤ RENAME 변경전 테이블명 TO 변경후 테이블명 : 테이블명 변경

```
ALTER TABLE 테이블명
RENAME COLUMN 변경해야할 컬럼명 TO 새로운 컬럼명;

ALTER TABLE PLAYER
RENAME COLUMN PLAYER_ID TO TEAM_ID;
```

⑥ ALTER TABLE + ADD CONSTRAINT : 제약조건 추가
⑦ ALTER TABLE + DROP CONSTRAINT : 제약조건 삭제

* ALTER TABLE 칼럼 변경 주의사항 *
- 칼럼의 크기를 늘릴 수는 있지만 줄이기 X
- null만 있거나 행이 없는 경우에만 칼럼 폭 줄이기 가능
- null이 있을때는 데이터 유형(숫자, 문자) 변경 가능
- null이 없으면 not null 제약조건 추가 가능
- 기본값(DEFAULT) 변경 작업 이후 발생하는 행 삽입에 대해서만 기본값 변경

* PK 제약조건 생성하는 DDL
1) CREATE TABLE 테이블명
(칼럼1 VARCHAR2(10) PRIMARY KEY ,칼럼2 VARCHAR2(200) NOT NULL);

2) CREATE TABLE 테이블명
(칼럼1 VARCHAR2(10) NOT NULL, 칼럼2 VARCHAR2(200) NOT NULL,CONSTRAINT constraint_name PRIMARY KEY (칼럼1));

3) ALTER TABLE
테이블명 **ADD** CONSTRAINT constraint_name PRIMARY KEY (col_1, col_2,...)

*. 외래키(FK) 참고사항
- 테이블 생성 시 설정할 수 있다
- 외래키는 NULL값 가질 수 있다
- 한 테이블에 여러 개 존재 가능하다
- 참조 무결성 제약을 받는다

7. DROP TABLE – 테이블 제거

```
ALTER TABLE 테이블명 DROP COLUMN 삭제할 컬럼명;

DROP TABLE PLAYER;                          —>(테이블 전부 삭제, 회복 불가)
ALTER TABLE PLAYER DROP COLUMN ADDRESS;     —>(테이블 의 일부 칼럼 삭제, 회복 불가)
```

- DROP TABLE 테이블명 [CASCADE CONSTRAINT]; 제약조건명;

- DROP 명령어를 사용하면 테이블의 모든 데이터 및 구조를 삭제
- CASCADE CONSTRAINT 옵션은 해당 테이블과 관계가 있었던 참조되는 제약조건에 대해서도 삭제한다는 것을 의미
- SQL Server에서는 CASCADE 옵션이 존재하지 않으며 테이블을 삭제하기 전에 참조하는 FK(FOREIGN KEY) 제약 조건 또는 참조하는 테이블을 먼저 삭제

8. TRUNCATE TABLE – 테이블 비우기

```
TRUNCATE TABLE 테이블명 DROP COLUMN 삭제할 컬럼명;
```

- 테이블 삭제가 아닌 해당 테이블의 모든 행 제거 후 저장공간을 재사용하도록
- 구조 자체 삭제는 DROP COLUMN 삭제할 칼럼명

9. DDL과 DML의 삭제
- DDL은 반드시 AUTO COMMIT이 일어남 → DROP, TRUNCATE 원상복구 불가
- DML은 사용자가 COMMIT 해야함 → DELETE 테이블 삭제해도 ROLLBACK으로 복구 가능

10. 참조 동작
- Automatic : 자식 삽입 시 부모 테이블에 pk가 없으면 부모 pk 생성 후 자식에 삽입
- Dependent : 자식 삽입 시 부모 테이블에 pk가 존재할 때만 자식 삽입허용
- Cascade : 부모 삭제 시 자식 같이 삭제
- Restrict : 부모 삭제 시 자식 테이블에 pk가 없는 경우에만 부모 삭제 허용

제3절 DML (SELECT, INSERT, UPDATE, DELETE)

- DML(Data Manipulation Language) : 입력 / 수정 / 삭제 / 조회

1. DML
- 호스트 프로그램 속에 삽입되어 사용, 데이터 부속어라고도 한다
- Procedural DML : 절차적 데이터 조작어 : 초급언어, 사용자가 무슨 데이터를 원하고 어떻게 접근해 처리할 것인지 명세해야한다
- Nonprocedural DML (비절차적 데이터 조작어) : 고급언어, 사용자가 무슨 데이터를 원하는지만 명세하고 어떻게 접근할 것인지는 하지 않는다. 선언적 언어라고도 한다.

2. INSERT – 데이터 입력 방법

```
INSERT INTO 테이블명 (COLUMN_LIST)VALUES (COLUMN_LIST에 넣을 VALUE_LIST);
INSERT INTO 테이블명VALUES (전체 COLUMN에 넣을 VALUE_LIST);

PLAYER INSERT INTO PLAYER
(PLAYER_ID, PLAYER_NAME, TEAM_ID, POSITION, HEIGHT, WEIGHT, BACK_NO)   —>(칼럼 리스트)
VALUES ('2002007', '박지성', 'K07', 'MF', 178, 73, 7);                 —>(벨류 리스트)
```

- INSERT INTO 테이블명 (칼럼리스트 / 생략 = 전체칼럼)
  VALUES (컬럼명 순서에 맞춰 입력할 값 매핑해 작성);
- 데이터가 문자형일 경우 ''로 묶어서 입력

3. UPDATE 정보 수정

```
UPDATE 테이블명 SET 수정되어야 할 칼럼명 = 수정되기를 원하는 새로운 값;

UPDATE PLAYER SET BACK_NO = 99;
UPDATE PLAYER SET POSITION = 'MF';          —>(문자값인 경우 ' ' 사용)
```

- UPDATE 테이블명
  SET 칼럼명=값 WHERE 조건;

4. DELETE – 삭제

```
DELETE FROM 삭제를 원하는 정보가 들어있는 테이블명 WHERE 조건절;

DELETE FROM PLAYER;          —>(조건절이 없으면 전체 테이블 삭제)
```

- DELETE FROM 테이블명 WHERE 조건;
- FROM 문구는 생략이 가능
- WHERE 절을 사용하지 않는다면 테이블의 전체 데이터 삭제

5. SELECT – 데이터 조회

```
SELECT 칼럼명 FROM 테이블;

SELECT PLAYER_ID, PLAYER_NAME, TEAM_ID, POSITION FROM PLAYER;
SELECT DISTINCT POSITION FROM PLAYER;        —>(DISTICNT: 중복데이터를 1건으로 표시)
SELECT * FROM PLAYER;          —>(*: 모든 칼럼명 선택)
SELECT PLAYER_NAME AS 선수명 FROM PLAYER;     —>(AS: 칼럼명에 별명붙이고 별명으로 표시)
```

- SELECT [ALL | DISTINCT] 칼럼1, 칼럼2, ... FROM 테이블명;
- ALL : : DEFAULT 옵션 (중복 데이터 모두 출력)
- DISTINCT : 중복제거 하여 1건으로 출력

* 조회된 결과에 **별명(엘리아스 / ALIAS, ALIASES)**을 부여해서 칼럼 레이블 변경 가능
- 칼럼명 바로 뒤에 오고, 칼럼명과 ALIAS 사이에 **AS** 키워드 사용 가능 (선택)
- 공백/특수문자, 대소문자 구분이 필요할 경우, " 으로 묶어서 사용

6. 산술 연산자

( )	연산자 우선순위를 변경하기 위한 괄호 (괄호 안의 연산이 우선된다)
*	곱하기
/	나누기
+	더하기
-	빼기

- NUMBER와 DATE 자료형에 적용 (수학 사칙연산과 동일)

7. 합성 연산자 – 문자와 문자를 연결하는 합성(CONCATENATION) 연산자

```
[Oracle] → ||
[SQL Server] → +

Oracle SQL>>
SELECT PLAYER_NAME || '선수, ' || HEIGHT || 'cm,' || WEIGHT || 'kg' 체격정보 FROM PLAYER;
SQL>>
SELECT PLAYER_NAME +'선수, '+ HEIGHT +'cm, '+ WEIGHT +'kg'체격정보 FROM PLAYER;
>>> 정경량선수173cm,65kg 정은익선수,176cm,63kg 레오마르선수,183cm,77kg 명재용선수,173cm,63kg
```

제4절 TCL (COMMIT, ROLLBACK)

1. TCL
- 논리적 작업단위를 묶어 DML에 의해 조작된 결과를 작업단위 별로 제어
- 일부에서는 DCL로 분류하기도 한다

2. 트랜잭션
- **데이터베이스의 논리적인 연산 단위**
- 트랜잭션에는 하나 이상의 SQL 문장이 포함된다
- 트랜잭션은 밀접히 관련되어 분리될 수 없는 한 개 이상의 DB조작을 가리킴
- 분할할 수 없는 최소단위, 전부 적용하거나 전부 취소해야한다

3. 트랜잭션의 특성

특성	설명
원자성 (atomicity)	트랜잭션에서 정의된 연산들은 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 한다. (all or nothing)
일관성 (consistency)	트랜잭션이 실행되기 전의 데이터베이스 내용이 잘못 되어 있지 않다면 트랜잭션이 실행된 이후에도 데이터베이스의 내용에 잘못이 있으면 안 된다.
고립성 (isolation)	트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안 된다.
지속성 (durability)	트랜잭션이 성공적으로 수행되면 그 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장된다.

4. 트랜잭션을 컨트롤하는 TCL (TRANSACTION CONTROL LANGUAGE)
- 커밋 (COMMIT) : 올바르게 반영된 데이터를 데이터베이스에 반영시키는 것
- 롤백 (ROLLBACK) : 트랜잭션 시작 이전의 상태로 되돌리는 것
- 저장점 (SAVEPOINT) : 저장점 기능
- 잠금 (LOCKING) : 다른 트랜잭션이 동시에 접근하지 못하도록 제한

* 트랜잭션 대상이 되는 SQL
① UPDATE, INSERT, DELETE 등 데이터를 수정하는 DML 문
② SELECT FOR UPDATE 등 배타적 LOCK을 요구하는 SELECT 문

5. COMMIT – 입력/수정/삭제한 자료가 문제가 없을 경우 변경 사항 적용

```
Oracle SQL>>
UPDATE PLAYER SET HEIGHT = 100;
COMMIT;
SQL>>
UPDATE PLAYER SET HEIGHT = 100;
```

① COMMIT 이전 상태
- 단지 Memory Buffer에만 영향을 주고, 이전 상태로 복구 가능
- 현재 사용자는 SELECT 문으로 변경 결과를 확인 가능
- 다른 사용자는 현재 사용자가 수행한 결과의 확인 불가능
- 변경된 행은 아직 잠금(Locking) 설정되어 다른 사용자가 변경 불가능

② COMMIT 이후 상태
- 데이터에 대한 변경사항을 데이터베이스에 영구반영
- 이전 데이터는 영원히 잃어버림
- 모든 사용자가 결과 조회 가능
- 변경된 행은 잠금(Locking)이 해제되어 다른 사용자가 변경 가능

* Auto COMMIT
- [Oracle] 임의로 COMMIT 혹은 ROLLBACK을 수행해 주어야 트랜잭션이 종료
- [SQL Server] 기본적으로 AUTO COMMIT 모드, DML 구문이 성공이면 자동으로 COMMIT이 되고 오류가 발생할 경우 자동으로 ROLLBACK 처리

6. ROLLBACK – COMMIT 이전으로 되돌림

```
Oracle SQL>>
UPDATE PLAYER SET HEIGHT = 100;          ———>(480개의 행이 수정되었다.)
ROLLBACK;                                ———>(롤백이 완료되었다.)
SQL>>
BEGIN TRAN UPDATE PLAYER SET HEIGHT = 100;
ROLLBACK;                                ———>(롤백이 완료되었다.)
```

- 테이블에 입력/수정/삭제한 데이터에 대해 COMMIT 이전에 변경사항을 취소
- 이전 데이터가 다시 재저장됨
- 관련 행에 대한 잠금이 풀리고, 다른 사용자들이 데이터 변경 가능

7. COMMIT과 ROLLBACK을 사용함으로써 얻을 수 있는 효과
- 데이터 무결성 보장
- 영구적인 변경을 하기 전에 데이터의 변경사항을 확인 가능
- 논리적으로 연관된 작업을 그룹핑하여 처리 가능

8. SAVEPOINT – 저장점, 데이터 변경을 사전에 지정한 저장점까지만 롤백

```
[Oracle] SAVEPOINT 포인트이름;
  → ROLLBACK TO 포인트이름;

[SQL Server] SAVE TRANSACTION 포인트이름;
  → ROLLBACK TRANSACTION 포인트이름;
```

- 저장점을 정의하면 롤백을 할 경우 전체 롤백이 아닌 저장점까지의 일부만 롤백
- SAVEPOINT는 여러 개 지정할 수 있음
- 동일 이름으로 저장점 지정 시 가장 나중에 정의한 저장점이 유효
- point1으로 되돌리고 나면 그보다 미래인 point2로는 되돌릴 수 없다
- 저장점 없이 롤백하면 모든 변경사항을 취소

제5절 WHERE 절

1. WHERE
- 자신이 원하는 자료만을 검색하기 위해 이용
- WHERE 절에 조건이 없는 FTS(full table scan) 문장은 SQL 튜닝 1차 검토 대상
  (FTS가 무조건 나쁜건 아님 → 병렬 처리를 이용해 유용하게 사용가능)

```
SELECT [DISTINCT/ALL] 칼럼명 [ALIAS명] FROM 테이블명 WHERE 조건식;

SELECT PLAYER_NAME FROM PLAYER WHERE TEAM_ID = 'K02';          —>(비교 연산자)
SELECT PLAYER_NAME, POSITION, BACK_NO, HEIGHT FROM PLAYER WHERE HEIGHT >= 170;
```

2. WHERE 연산자의 종류

- **처리 순서** : 부정 연산자 → 비교 연산자 → 논리 연산자

구분	연산자	연산자의 의미
비교 연산자	=	같다.
	〉	보다 크다.
	〉=	보다 크거나 같다.
	〈	보다 작다.
	〈=	보다 작거나 같다.
SQL 연산자	BETWEEN a AND b	a와 b의 값 사이에 있으면 된다.(a와 b 값이 포함됨)
	IN (list)	리스트에 있는 값 중에서 어느 하나라도 일치하면 된다.
	LIKE '비교문자열'	비교문자열과 형태가 일치하면 된다.(%, _ 사용)
	IS NULL	NULL 값인 경우
논리 연산자	AND	앞에 있는 조건과 뒤에 오는 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 된다. 즉, 앞의 조건과 뒤의 조건을 동시에 만족해야 한다.
	OR	앞의 조건이 참(TRUE)이거나 뒤의 조건이 참(TRUE)이 되어야 결과도 참(TRUE)이 된다. 즉, 앞뒤의 조건 중 하나만 참(TRUE)이면 된다.
	NOT	뒤에 오는 조건에 반대되는 결과를 되돌려 준다.
부정 비교 연산자	!=	같지 않다.
	^=	같지 않다.
	◇	같지 않다.(ISO 표준, 모든 운영체제에서 사용 가능)
	NOT 칼럼명 =	~와 같지 않다.
	NOT 칼럼명 〉	~보다 크지 않다.
부정 SQL 연산자	NOT BETWEEN a AND b	a와 b의 값 사이에 있지 않다. (a, b 값을 포함하지 않는다)
	NOT IN (list)	list 값과 일치하지 않는다.
	IS NOT NULL	NULL 값을 갖지 않는다.

연산 우선순위	설 명
1	괄호 ()
2	NOT 연산자
3	비교 연산자, SQL 비교 연산자
4	AND
5	OR

* 문자열 비교 방법

구분	비교 방법
비교 연산자의 양쪽이 모두 CHAR 유형 타입인 경우	길이가 서로 다른 CHAR형 타입이면 작은 쪽에 SPACE를 추가하여 길이를 같게 한 후에 비교한다.
	서로 다른 문자가 나올 때까지 비교한다.
	달라진 첫 번째 문자의 값에 따라 크기를 결정한다.
	BLANK의 수만 다르다면 서로 같은 값으로 결정한다.
비교 연산자의 어느 한 쪽이 VARCHAR 유형 타입인 경우	서로 다른 문자가 나올 때까지 비교한다.
	길이가 다르다면 짧은 것이 끝날 때까지만 비교한 후에 길이가 긴 것이 크다고 판단한다.
	길이가 같고 다른 것이 없다면 같다고 판단한다.
	VARCHAR는 NOT NULL까지 길이를 말한다.
상수값과 비교할 경우	상수 쪽을 변수 타입과 동일하게 바꾸고 비교한다.
	변수 쪽이 CHAR 유형 타입이면 위의 CAHAR 유형 타입의 경우를 적용한다.
	변수 쪽이 VARCHAR 유형 타입이면 위의 VARCHAR 유형 타입의 경우를 적용한다.

3. IS NULL / IS NOT NULL

- NULL(ASCII 00) : 값이 존재하지 않는 것으로 확정되지 않은 값을 표현할 때 사용
- 어떤 값보다 크거나 작지도 않고 ' '(공백)이나 0(Zero)과 달리 비교 자체가 불가능한 값
- 어떤 값과 비교할 수도 없으며, 특정 값보다 크다, 적다라고 표현할 수 없다.

SELECT PLAYER_NAME 선수이름, POSITION 포지션, TEAM_ID FROM PLAYER WHERE POSITION IS NULL;

결과〉〉〉 선수이름 포지션 TEAM_ID ------ ----- ------- 정학범 K08 안익수 K08 차상광 K08

(포지션 데이블에서 포지션이 NULL값을 갖는 선수이름, 포지션, 팀ID를 출력하라!)

* **IS NULL** : NULL 값이면 True 아니면 False
- NULL 값과의 수치 연산은 NULL 값을 리턴
- NULL 값과의 비교 연산은 거짓(FALSE)을 리턴

* **IS NOT NULL** : NULL이 아닌 경우를 찾기 위해 사용
* NULL과 모든 사칙연산의 결과는 NULL 이다

4. ROWNUM / TOP – 행의 개수를 제한

① ROWNUM (Oracle)
- Oracle의 ROWNUM은 칼럼과 비슷한 성격의 Pseudo Column
- SQL 처리 결과 집합의 각 행에 대해 임시로 부여되는 일련번호
- 테이블/집합에서 원하는 만큼의 행만 가져올 때 WHERE 절에서 행의 개수를 제한
- 1건의 행은 [=] 연산자 사용 가능, 2건 이상부터는 [=] 사용 불가

```
Oracle SQL>>
"MY_TABLE 이라는 테이블의 첫번 째 칼럼을 '고유한 키'값 혹은 '인덱스 값'으로 설정하라!"
--> 새롭게 넘버링 칼럼을 설정하고, 그 값을 기 테이블의 '고유한 키'값 혹은 '인덱스 값'으로 설정
UPDATE MY_TABLE SET COLUMN1 = ROWNUM;

"PLAYER 테이블에서 PLAYER_NAME 번호가 3 이하인 선수 이름을 출력하라"
SELECT PLAYER_NAME FROM PLAYER WHERE ROWNUM <= 3;
```

[Oracle] SELECT 칼럼명 FROM 테이블명 ROWNUM <= N or ROWNUM < N; 고유키나 인덱스 생성 가능 → UPDATE 테이블명 SET 칼럼명 = ROWNUM

② TOP (SQL server)
- SQL Server는 TOP 절을 사용하여 결과 집합으로 출력되는 행의 수를 제한
- TOP (Expression) [PERCENT] [WITH TIES];
- Expression : 반환할 행의 수를 지정
- PERCENT : 쿼리 결과 집합에서 처음 Expression%의 행만 반환됨을 나타냄
- WITH TIES : ORDER BY 절이 지정된 경우만 사용 가능, 마지막 행 같은 값 추가 출력

```
SQL>>
TOP (Expression) [PERCENT] [WITH TIES];

"PLAYER 테이블에서 1~5행까지의 PLAYER_NAME 을 출력하라"
SELECT TOP(5) PLAYER_NAME FROM PLAYER;
```

[SQL Server] SELECT TOP(N) 칼럼명 FROM 테이블명;
→ TOP(Expression) / PERCENT / WITH TIES

제6절 함수

1. 내장함수 (Built-in Function) – SQL을 더욱 강력하게 해주고 데이터 값을 간편 조작하는데 사용
- 벤더에서 제공하는 함수인 내장 함수 (Built-In Function)
- 사용자가 정의할 수 있는 함수 (User Defined Function)

- SQL을 더욱 강력하게 해주고 데이터 값을 간편하게 조작하는데 사용
- 핵심적인 기능들은 이름/표기법이 달라도 대부분의 데이터베이스가 공통적으로 제공
- 내장함수는 다시 함수의 입력 값에 따라 단일행 함수 / 다중행 함수
- 함수는 **입력값이 아무리 많아도 출력값은 하나라는 M:1 관계**라는 중요한 특징을 가짐

- **단일행 함수** : 단일 행 내에 있는 하나의 값 또는 여러 값이 입력 인수로 사용
- **다중행 함수** : 여러 레코드의 값들을 입력 인수로 사용

① 단일행 함수

종류	내용	함수의 예
문자형 함수	문자를 입력하면 문자나 숫자 값을 반환한다.	LOWER, UPPER, SUBSTR/SUBSTRING, LENGTH/LEN, LTRIM, RTRIM, TRIM, ASCII,
숫자형 함수	숫자를 입력하면 숫자 값을 반환한다.	ABS, MOD, ROUND, TRUNC, SIGN, CHR/CHAR, CEIL/CEILING, FLOOR, EXP, LOG, LN, POWER, SIN, COS, TAN
날짜형 함수	DATE 타입의 값을 연산한다.	SYSDATE/GETDATE, EXTRACT/DATEPART, TO_NUMBER(TO_CHAR(d,'YYYY'|'MM'|'DD')) / YEAR|MONTH|DAY
변환형 함수	문자, 숫자, 날짜형 값의 데이터 타입을 변환한다.	TO_NUMBER, TO_CHAR, TO_DATE / CAST, CONVERT
NULL 관련 함수	NULL을 처리하기 위한 함수	NVL/ISNULL, NULLIF, COALESCE

※ 주: Oracle함수/SQL Server함수 표시, '/' 없는 것은 공통 함수

- 추출되는 각 행마다 작업을 수행
- 각 행마다 하나의 결과를 반환
- SELECT, WHERE, ORDER BY, UPDATE의 SET절에 사용가능
- 데이터 타입 변경 가능
- 중첩해서 사용 가능

② 다중 행 함수
- 여러 개의 행이 입력, 하나의 값 반환
- 그룹(집계) 함수가 다중 행 함수
- SUM, AVG, MAX, MIN, COUNT 등

2. 단일행 – **문자형 함수**

문자형 함수	함수 설명
LOWER(문자열)	문자열의 알파벳 문자를 소문자로 바꾸어 준다.
UPPER(문자열)	문자열의 알파벳 문자를 대문자로 바꾸어 준다.
ASCII(문자)	문자나 숫자를 ASCII 코드 번호로 바꾸어 준다.
CHR/CHAR(ASCII번호)	ASCII 코드 번호를 문자나 숫자로 바꾸어 준다.
CONCAT (문자열1, 문자열2)	Oracle, My SQL에서 유효한 함수이며 문자열1과 문자열2를 연결한다. 합성 연산자'||'(Oracle)나 '+'(SQL Server)와 동일하다.
SUBSTR/SUBSTRING (문자열, m[, n])	문자열 중 m위치에서 n개의 문자 길이에 해당하는 문자를 돌려준다. n이 생략되면 마지막 문자까지이다.
LENGTH/LEN(문자열)	문자열의 개수를 숫자값으로 돌려준다.
LTRIM (문자열 [, 지정문자])	문자열의 첫 문자부터 확인해서 지정 문자가 나타나면 해당 문자를 제거한다. (지정 문자가 생략되면 공백 값이 디폴트) SQL Server에서는 LTRIM 함수에 지정문자를 사용할 수 없다. 즉, 공백만 제거할 수 있다.
RTRIM (문자열 [, 지정문자])	문자열의 마지막 문자부터 확인해서 지정 문자가 나타나는 동안 해당 문자를 제거한다.(지정 문자가 생략되면 공백 값이 디폴트) SQL Server에서는 LTRIM 함수에 지정문자를 사용할 수 없다. 즉, 공백만 제거할 수 있다.
TRIM ([leading|trailing|both] 지정문자 FROM 문자열)	문자열에서 머리말, 꼬리말, 또는 양쪽에 있는 지정 문자를 제거한다. (leading | trailing | both 가 생략되면 both가 디폴트) SQL Server에서는 TRIM 함수에 지정문자를 사용할 수 없다. 즉, 공백만 제거할 수 있다.

"경기장의 지역번호와 전화번호를 합친 번호의 길이를 구하시오."

SELECT STADIUM_ID, DDD+TEL as TEL, LEN(DDD+TEL) as T_LEN FROM STADIUM;

—> (LEN -> 문자열의 갯수를 숫자값으로 리턴)

* 문자형 함수 적용되었을 때 리턴되는 값

문자형 함수 사용	결과 값 및 설명
LOWER('SQL Expert')	'sql expert'
UPPER('SQL Expert')	'SQL EXPERT'
ASCII('A')	65
CHR(65) / CHAR(65)	'A'
CONCAT('RDBMS',' SQL') 'RDBMS' || ' SQL' / 'RDBMS' + ' SQL'	'RDBMS SQL'
SUBSTR('SQL Expert', 5, 3) SUBSTRING('SQL Expert', 5, 3)	'Exp'
LENGTH('SQL Expert') / LEN('SQL Expert')	10
LTRIM('xxxYYZZxYZ','x') RTRIM('XXYYzzXYzz','z') TRIM('x' FROM 'xxYYZZxYZxx')	'YYZZxYZ' 'XXYYzzXY' 'YYZZxYZ'
RTRIM('XXYYZZXYZ ') → 공백 제거 및 CHAR와 VARCHAR 데이터 유형을 비교할 때 용이하게 사용된다.	'XXYYZZXYZ'

3. 단일행 – **숫자형 함수**

숫자형 함수	함수 설명
ABS(숫자)	숫자의 절대값을 돌려준다.
SIGN(숫자)	숫자가 양수인지, 음수인지 0인지를 구별한다.
MOD(숫자1, 숫자2)	숫자1을 숫자2로 나누어 나머지 값을 리턴한다. MOD 함수는 % 연산자로도 대체 가능함 (ex:7%3)
CEIL/CEILING(숫자)	숫자보다 크거나 같은 최소 정수를 리턴한다.
FLOOR(숫자)	숫자보다 작거나 같은 최대 정수를 리턴한다.
ROUND(숫자 [, m])	숫자를 소수점 m자리에서 반올림하여 리턴한다. m이 생략되면 디폴트 값은 0이다.
TRUNC(숫자 [, m])	숫자를 소수 m자리에서 잘라서 버린다. m이 생략되면 디폴트 값은 0이다. SQL SERVER에서 TRUNC 함수는 제공되지 않는다.
SIN, COS, TAN,...	숫자의 삼각함수 값을 리턴한다.
EXP(), POWER(), SQRT(), LOG(), LN()	숫자의 지수, 거듭 제곱, 제곱근, 자연 로그 값을 리턴한다.

"경기장의 지역번호와 전화번호를 합친 번호의 길이를 구하시오."

SELECT STADIUM_ID, DDD+TEL as TEL, LEN(DDD+TEL) as T_LEN FROM STADIUM;

—> (LEN -> 문자열의 갯수를 숫자값으로 리턴)

* 숫자형 함수 적용되었을 때 리턴되는 값

숫자형 함수 사용	결과 값 및 설명
ABS(-15)	15
SIGN(-20) SIGN(0) SIGN(+20)	-1 0 1
MOD(7,3) / 7%3	1
CEIL(38.123) / CEILING(38.123) CEILING(-38.123)	39 39 -38
FLOOR(38.123) FLOOR(-38.123)	38 -39
ROUND(38.5235, 3) ROUND(38.5235, 1) ROUND(38.5235, 0) ROUND(38.5235)	38.524 38.5 39 39 (인수 0이 Default)
TRUNC(38.5235, 3) TRUNC(38.5235, 1) TRUNC(38.5235, 0) TRUNC(38.5235)	38.523 38.5 38 38 (인수 0이 Default)

4. 단일행 – **날짜형 함수**

날짜형 함수	함수 설명
SYSDATE / GETDATE()	현재 날짜와 시각을 출력한다.
EXTRACT('YEAR'|'MONTH'|'DAY' from d) / DATEPART('YEAR'|'MONTH'|'DAY', *d*)	날짜 데이터에서 년/월/일 데이터를 출력할 수 있다. 시간/분/초도 가능함
TO_NUMBER(TO_CHAR(*d*,'YYYY')) / YEAR(*d*), TO_NUMBER(TO_CHAR(*d*,'MM')) / MONTH(*d*), TO_NUMBER(TO_CHAR(*d*,'DD')) / DAY(*d*)	날짜 데이터에서 년/월/일 데이터를 출력할 수 있다. Oracle EXTRACT YEAR/MONTH/DAY 옵션이나 SQL Server DEPART YEAR/MONTH/DAY 옵션과 같은 기능이다. TO_NUMBER 함수 제외시 문자형으로 출력됨

※ 주: Oracle함수/SQL Server함수 표시, '/' 없는 것은 공통 함수

연산	결과	설명
날짜 + 숫자	날짜	숫자만큼의 날수를 날짜에 더한다.
날짜 - 숫자	날짜	숫자만큼의 날수를 날짜에서 뺀다.
날짜1 - 날짜2	날짜수	다른 하나의 날짜에서 하나의 날짜를 빼면 일수가 나온다.
날짜 + 숫자/24	날짜	시간을 날짜에 더한다.

Oracle SQL>>

"사원(EMP) 테이블의 입사일자에서 년, 월, 일 데이터를 각각 출력한다."

EXTRACT(MONTH FROM HIREDATE) 입사월, EXTRACT(DAY FROM HIREDATE) 입사일 FROM EMP;

SQL>>

"사원(EMP) 테이블의 입사일자에서 년, 월, 일 데이터를 각각 출력한다."

SELECT ENAME, HIREDATE, DATEPART(YEAR, HIREDATE) 입사년도,

DATEPART(MONTH, HIREDATE) 입사월, DATEPART(DAY, HIREDATE) 입사일 FROM EMP;

(or 다음도 같은 코드)

SELECT ENAME, HIREDATE, YEAR(HIREDATE) 입사년도, MONTH(HIREDATE) 입사월,

DAY(HIREDATE) 입사일 FROM EMP;

5. 단일행 – **변환형 함수**

[표 Ⅱ-1-31] 데이터 유형 변환의 종류

종류	설명
명시적(Explicit) 데이터 유형 변환	데이터 변환형 함수로 데이터 유형을 변환하도록 명시해 주는 경우
암시적(Implicit) 데이터 유형 변환	데이터베이스가 자동으로 데이터 유형을 변환하여 계산하는 경우

[표 Ⅱ-1-32] 단일행 변환형 함수의 종류

변환형 함수 – Oracle	함수 설명
TO_NUMBER(문자열)	alphanumeric 문자열을 숫자로 변환한다.
TO_CHAR(숫자|날짜 [, FORMAT])	숫자나 날짜를 주어진 FORMAT 형태로 문자열 타입으로 변환한다.
TO_DATE(문자열 [, FORMAT])	문자열을 주어진 FORMAT 형태로 날짜 타입으로 변환한다.

변환형 함수 – SQL Server	함수 설명
CAST (expression AS data_type [(length)])	expression을 목표 데이터 유형으로 변환한다.
CONVERT (data_type [(length)], expression [, style])	expression을 목표 데이터 유형으로 변환한다.

6. CASE 표현 (IF-THEN-ELSE논리와 유사한 방식)

- CASE 표현은 IF-THEN-ELSE 논리와 유사한 방식으로 표현식을 작성
- SQL의 비교 연산 기능을 보완하는 역할
- ANSI/ISO 표준에는 CASE Expression이라고 표시.
- 함수와 같은 성격을 가지고 있으며 Oracle의 DECODE 함수와 같은 기능

* 단일행 CASE 표현의 종류

CASE 표현	함수 설명
CASE SIMPLE_CASE_EXPRESSION 조건 ELSE 표현절 END	SIMPLE_CASE_EXPRESSION 조건이 맞으면 SIMPLE_CASE_EXPRESSION 조건내의 THEN 절을 수행하고, 조건이 맞지 않으면 ELSE 절을 수행한다.
CASE SEARCHED_CASE_EXPRESSION 조건 ELSE 표현절 END	SEARCHED_CASE_EXPRESSION 조건이 맞으면 SEARCHED_CASE_EXPRESSION 조건내의 THEN 절을 수행하고, 조건이 맞지 않으면 ELSE 절을 수행한다.
DECODE(표현식, 기준값1, 값1 [, 기준값2, 값2, … , 디폴트값)	Oracle에서만 사용되는 함수로, 표현식의 값이 기준값1이면 값1을 출력하고, 기준값2이면 값2를 출력한다. 그리고 기준값이 없으면 디폴트 값을 출력한다. CASE 표현의 SIMPLE_CASE_EXPRESSION 조건과 동일하다.

```
SELECT 칼럼명,
        CASE
       WHEN 조건
        THEN 조건이 TRUE일 때 반환
         ELSE 조건이 FALSE일 때 반환
         END AS 칼럼명
FROM 테이블명;
```

SQL>>

"사원 정보에서 급여가 3000 이상이면 상등급으로, 1000 이상이면 중등급으로, 1000 미만이면 하등급으로 분류하라."

```
SELECT ENAME,
CASE WHEN SAL >= 3000 THEN 'HIGH' WHEN SAL >= 1000 THEN 'MID'
ELSE 'LOW' END
AS SALARY_GRADE FROM EMP;
```

7. NULL 관련 함수

① NVL / ISNULL 함수

- 표현식 1의 값이 NULL이면 표현식2 값을 출력한다.
- 결과값을 NULL이 아닌 다른 값을 얻고자 할 때 NVL/ISNULL 함수를 사용한다.
- NULL 값의 대상이 숫자 유형 데이터인 경우는 주로 0(Zero)으로,
- 문자 유형 데이터인 경우는 블랭크보다는 'x' 같이 시스템에서 의미 없는 문자로 바꾼다.

[표 Ⅱ-1-34] NULL 포함 연산의 결과

연산	연산의 결과
NULL + 2, 2 + NULL	NULL
NULL - 2, 2 - NULL	NULL
NULL * 2, 2 * NULL	NULL
NULL / 2, 2 / NULL	NULL

[표 Ⅱ-1-35] 단일행 NULL 관련 함수의 종류

일반형 함수	함수 설명
NVL(표현식1, 표현식2) / ISNULL(표현식1, 표현식2)	표현식1의 결과값이 NULL이면 표현식2의 값을 출력한다. 단, 표현식1과 표현식2의 결과 데이터 타입이 같아야 한다. NULL 관련 가장 많이 사용되는 함수이므로 상당히 중요하다.
NULLIF(표현식1, 표현식2)	표현식1이 표현식2와 같으면 NULL을, 같지 않으면 표현식1을 리턴한다.
COALESCE(표현식1, 표현식2, ……)	임의의 개수 표현식에서 NULL이 아닌 최초의 표현식을 나타낸다. 모든 표현식이 NULL이라면 NULL을 리턴한다.

※ 주: Oracle함수/SQL Server함수 표시, '/' 없는 것은 공통 함수

② NULL과 공집합

- SELECT 1 FROM DUAL WHERE 1 = 2; 와 같은 조건이 대표적인 공집합 발생 쿼리
- 조건에 맞는 데이터가 한 건도 없는 경우를 공집합
- NULL 데이터와는 또 다르게 이해해야 한다

③ NULLIF (표현식1, 표현식2)

- 표현식1이 표현식2와 같으면 NULL을, 같지 않으면 표현식1을 리턴

SQL>>

"사원 테이블에서 MGR와 7698이 같으면 NULL을 표시하고, 같지 않으면 MGR를 표시한다."

```
SELECT ENAME, EMPNO, MGR, NULLIF(MGR,7698) NUIF FROM EMP;
```

④ COALESCE (표현식1, 표현식2,...)
- 임의의 개수 표현식에서 NULL이 아닌 최초의 표현식
- COALESCE 함수는 인수의 숫자가 한정되어 있지 않음
- 임의의 개수 EXPR에서 NULL이 아닌 최초의 EXPR을 나타낸다.
- 만일 모든 EXPR이 NULL이라면 NULL을 리턴

SQL>>
"사원 테이블에서 커미션(COMM)을 1차 선택값으로, 급여(SAL)를 2차 선택값으로 선택하되 두 칼럼 모두 NULL인 경우는 NULL로 표시한다."
SELECT ENAME, COMM, SAL, COALESCE(COMM, SAL) COAL FROM EMP;

제7절 GROUP BY, HAVING 절

1. 집계함수
- 여러 행들의 그룹이 모여서 그룹 당 단 하나의 결과를 돌려주는 다중행 함수
- GROUP BY 절은 행들을 소그룹
- SELECT 절, HAVING 절, ORDER BY 절에 사용

- 집계함수명 (ALL | Distinct 칼럼)
- 주로 숫자형에서 사용, MIN MAX COUNT는 문자 날짜도 적용가능

[표 Ⅱ-1-36] 집계 함수의 종류

집계 함수	사용 목적
COUNT(*)	NULL 값을 포함한 행의 수를 출력한다.
COUNT(표현식)	표현식의 값이 NULL 값인 것을 제외한 행의 수를 출력한다.
SUM([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 합계를 출력한다.
AVG([DISTINCT | ALL] 표현식)	표현식의 NULL 값을 제외한 평균을 출력한다.
MAX([DISTINCT | ALL] 표현식)	표현식의 최대값을 출력한다. (문자,날짜 데이터 타입도 사용가능)
MIN([DISTINCT | ALL] 표현식)	표현식의 최소값을 출력한다. (문자,날짜 데이터 타입도 사용가능)
STDDEV([DISTINCT | ALL] 표현식)	표현식의 표준 편차를 출력한다.
VARIAN([DISTINCT | ALL] 표현식)	표현식의 분산을 출력한다.
기타 통계 함수	벤더별로 다양한 통계식을 제공한다.

3. GROUP BY 절
- FROM 절과 WHERE 절 뒤에 오며, 데이터들을 작은 그룹으로 분류하여 소그룹에 대한 항목별 통계 정보를 얻을 때 사용
- ROLLUP이나 CUBE에 의한 소계가 계산된 결과에는 GROUPING(EXPR)=1 이 표시
- 그 외 결과에는 GROUPING(EXPR)=0 이 표시

SQL>>
SELECT [DISTINCT] 칼럼명 [ALIAS명] FROM 테이블명 [WHERE 조건식]
[GROUP BY 칼럼(Column)이나 표현식] [HAVING 그룹조건식] ;

"K-리그 선수들의 포지션별 평균키는 어떻게 되는가?"
SELECT POSITION 포지션, COUNT(*) 인원수, COUNT(HEIGHT) 키대상, MAX(HEIGHT) 최대키, MIN(HEIGHT) 최소키, ROUND(AVG(HEIGHT),2) 평균키 FROM PLAYER GROUP BY POSITION;
결과 >>> 포지션 인원수 키대상 최대키 43 43 196 174 186.26 DF 172 142 190 170 180.21
FW 100 100 194 168 179.91 MF 162 162 189 165 176.31

4. HAVING 절

SQL>>
"HAVING 절을 이용해 평균키가 180 센티미터 이상인 정보만 표시"
SELECT POSITION 포지션, ROUND(AVG(HEIGHT),2) 평균키 FROM PLAYER
GROUP BY POSITION HAVING AVG(HEIGHT) >= 180;

- HAVING 절은 WHERE 절과 비슷하지만 그룹을 나타내는 결과 집합의 행에 조건이 적용

- WHERE 절에는 집계 함수를 사용할 수 없다
- GROUP BY 절보다 HAVING 절을 앞에 사용해도 같은 결과가 나오긴 하지만 논리적 순서를 지키는 것을 권고한다
- HAVING 절은 SELECT 절에 사용되지 않은 칼럼이나 집계함수가 아니더라도, GROUP BY 절의 기준 항목이나 소그룹의 집계 함수를 이용한 조건을 표시 가능
- WHERE 절 조건 변경은 대상 데이터 개수가 변경되므로 결과 데이터 값이 변경 가능성 있음
- HAVING 절 조건 변경은 결과 데이터 변경은 없고 출력되는 레코드 개수만 변경 가능성 있음

5. GROUP BY 절과 HAVING 절의 특성
- Group By에 의한 소그룹별 만들어진 집계 데이터 중 Having 조건 만족하는 내용만 출력
- 가능하면 Group By 하기 전에, Where 절로 계산 대상을 줄이는게 효과적
- Where 절은 전체 데이터를 Group으로 나누기 전에 필요없는 조건을 미리 제거하는 역할
- Having 절은 Group By로 만들어진 소그룹에 대해서만 조건임

6. CASE 표현을 활용한 월별 데이터 집계 집계 – 함수(CASE( ))~GROUP BY
: 모델링의 제1정규화로 인해 반복되는 칼럼의 경우 구분 칼럼을 두고 여러 개의 레코드로 만들어진 집합을, 정해진 칼럼 수만큼 확장해서 집계 보고서를 만드는 유용한 기법

7. 집계함수와 NULL 처리
: 리포트 출력 때 NULL이 아닌 0을 표시하고 싶은 경우에는 NVL(SUM(SAL),0) 이나, ISNULL(SUM(SAL),0) 처럼 전체 SUM의 결과가 NULL인 경우(대상 건수가 모두 NULL인 경우)에만 한 번 NVL/ISNULL 함수를 사용

제8절 ORDER BY절

1. ORDER BY 정렬
- SQL 문장으로 조회된 데이터들을 목적에 맞게 특정 칼럼을 기준으로 정렬하여 출력
- ORDER BY 절에 칼럼명 대신 SELECT 절에서 사용한 ALIAS 명, 칼럼 순서를 나타내는 정수도 사용이 가능
- 기본적으로 오름차순이며 SQL 문장이 제일 마지막에 위치

SQL>>
"선수 테이블에서 선수들의 이름, 포지션, 백넘버를 출력하는데 사람 이름을 내림차순(DESC)으로 정렬하여 출력
키가 NULL인 데이터는 제외"
SELECT PLAYER_NAME 선수명, POSITION 포지션, BACK_NO 백넘버 FROM PLAYER
WHERE BACK_NO IS NOT NULL ORDER BY PLAYER_NAME DESC;

- 숫자형 타입은 오름차순시 작은 값 / 날짜형 타입은 오름차순시 빠른 날부터 출력
- ORDER BY에는 GROUP BY 칼럼이나 SELECT의 칼럼만 올 수 있다
- Oracle에선 null을 가장 큰 값으로 SQL Server에선 null을 가장 작은 값으로 간주

2. SELECT 문장의 실행 순서

=> 1. 발췌 대상 테이블을 참조한다. (FROM) => 2. 발췌 대상 데이터가 아닌 것은 제거한다. (WHERE)
=> 3. 행들을 소그룹화 한다. (GROUP BY) => 4. 그룹핑된 값의 조건에 맞는 것만을 출력한다. (HAVING)
=> 5. 데이터 값을 출력/계산한다. (SELECT) 6. 데이터를 정렬한다. (ORDER BY)

5 SELECT 칼럼명 [ALIAS명]
1 FROM 테이블명
2 WHERE 조건식
3 GROUP BY 칼럼이나 표현식
4 HAVING 그룹조건식
6 ORDER BY칼럼이나 표현식;

3. TOP N 쿼리

① ROWNUM
- Oracle에서 순위가 높은 N개의 로우를 추출하기 위해 ORDER BY 절과 WHERE 절의 ROWNUM 조건을 같이 사용하는 경우, 이 두 조건으로는 원하는 결과를 얻을 수 없다.
- Oracle의 경우 정렬이 완료된 후 데이터의 일부가 출력되는 것이 아니라 데이터의 일부가 먼저 추출된 후(ORDER BY 절은 결과 집합을 결정하는데 관여하지 않음) 데이터에 대한 정렬작업 수행

② TOP ( )
- SQL Server는 TOP 조건을 사용하게 되면 별도 처리 없이 관련 ORDER BY 절의 데이터 정렬 후 원하는 일부 데이터만 쉽게 출력
- TOP 절을 사용하여 결과 집합으로 반환되는 행의 수를 제한

SQL>>
"사원 테이블에서 급여가 높은 2명을 내림차순으로 출력하는데 같은 급여를 받는 사원이 있으면 같이 출력한다."
SELECT TOP(2) WITH TIES ENAME, SAL FROM EMP ORDER BY SAL DESC;
결과 >>> KING 5000 SCOTT 3000 FORD 3000

제9절 조인(JOIN)

1. JOIN – **두 개 이상의 테이블들을 연결** / 결합하여 데이터를 출력
- JOIN은 관계형 데이터베이스의 가장 큰 장점이면서 대표적인 핵심 기능
- 일반적인 경우 행들은 PK나 FK 값의 연관에 의해 JOIN이 성립된다
- 어떤 경우에는 PK, FK 관계가 없어도 논리적인 값들의 연관만으로 JOIN 성립이 가능

- 하나의 SQL 문장에서 여러 테이블을 조인해서 사용할 수도 있다.
- FROM 절에 여러 테이블이 나열되더라도 SQL에서 데이터를 처리할 때는 두 개의 집합 간에만 JOIN이 일어난다.
- FROM 절에 A, B, C 3개의 테이블이 나열되었더라도 특정 2개의 테이블만 먼저 조인되고, 그 조인된 새로운 결과 집합과 남은 한 개의 테이블이 다음 차례로 조인

2. EQUI JOIN (등가 조인)
- **두 테이블의 칼럼 값이 정확하게 일치하는 경우**, 대부분 PK ↔ FK 관계 기반
- JOIN 조건은 WHERE 절에 기술

```
SELECT 테이블1.칼럼명, 테이블2.칼럼명,
FROM 테이블1, 테이블2
WHERE 테이블1.칼럼명1 = 테이블2.칼럼명2;   →  WHERE 절에 JOIN 조건을 넣는다.

SELECT 테이블1.칼럼명, 테이블2.칼럼명,
FROM 테이블1 INNER JOIN 테이블2 ON 테이블1.칼럼명1 = 테이블2.칼럼명2
→ ON 절에 JOIN 조건을 넣는다.
```

SQL>>
"선수 테이블과 팀 테이블에서 선수 이름과 소속된 팀의 이름을 출력하시오."

```
SELECT PLAYER.PLAYER_NAME 선수명, TEAM.TEAM_NAME 소속팀명 FROM PLAYER, TEAM
WHERE PLAYER.TEAM_ID = TEAM.TEAM_ID;
```

->(다음도 같은 코드)

```
SELECT PLAYER.PLAYER_NAME 선수명, TEAM.TEAM_NAME 소속팀명 FROM PLAYER
INNER JOIN TEAM ON PLAYER.TEAM_ID = TEAM.TEAM_ID;
```

* 조인 시 주의사항 *
: 조건 절에 테이블에 대한 ALIAS명을 적용하여 SQL 문장을 작성했을 경우, WHERE 절과 SELECT 절에는 테이블명이 아닌 ALIAS를 사용

3. Non EQUI JOIN (비등가 조인)
- **두 테이블의 칼럼 값이 정확하게 일치하지 않는 경우**
- Non EQUI JOIN의 경우에는 "=" 연산자가 아닌 다른 (Between, >, >=, <, <= 등) 연산자들을 사용하여 JOIN을 수행

```
SELECT 테이블1.칼럼명, 테이블2.칼럼명,
FROM 테이블1, 테이블2
WHERE 테이블1.칼럼명1 BETWEEN 테이블2.칼럼명1 AND 테이블2.칼럼명2;
```

SQL>>
"선수들 별로 홈그라운드 경기장이 어디인지를 출력하고 싶다고 했을 때,
선수 테이블과 운동장 테이블이 서로 관계가 없으므로 중간에 팀 테이블이라는 서로 연관관계가 있는 테이블을
추가해서 세 개의 테이블을 JOIN 해야만 원하는 데이터를 얻을 수 있다."

```
SELECT P.PLAYER_NAME 선수명, P.POSITION 포지션, T.REGION_NAME 연고지, T.TEAM_NAME 팀명,
S.STADIUM_NAME 구장명 FROM PLAYER P, TEAM T, STADIUM S
WHERE P.TEAM_ID = T.TEAM_ID AND T.STADIUM_ID = S.STADIUM_ID
ORDER BY 선수명;
```

->(다음도 같은 코드)

```
SELECT PLAYER.PLAYER_
SELECT P.PLAYER_NAME 선수명, P.POSITION 포지션, T.REGION_NAME 연고지, T.TEAM_NAME 팀명,
S.STADIUM_NAME 구장명 FROM PLAYER P INNER JOIN TEAM T ON P.TEAM_ID = T.TEAM_ID
INNER JOIN STADIUM S ON T.STADIUM_ID = S.STADIUM_ID ORDER BY 선수명;
```

2장 SQL 활용

제1절 표준 조인

1. STANDAR SQL (표준 SQL) 개요

① 표준 SQL의 기능
- STANDARD JOIN 기능 추가 (CROSS, OUTER JOIN 등 새로운 FROM 절 JOIN 기능들)
- SCALAR SUBQUERY, TOP-N QUERY 등의 새로운 서브쿼리 기능들
- ROLLUP, CUBE, GROUPING SETS 등의 새로운 리포팅 기능
- WINDOW FUNCTION 같은 새로운 개념의 분석 기능들

② 일반 집합 연산자 → 현재 SQL
- UNION 연산 → UNION 기능 : 합집합
- INTERSECTION 연산 → INTERSECT 기능으로: 교집합
- DIFFERENCE 연산 → EXCEPT 기능으로 (Oracle은 MINUS) : 차집합
- PRODUCT 연산 → CROSS JOIN 기능 : 곱집합 (생길 수 있는 모든 데이터 조합)

③ 순수 관계 연산자 → 현재 SQL
- SELECT 연산 → WHERE 절 : 조건에 맞는 행 조회
- PROJECT 연산 → SELECT 절 : 조건에 맞는 칼럼 조회
- (NATURAL) JOIN 연산 → 다양한 JOIN 기능 : 여러 조인 존재
- DIVIDE 연산은 현재 사용되지 않는다.

2. FROM 절의 JOIN 형태
- ANSI/ISO SQL에서 표시하는 FROM 절의 JOIN 형태
: INNER JOIN / NATURAL JOIN / USING 조건절 / ON 조건절 / CROSS JOIN / OUTER JOIN

- 기존 Where절 그대로 사용 가능
- FROM 절에서 JOIN 조건을 명시적으로 정의 가능

3. INNER JOIN - 내부 JOIN
- JOIN 조건에서 동일한 값이 있는 행만 반환
- DEFAULT 옵션이므로 생략이 가능하지만, CROSS JOIN / OUTER JOIN과는 같이 사용 X
- USING 조건절이나 ON 조건절을 필수적으로 사용
- 중복 테이블의 경우 별개의 칼럼으로 표시

```
SQL>>
"사원 번호와 사원 이름, 소속부서 코드와 소속부서 이름을 출력하시오."
SELECT EMP.DEPTNO, EMPNO, ENAME, DNAME FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
->(다음도 같은 코드)
SELECT EMP.DEPTNO, EMPNO, ENAME, DNAME FROM EMP
INNER JOIN DEPT ON EMP.DEPTNO = DEPT.DEPTNO;
->(다음도 같은 코드, INNER JOIN을 JOIN으로 써도 상관 없다. 디폴트값이 INNER JOIN)
SELECT EMP.DEPTNO, EMPNO, ENAME, DNAME FROM EMP
JOIN DEPT ON EMP.DEPTNO = DEPT.DEPTNO;
```

5. NATURAL JOIN
- 두 테이블간 동일한 이름을 갖는 모든 칼럼에 대해 EQUI JOIN을 수행
- USING, ON, WHERE에서 JOIN을 정의할 수 없다
- JOIN에 사용된 컬럼은 같은 데이터 타입이여야 함
- ALIAS나 접두사 붙일 수 없다

```
SQL>>
"사원 번호와 사원 이름, 소속부서 코드와 소속부서 이름을 출력하시오."
SELECT DEPTNO, EMPNO, ENAME, DNAME FROM EMP NATURAL JOIN DEPT;
```

6. USING 조건절
- FROM 절에 USING 조건절을 이용해서 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI JOIN을 할 수 있다
- SQL Server에서는 지원하지 않는다
- JOIN 칼럼에 대해서는 ALIAS나 테이블이름과 같은 접두사를 붙일 수 없다
- JOIN에 사용되는 칼럼은 1개만 표시한다

```
Oracle SQL>>
SELECT * FROM DEPT JOIN DEPT_TEMP USING (DEPTNO);
```

7. ON 조건절
- 칼럼명이 달라도 JOIN 사용가능
- WHERE 검색 조건은 충돌 없이 사용할 수 있다
- ON 조건절에서 사용된 괄호는 옵션사항이다
- **ALIAS 및 테이블명과 같은 접두사를 반드시 사용**

```
가. WHERE 절과의 혼용
SQL>>
"부서코드 30인 부서의 소속 사원 이름 및 소속 부서 코드, 부서 코드, 부서 이름을 출력하시오."
SELECT E.ENAME, E.DEPTNO, D.DEPTNO, D.DNAME FROM EMP E
JOIN DEPT D ON (E.DEPTNO = D.DEPTNO) WHERE E.DEPTNO = 30;
```

나. ON 조건절 + 데이터 검증 조건 추가

SQL>>

"매니저 사원번호가 7698번인 사원들의 이름 및 소속 부서 코드, 부서 이름을 출력하시오."

```
SELECT E.ENAME, E.MGR, D.DEPTNO, D.DNAME FROM EMP E
JOIN DEPT D ON (E.DEPTNO = D.DEPTNO AND E.MGR = 7698);
```

—>(다음도 같은 코드)

```
SELECT E.ENAME, E.MGR, D.DEPTNO, D.DNAME FROM EMP E
JOIN DEPT D ON (E.DEPTNO = D.DEPTNO) WHERE E.MGR = 7698;
```

다. ON 조건절 예제

SQL>>

"팀과 스타디움 테이블을 팀ID로 JOIN하여 팀이름, 팀ID, 스타디움 이름을 찾아본다.
STADIUM에는 팀ID가 HOMETEAM_ID라는 칼럼으로 표시되어 있다."

```
SELECT TEAM_NAME, TEAM_ID, STADIUM_NAME FROM TEAM
JOIN STADIUM ON TEAM.TEAM_ID = STADIUM.HOMETEAM_ID ORDER BY TEAM_ID;
```

라. 다중 테이블 JOIN

SQL>>

"사원과 DEPT 테이블의 소속 부서명, DEPT_TEMP 테이블의 바뀐 부서명 정보를 찾아본다."

```
SELECT E.EMPNO, D.DEPTNO, D.DNAME, T.DNAME New_DNAME FROM EMP E
JOIN DEPT D ON (E.DEPTNO = D.DEPTNO) JOIN DEPT_TEMP T ON (E.DEPTNO = T.DEPTNO);
```

8. CROSS JOIN (= CARTESIAN PRODUCT / CROSS PRODUCT)

- JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합
- JOIN 할 때 적절한 JOIN 조건 칼럼이 없는 경우 사용
- 생길 수 있는 모든 데이터 조합을 출력
- 결과는 양쪽 집합의 M*N 건의 데이터 조합 발생

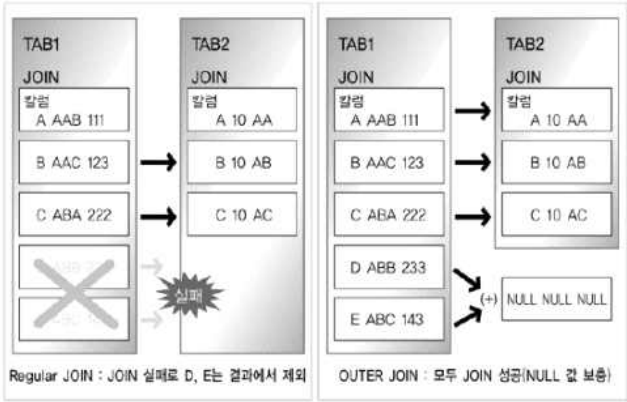
SQL>>

"사원 번호와 사원 이름, 소속부서 코드와 소속부서 이름을 찾아본다."

```
SELECT ENAME, DNAME FROM EMP CROSS JOIN DEPT ORDER BY ENAME;
```

9. OUTER JOIN

- JOIN 조건에서 동일한 값이 없는 행도(NULL도) 출력
- USING 조건절이나 ON 조건절을 필수로 사용
- IN/ ON 연산자 사용시 에러
- 표시가 누락된 칼럼이 있을 경우 OUTER JOIN 오류 발생, FULL OUTER JOIN 미지원
- FULL OUTER JOIN 미지원으로 인해 STANDART JOIN을 주로 사용



[그림 Ⅱ-2-3] OUTER JOIN 설명

① LEFT OUTER JOIN (↔ RIGHT)

- 좌측 테이블에서 먼저 데이터를 읽은 후, 우측 테이블에서 JOIN 대상을 읽음
- 좌측 테이블 기준이며, OUTER 키워드는 생략 가능

SQL>>

"STADIUM에 등록된 운동장 중에는 홈팀이 없는 경기장도 있다.
STADIUM과 TEAM을 JOIN 하되 홈팀이 없는 경기장의 정보도 같이 출력하도록 한다."

```
SELECT STADIUM_NAME, STADIUM.STADIUM_ID, SEAT_COUNT, HOMETEAM_ID, TEAM_NAME
FROM STADIUM LEFT OUTER JOIN TEAM ON STADIUM.HOMETEAM_ID = TEAM.TEAM_ID
ORDER BY HOMETEAM_ID;
```

—>(OUTER는 생략 가능)

```
* SELECT X.KEY1, Y.KEY2
FROM TAB1 X LEFT JOIN TAB2 Y
ON (X.KEY=Y.KEY)
```

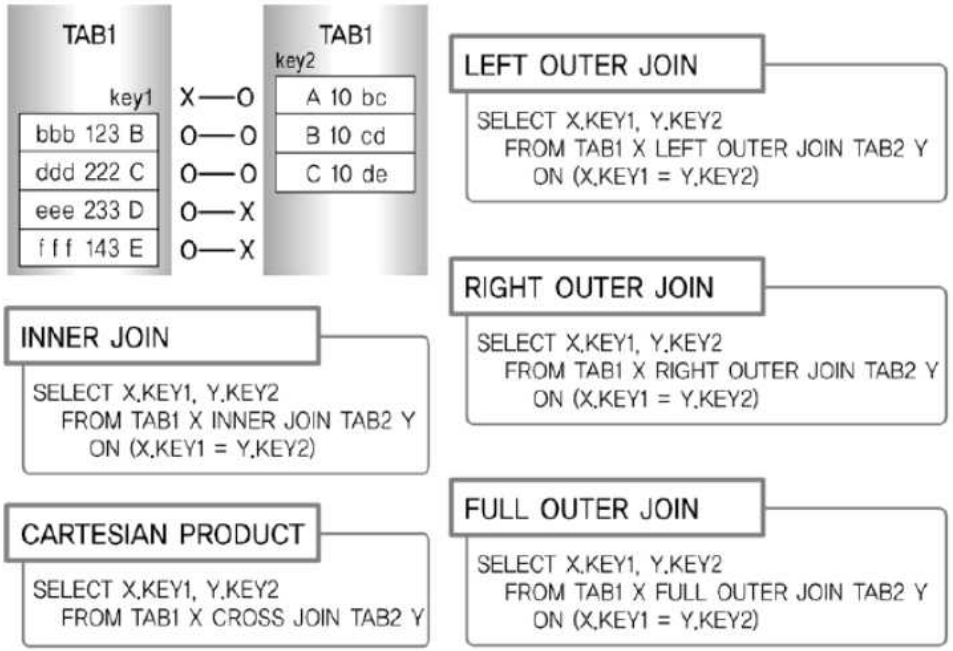
② RIGHT OUTER JOIN
- LEFT OUTER JOIN와 반대로 우측 테이블이 기준이 되어 결과 생성

③ FULL OUTER JOIN
- 합집합 개념으로 LEFT와 RIGHT를 모두 읽어 온다
- 조인이 되는 모든 테이블의 데이터를 읽어 JOIN 함

```
SQL>>
UPDATE DEPT_TEMP SET DEPTNO = DEPTNO + 20;          —>(먼저 업데이트)
SELECT * FROM DEPT_TEMP;                            —>(업데이트한 모두를 읽어온다)
```

```
* SELECT X.KEY1, Y.KEY2
FROM TAB1 X FULL JOIN TAB2 Y
ON (X.KEY=Y.KEY)
```

10. INNER JOIN vs OUTER JOIN vs CROSS JOIN 비교



[그림 Ⅱ-2-4] INNER vs OUTER vs CROSS JOIN 문장 비교

첫 번째, INNER JOIN의 결과는 다음과 같다.
양쪽 테이블에 모두 존재하는 키 값이 B-B, C-C 인 2건이 출력된다.

두 번째, LEFT OUTER JOIN의 결과는 다음과 같다.
TAB1을 기준으로 키 값 조합이 B-B, C-C, D-NULL, E-NULL 인 4건이 출력된다.

세 번째, RIGHT OUTER JOIN의 결과는 다음과 같다.
TAB2를 기준으로 키 값 조합이 NULL-A, B-B, C-C 인 3건이 출력된다.

네 번째, FULL OUTER JOIN의 결과는 다음과 같다.
양쪽 테이블을 기준으로 키 값 조합이 NULL-A, B-B, C-C, D-NULL, E-NULL 인 5건이 출력된다.

다섯 번째, CROSS JOIN의 결과는 다음과 같다. JOIN 가능한 모든 경우의 수를 표시하지만 단, OUTER JOIN은 제외한다.

양쪽 테이블 TAB1과 TAB2의 데이터를 곱한 개수인 4 * 3 = 12건이 추출됨
키 값 조합이
B-A, B-B, B-C, C-A, C-B, C-C, D-A, D-B, D-C, E-A, E-B, E-C 인 12건이 출력된다.

제2절 집합 연산자

1. 집합연산자
- 두 개 이상의 테이블에서 JOIN을 사용하지 않고, 연관된 데이터를 조회하는 방법
- 집합 연산자는 2개 이상의 질의 결과를 하나의 결과로 만든다
- SELECT절의 컬럼 수가 동일해야 하고, 동일 위치 데이터 타입이 상호 호환 가능해야 함

2. 집합연산자 종류

집합 연산자	연산자의 의미
UNION	여러 개의 SQL문의 결과에 대한 합집합으로 결과에서 모든 중복된 행은 하나의 행으로 만든다.
UNION ALL	여러 개의 SQL문의 결과에 대한 합집합으로 중복된 행도 그대로 결과로 표시된다. 즉, 단순히 결과만 합쳐놓은 것이다. 일반적으로 여러 질의 결과가 상호 배타적인(Exclusive)일 때 많이 사용한다. 개별 SQL문의 결과가 서로 중복되지 않는 경우, UNION과 결과가 동일하다. (결과의 정렬 순서에는 차이가 있을 수 있음)
INTERSECT	여러 개의 SQL문의 결과에 대한 교집합이다. 중복된 행은 하나의 행으로 만든다.
EXCEPT	앞의 SQL문의 결과에서 뒤의 SQL문의 결과에 대한 차집합이다. 중복된 행은 하나의 행으로 만든다. (일부 데이터베이스는 MINUS를 사용함)

제3절 계층형 질의와 셀프 조인

1. 계층형 질의
- 계층형 데이터 : 동일 테이블에 계층적으로 상/하위 데이터가 포함된 데이터
- 테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해 사용
- 엔터티를 순환관계 데이터 모델로 설계할 경우 계층형 데이터 발생 (조직, 사원, 메뉴 등)

① Oracle 계층형 질의

```
SELECT…
FROM 테이블
WHERE condition AND condition…
START WITH condition
CONNECT BY [NOCYCLE] condition AND condition…
[ORDER SIBLINGS BY column, column, …]
```

[표 Ⅱ-2-2] 계층형 질의에서 사용되는 가상 칼럼

가상 칼럼	설명
LEVEL	루트 데이터이면 1, 그 하위 데이터이면 2이다. 리프(Leaf) 데이터까지 1씩 증가한다.
CONNECT_BY_ISLEAF	전개 과정에서 해당 데이터가 리프 데이터이면 1, 그렇지 않으면 0이다.
CONNECT_BY_ISCYCLE	전개 과정에서 자식을 갖는데, 해당 데이터가 조상으로서 존재하면 1, 그렇지 않으면 0이다. 여기서 조상이란 자신으로부터 루트까지의 경로에 존재하는 데이터를 말한다. CYCLE 옵션을 사용했을 때만 사용할 수 있다.

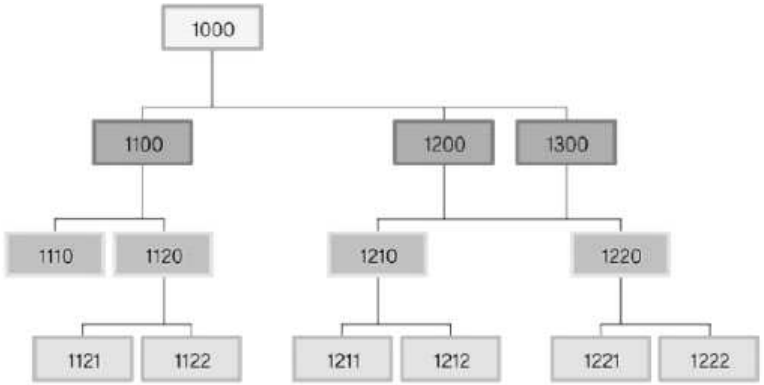
- START WITH절 : 레벨의 시작
- CONNECT BY절 : 그 다음에 자식 레벨 지정 (이때 CONNECT BY절의 조건 만족해야)
- PRIOR : CONNECT BY절에 사용되며, 현재 읽은 칼럼을 지정
- PRIOR 자식=부모 : [부모→자식] 으로 순방향 전개, 리프=1
- PRIOR 부모=자식 : [자식→부모] 으로 역방향 전개, 루트=1
- ORDER SIBLINGS BY : 형제 NODE 위치를 바꿈
- NOCYCLE : 이미 나타난 동일한 데이터가 전개 중에 다시 나타나면 이것을 CYCLE 형성이라 함. 사이클 발생한 데이터는 런타임 오류 발생 → NOCYCLE 추가 → 사이클 발생 이후 데이터는 전개 X

[표 Ⅱ-2-3] 계층형 질의에서 사용되는 함수

함수	설명
SYS_CONNECT_BY_PATH	루트 데이터부터 현재 전개할 데이터까지의 경로를 표시한다. 사용법 : SYS_CONNECT_BY_PATH(칼럼, 경로분리자)
CONNECT_BY_ROOT	현재 전개할 데이터의 루트 데이터를 표시한다. 단항 연산자이다. 사용법 : CONNECT_BY_ROOT 칼럼

② SQL Server 계층형 질의
- CTE(Common Table Expression)로 재귀 호출하여 상위부터 하위 방향 전개

```
WITH 테이블명_ANCHOR AS
( SELECT 하위칼럼명, 칼럼명, 상위칼럼명, 0 AS LEVEL
  FROM 테이블명
  WHERE 상위칼럼명 IS NULL /* 재귀 호출의 시작점 */
  UNION ALL
  SELECT R.칼럼명, R.칼럼명, R.계층칼럼명, A.LEVEL + 1
  FROM 테이블명_ANCHOR A, 테이블명 R
  WHERE A.하위칼럼 = R.상위칼럼 )
```



[그림 Ⅱ-2-10] 조직도 예제

3. 셀프 조인
- 동일 테이블 사이의 조인. 반드시 테이블 별칭(Alias)을 사용해야 한다.

```
SELECT ALIAS명1.칼럼명1, ALIAS명2.칼럼명1, ....
FROM 테이블명 ALIAS명1, 테이블명 ALIAS명2
WHERE ALIAS명1.칼럼명2 = ALIAS명2.칼럼명1
```

제4절 서브쿼리

1. 서브쿼리
- 하나의 SQL문 안의 SQL문
- 단일행 또는 복수행 비교 연산자와 함께 사용 가능
- 서브쿼리에선 ORDER BY 사용 불가 (메인쿼리의 마지막 부분에만 위치 가능)
- 서브쿼리는 메인쿼리의 테이블의 칼럼 사용 가능 (메인쿼리에선 서브쿼리의 칼럼 사용불가)

[그림 Ⅱ-2-12] 메인쿼리와 서브쿼리

* 서브쿼리가 SQL 문에서 사용 가능한 곳 *
- SELECT, FROM, WHERE, HAVING, ORDER BY절
- INSERT문의 VALUES절
- UPDATE문의 SET절
- DELECT문 사용 불가

2. 동작방식에 따른 분류

서브쿼리 종류	설명
Un-Correlated(비연관) 서브쿼리	서브쿼리가 메인쿼리 칼럼을 가지고 있지 않는 형태의 서브쿼리이다. 메인쿼리에 값(서브쿼리가 실행된 결과)을 제공하기 위한 목적으로 주로 사용한다.
Correlated(연관) 서브쿼리	서브쿼리가 메인쿼리 칼럼을 가지고 있는 형태의 서브쿼리이다. 일반적으로 메인쿼리가 먼저 수행되어 읽혀진 데이터를 서브쿼리에서 조건이 맞는지 확인하고자 할 때 주로 사용된다.

3. 반환되는 데이터 형태에 따른 분류

서브쿼리 종류	설명
Single Row 서브쿼리 (단일 행 서브쿼리)	서브쿼리의 실행 결과가 항상 1건 이하인 서브쿼리를 의미한다. 단일 행 서브쿼리는 단일 행 비교 연산자와 함께 사용된다. 단일 행 비교 연산자에는 =, <, <=, >, >=, ◇이 있다.
Multi Row 서브쿼리 (다중 행 서브쿼리)	서브쿼리의 실행 결과가 여러 건인 서브쿼리를 의미한다. 다중 행 서브쿼리는 다중 행 비교 연산자와 함께 사용된다. 다중 행 비교 연산자에는 IN, ALL, ANY, SOME, EXISTS가 있다.
Multi Column 서브쿼리 (다중 칼럼 서브쿼리)	서브쿼리의 실행 결과로 여러 칼럼을 반환한다. 메인쿼리의 조건절에 여러 칼럼을 동시에 비교할 수 있다. 서브쿼리와 메인쿼리에서 비교하고자 하는 칼럼 개수와 칼럼의 위치가 동일해야 한다.

* 다중 행 서브쿼리

[표 Ⅱ-2-6] 다중 행 비교 연산자

다중 행 연산자	설명
IN (서브쿼리)	서브쿼리의 결과에 존재하는 임의의 값과 동일한 조건을 의미한다. (Multiple OR 조건)
비교연산자 ALL (서브쿼리)	서브쿼리의 결과에 존재하는 모든 값을 만족하는 조건을 의미한다. 비교 연산자로 ">"를 사용했다면 메인쿼리는 서브쿼리의 모든 결과 값을 만족해야 하므로, 서브쿼리 결과의 최대값보다 큰 모든 건이 조건을 만족한다.
비교연산자 ANY (서브쿼리)	서브쿼리의 결과에 존재하는 어느 하나의 값이라도 만족하는 조건을 의미한다. 비교 연산자로 ">"를 사용했다면 메인쿼리는 서브쿼리의 값들 중 어떤 값이라도 만족하면 되므로, 서브쿼리의 결과의 최소값보다 큰 모든 건이 조건을 만족한다. (SOME은 ANY와 동일함)
EXISTS (서브쿼리)	서브쿼리의 결과를 만족하는 값이 존재하는지 여부를 확인하는조건을 의미한다. 조건을 [출처] 서브쿼리의 종류|작성자 찐 만족하는 건이 여러 건이더라도 1건만 찾으면 더 이상 검색하지 않는다.

★ 다중행 서브 쿼리 예제

SQL>>

"선수들 중에서 '정현수'라는 선수가 소속되어 있는 팀 정보를 출력하라!"

```
SELECT REGION_NAME 연고지명, TEAM_NAME 팀명, E_TEAM_NAME 영문팀명 FROM TEAM
WHERE TEAM_ID IN (SELECT TEAM_ID FROM PLAYER WHERE PLAYER_NAME = '정현수')
ORDER BY TEAM_NAME;
```

4. 그 밖의 위치에서 사용하는 서브쿼리

① SELECT 절에 서브쿼리 사용 → 스칼라 서브쿼리 (Scalar Subquery)
- 한 행, 한 칼럼만을 반환
- 값 하나를 반환하는 서브쿼리, SELECT절에 사용하는 서브쿼리
- 스칼라 서브쿼리 대신 JOIN으로 동일한 결과 추출 가능

② 뷰
- 가상의 테이블, FROM절에 사용하는 뷰는 인라인 뷰 라고 한다. 실제 데이터 X
- SQL이 실행될 때만 임시적으로 생성되는 동적 뷰 (일회성)
- 일반 뷰가 정적 뷰, 인라인 뷰는 동적 뷰

* 장점 *
- 독립성 : 테이블 구조 변경 자동 반영
- 편리성 : 쿼리를 단순하게 작성 가능
- 보안성 : 뷰를 생성할 때 칼럼을 제외할 수 있음

제5절 그룹함수

1. 데이터분석 개요
- ANSI/ISO SQL 표준은 데이터 분석을 위해서 다음 세 가지 함수를 정의
- AGGREGATE FUNCTION (집계), GROUP FUNCTION (그룹), WINDOW FUNCTION (윈도우)

2. ROLLUP 함수
- ROLLUP에 지정된 Grouping Columns의 List는 Subtotal을 생성하기 위해 사용
- Grouping Columns의 수를 N이라고 했을 때 N+1 Level의 Subtotal이 생성
- GROUP BY로 묶인 칼럼의 소계 계산, 계층 구조
- GROUP BY 칼럼 순서가 바뀌면 결과 값 바뀜
- GROUP BY의 확장된 형태

* GROUP BY ROLLUP(A) : 전체 합계, 칼럼 A소계
* GROUP BY ROLLUP(A,B) : 전체 합계, 칼럼 A소계, 칼럼 (A,B) 조합 소계
* GROUP BY ROLLUP(A,B,C) : 전체 함계, 칼럼 A소계, 칼럼 (A,B) 조합 소계, (A,B,C) 조합 소계
* GROUP BY ROLLUP(A,(B,C)) : 전체 합계, 칼럼 A소계, 칼럼 (A,(B,C)) 조합 소계
* GROUP BY A, ROLLUP(B) : A그룹별 집계, A그룹 내부에서 B칼럼별 집계

SQL>>
"부서명과 업무명을 기준으로 사원수와 급여 합을 집계한 일반적인 GROUP BY SQL 문장을 수행"

```
SELECT DNAME, JOB, COUNT(*) "Total Empl", SUM(SAL) "Total Sal" FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO GROUP BY DNAME, JOB;
```

3. CUBE 함수
- 결합 가능한 모든 값에 대한 다차원 집계

* GROUP BY CUBE(A) : 전체 합계, 칼럼 A소계
* GROUP BY CUBE(B) : 전체 합계, 칼럼 A소계, 칼럼 B소계, 칼럼 (A,B) 조합 소계

SQL>>
"부서명과 업무명을 기준으로 사원수와 급여 합을 집계한 일반적인 GROUP BY SQL 문장을 수행"

```
SELECT CASE GROUPING(DNAME) WHEN 1 THEN 'All Departments' ELSE DNAME END AS DNAME,
CASE GROUPING(JOB) WHEN 1 THEN 'All Jobs'
ELSE JOB END AS JOB, COUNT(*) "Total Empl", SUM(SAL) "Total Sal" FROM EMP, DEPT
WHERE DEPT.DEPTNO = EMP.DEPTNO GROUP BY CUBE (DNAME, JOB) ;
```

4. GROUPING SETS 함수
- 특정 항목에 대한 소계 계산, GROUP BY 칼럼 순서와 무관하게 개별적으로 처리
- 내가 보고싶은 것만 소계를 생성

* Group By Grouping Sets(A) : 컬럼 A소계
* Group By Grouping Sets(A, B) : 컬럼 A소계, 컬럼 B소계
* Group By Grouping Sets((A, B) : 컬럼 (A, B)소계

표현식	집계종류
ROLLUP(expr1, expr2)	expr1 + expr2
	expr1
	전체
GROUP BY expr1, ROLLUP(expr2, expr3)	expr1 + (expr2 + expr3)
	expr1 + (expr2)
	expr1
GROUP BY ROLLUP(expr1), expr2	exp2 + expr1
	expr2
CUBE(expr1, expr2)	expr1 + expr2
	expr1
	expr2
	전체
GROUP BY expr1, CUBE(expr2, expr3)	expr1 + (expr2 + expr3)
	expr1 + (expr2)
	expr1 + (expr3)
	expr1

제6절 윈도우 함수

1. 윈도우 함수 (WINDOW FUNCTION)
- 여러 행 간의 관계 정의 함수, 중첩 불가

2. 윈도우 함수(WINDOW FUNCTION)의 종류
① 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

SQL>>
"사원 데이터에서 급여가 높은 순서와 JOB 별로 급여가 높은 순서를 같이 출력한다."

```
SELECT JOB, ENAME, SAL, RANK( )
OVER (ORDER BY SAL DESC) ALL_RANK, RANK( )
OVER (PARTITION BY JOB ORDER BY SAL DESC) JOB_RANK FROM EMP;
```

② 윈도우 일반 집계 (AGGREGATE) 함수
- SUM(합), MAX(최대값), MIN(최소값), AVG(평균값) 등

③ 행 순서 함수
- FIRST_VALUE / LAST_VALUE 함수 : 첫 값 / 끝 값
- **LAG / LEAD** : 이전 값 / 이후 값
- LEAD(E,A)는 E에서 A번째 행의 값을 호출하는 형태로도 쓰임 (A의 기본값은 1)

④ 비율 함수
- RATIO_TO_REPORT : 전체 SUM(칼럼)값에 대한 행별 칼럼 값의 백분율을 소수점 반환
- PERCENT_RANK : 제일 먼저 나오는 것 0, 제일 늦게 나오는 것 1, 행의 순서별 백분율
- CUME_DIST : 전체건수에서 현재 행보다 작거나 같은 건수에 대한 누적백분율
- NTILE : 전체 건수를 ARGUMENT 값으로 N 등분한 결과

제7절 DCL

1. DCL
- 유저를 생성하거나 권한을 제어하는 명령어, 보안을 위해 필요
- GRANT : 권한 부여 (SQL → GRANT 권한 ON 오브젝트 TO 유저명;)
- REVOKE : 권한 제거 (SQL → REVOKE 권한 ON 오브젝트 TO 유저명;)

2. 권한
- SELECT, INSERT, UPDATE, DELETE, ALTER, ALL : DML 관련 권한
- REFERENCES : 지정된 테이블을 참조하는 제약조건을 생성하는 권한
- INDEX : 지정된 테이블에서 인덱스를 생성하는 권한

3. Oracle 기본 유저 종류

유저	역할
SCOTT	Oracle 테스트용 샘플 유저 Default 패스워드 : TIGER
SYS	DBA ROLE을 부여받은 유저
SYSTEM	데이터베이스의 모든 시스템 권한을 부여받은 DBA 유저 Oracle 설치 완료 시에 패스워드 설정

4. ROLE을 이용한 권한 부여
- 많은 데이터베이스에서 유저들과 권한들 사이에서 중개 역할을 하는 ROLE을 제공
- 권한의 집합, 권한을 일일이 부여하지 않고 ROLE로 편리하게 여러 권한을 부여
- 시스템 권한, 오브젝트 권한 모두 부여 가능
- ROLE은 유저에게 직접 부여하거나 다른 ROLE에 포함되어 유저에게 부여될 수 있다

[Oracle]
```
CONN SYSTEM/MANAGER
CREATE ROLE 롤이름;
GRANT 부여권한, 부여권한, ..... TO 롤이름;
GRANT 롤이름 TO 유저명;
```

5. Oracle에서 제공하는 ROLE 종류
- CONNECT : CREATE SESSION과 같은 로그인 권한
- RESOURCE : CREATE TABLE과 같은 오브젝트(리소스) 생성 권한

6. 유저 삭제 명령어와 권한
- DROP USER 유저명 CASCADE;
- CASCADE 옵션은 해당 유저가 생성한 오브젝트를 먼저 삭제 후 유저를 삭제

7. SQL Server에서 데이터베이스 수준 역할명

[표 Ⅱ-2-12] 서버 수준 역할명 (SQL Server 사례)

서버 수준 역할명	설명
public	모든 SQL Server 로그인은 PUBLIC 서버 역할에 속한다. 서버 보안 주체에게 보안 객체에 대한 특정 사용 권한이 부여되지 않았거나 거부된 경우 사용자는 해당 개체에 대해 PUBLIC 으로 부여된 사용 권한을 상속 받는다. 모든 사용자가 개체를 사용할 수 있도록 하려는 경우에만 개체에 PUBLIC 권한을 할당해야 한다.
bulkadmin	BULK INSERT문을 수행할 수 있다.
dbcreator	데이터베이스를 생성, 변경, 삭제 및 복원할 수 있다.
diskadmin	디스크 파일을 관리하는데 사용된다.
processadmin	SQL Server의 인스턴스에서 실행중인 프로세스를 종료할 수 있다.
securityadmin	로그인 및 해당 속성을 관리한다. 서버 및 데이터베이스 수준의 사용 권한을 부여(GRANT), 거부(DENY), 취소(REVOKE)할 수 있다. 또한, 로그인의 암호를 다시 설정할 수 있다.
serveradmin	서버 차원의 구성 옵션을 변경하고 서버를 종료할 수 있다.
setupadmin	연결된 서버를 추가하거나 제거할 수 있다.
sysadmin	서버에서 모든 작업을 수행할 수 있다. 기본적으로 Windows BUILTIN ₩Administrators 그룹의 멤버인 로컬 관리자 그룹은 sysamdin 고정 서버 역할의 멤버이다.

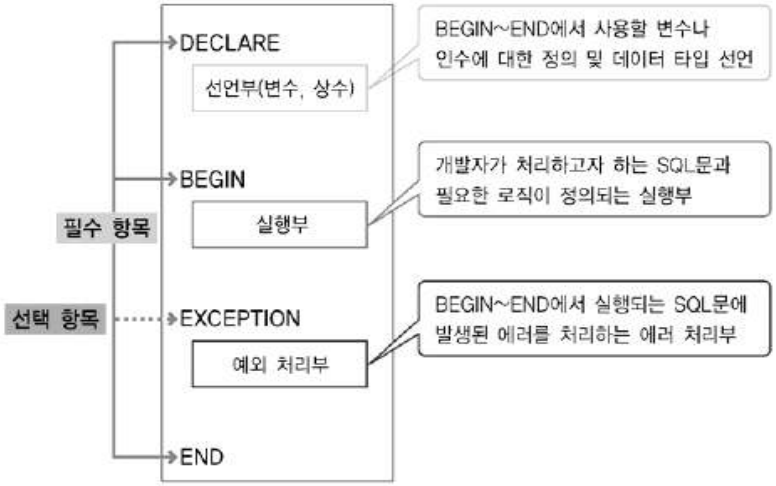
제8절 절차형 SQL

1. 절차형 SQL

- 일반적인 개발 언어처럼 절차 지향적인 프로그램을 작성할 수 있도록 DBMS 벤더별로 절차별 SQL 제공
- SQL문의 연속적인 실행이나 조건에 따른 분기처리를 이용하여 특정 기능을 수행하는 저장 모듈을 생성

2. PL/SQL (Oracle)

- Oracle의 PL/SQL은 Block 구조로 되어있고 Block 내에는 DML 문장과 QUERY 문장, 그리고 절차형 언어(IF, LOOP) 등을 사용할 수 있음
- 절차적 프로그래밍을 가능하게 하는 트랜잭션 언어



[그림 Ⅱ-2-19] PL/SQL 블록 구조

- PL/SQL 기본 문법

```
CREATE OR REPLACE Procedure 프로시저명 (argument1 mode data_type1, ... ) IS AS .....
BEGIN .....
EXCEPTION .....
END; /
```

- 생성된 프로시저 삭제

```
DROP Procedure 프로시저명;
```

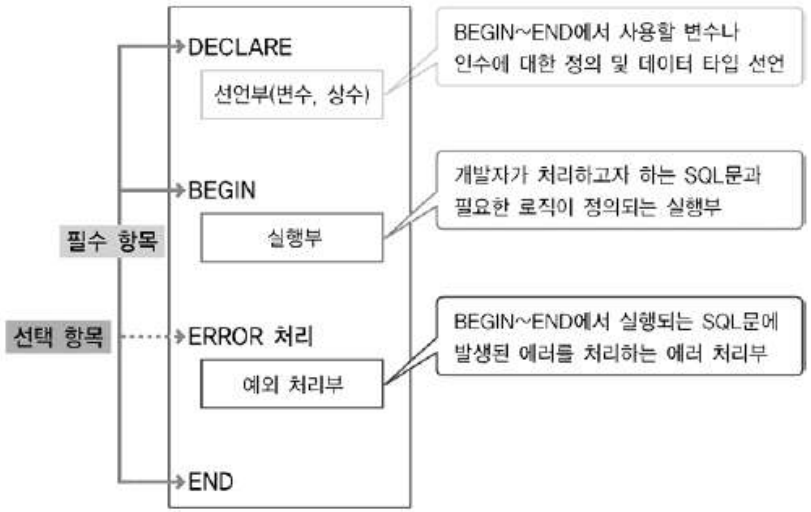
3. T-SQL (SQL Server)

- T-SQL은 근본적으로 SQL Server를 제어하기 위한 언어
- MS사에서 ANSI/ISO 표준의 SQL에 약간의 기능을 더 추가해 보완

① 기능

- 변수 선언 기능 : @@ - 전역변수(시스템 함수), @ - 지역변수
- 데이터 타입을 제공 (INT, FLOAT, VARCHAR 등의 자료형)
- 산술연산자, 비교연산자, 논리연산자 사용이 가능
- 흐름 제어 기능 : IF-ELSE 와 WHILE, CASE-THEN 사용이 가능하다.
- 주석 기능 : 한줄 주석은 -- 뒤의 내용 주석, 범위 주석은 /* 내용 */ 형태를 사용하며 여러 줄도 가능

② T-SQL 구조 (PL/SQL와 유사)



[그림 Ⅱ-2-20] T-SQL 구조

③ T-SQL 기본 문법

```
CREATE Procedure 스키마명.프로시저명 @parameter1 data_type1 mode, .....
WITH AS .....
BEGIN .....
ERROR 처리 .....
END;

DROP Procedure 스키마명.프로시저명;
```

3. 프로시저 (Procedure)
- 주로 DML을 사용해 주기적으로 진행해야 되는 작업을 저장
- 별도의 호출을 통해 실행
- CREATE OR REPLACE PROCEDURE 문으로 프로시저를 생성

* OR REPLACE는 기존에 같은 이름의 프로시저가 있으면 새로운 내용으로 덮어쓴다
- 작업의 결과를 DB에 저장(트랜잭션 처리)
- BEGIN~END문 사이에 작업 영역 생성 (1~3은 섞어서 사용하고, 4번만 마지막에 작성)
→ 1) 조건/반복 영역
→ 2) SQL을 사용해 데이터 관리하는 영역
→ 3) 예외 처리 영역
→ 4) 트랜잭션 영역 (작업 결과를 실제로 반영하거나 취소하는 영역)

4. 사용자 정의 함수(User Defined Function)
- 절차형 SQL을 로직과 함께 DB내에 저장해 놓은 명령문 집합
- RETURN을 통해 반드시 하나의 값 반환 (↔ 프로시저는 DB에 저장)

5. 트리거 (Trigger)
- DML문이 수행되었을 때 자동으로 동작하는 프로그램(↔ 프로시저는 EXCUTE로 실행)
- DCL과 TCL사용불가 (↔ 프로시저는 사용 가능)
- 데이터의 무결성과 일관성을 위해서 사용
- Trigger는 데이터베이스 보안의 적용, 유효하지 않은 트랜잭션의 예방, 업무 규칙 자동 적용 제공 등에 사용

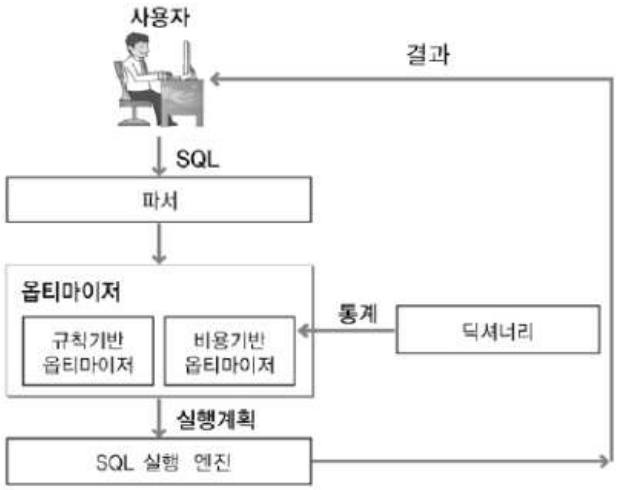
6. 프로시저와 트리거의 차이
- 프로시저는 BEGIN~END절 내에 COMMIT, ROLLBACK과 같은 트랜잭션 종료 명령어 사용
- 데이터베이스 트리거는 BEGIN ~ END 절 내에 사용할 수 없다

[표 Ⅱ-2-19] 프로시저와 트리거의 차이점

프로시저	트리거
CREATE Procedure 문법사용	CREATE Trigger 문법사용
EXECUTE 명령어로 실행	생성 후 자동으로 실행
COMMIT, ROLLBACK 실행 가능	COMMIT, ROLLBACK 실행 안됨

제1절 옵티마이저와 실행계획

1. 옵티마이저
- SQL문에 대한 최적의 실행방법을 결정하여 실행 계획 도출
- 최적의 실행방법, 실행계획을 짠다



[그림 Ⅱ-3-1] 옵티마이저의 종류

* 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

2. 규칙기반 옵티마이저 – 규칙(우선순위)를 가지고 실행계획 생성
- 인덱스를 이용한 엑세스 방식이 전체 테이블 액세스 방식보다 우선 순위가 높음
→ 이용 가능한 인덱스가 존재하면 전체 테이블 엑세스 방식보다 항상 인덱스를 사용하는 실행계획을 생성

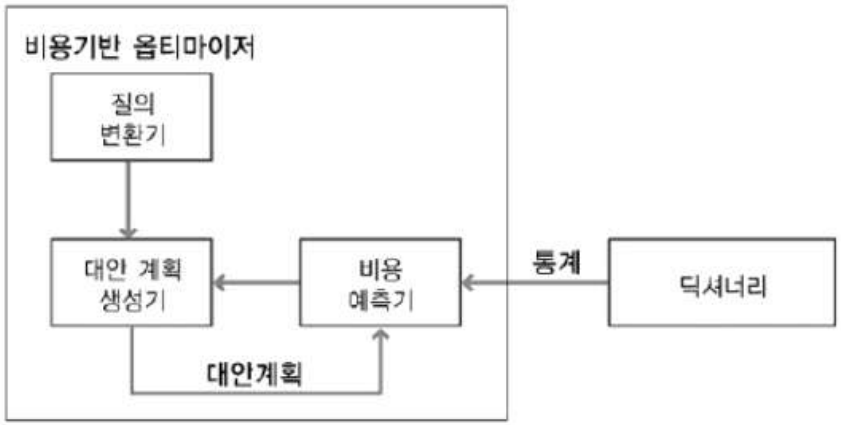
- 조인 칼럼에 대한 인덱스가 양쪽에 존재 : 우선순위가 높은 테이블이 선행(Driving)
- 한쪽에만 인덱스 존재 : 인덱스 없는 테이블이 선행 (NL Join 사용)

- 모두 인덱스 존재 X : FROM 절의 뒤에 나열된 테이블이 선행 (Sort Merge Join 사용)
- 우선순위가 동일 : FROM절에 나열된 테이블의 역순으로 선행 테이블 선택

3. 비용기반 옵티마이저 – SQL문을 처리하는데 필요한 비용이 가장 적은 실행계획을 선택

① 특징

처리 비용이 가장 적은 실행계획 선택, 데이터 딕셔너리의 통계정보나 DBMS의 차이로 같은 쿼리도 다른 실행계획이 생성될 수 있음, 실행계획의 예측 및 제어가 어려움



[그림 Ⅱ-3-3] 비용기반 옵티마이저의 구성 요소

② 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 계획 생성 모듈

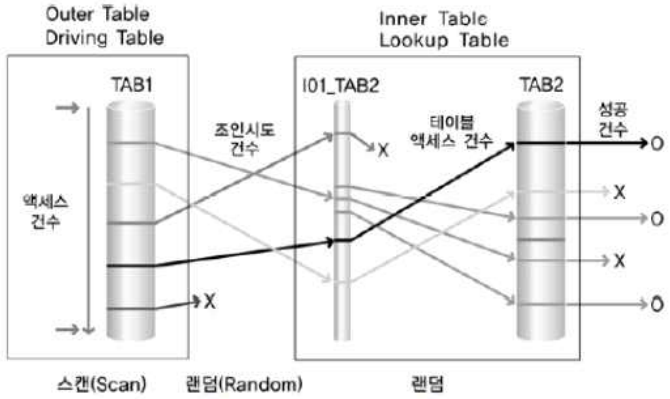
→ 1) 연산적용순서 2) 연산방법 3) 조인순서변경을 통해 대안 계획 생성

4. 옵티마이저 실행계획

- SQL에서 요구한 사항을 처리하기 위한 절차와 방법
- 다양한 실행계획(처리방법)마다 성능(실행시간)은 서로 다를 수 있다. (옵티마이저는 최적의 실행계획 생성)
- 구성 요소 : 조인순서, 조인기법, 액세스기법, 최적화정보, 연산 등

5. SQL 처리 흐름도

- SQL 내부적 처리 절차를 시각적으로 표현한 도표 (실행시간을 알 수 없다)
- 인덱스 스캔, 테이블 전체 스캔 등과 같은 엑세스기법을 표현
- 성능적인 측면도 표현 가능



[그림 Ⅱ-3-5] SQL 처리 흐름도

제2절 인덱스 기본

1. 인덱스

- 검색 조건에 부합하는 데이터를 효과적으로 검색할 수 있도록 돕는 기능 (찾아보기)
- 인덱스키로 정렬되어 있어 조회속도가 빠름, DML 작업 효율은 저하
- 주로 WHERE절 ORDER BY절에서 자주 쓰이는 칼럼을 인덱스로 지정

2. BLOCK의 개념

- 블록은 데이터가 저장되는 최소 단위, 테이블의 데이터들이 행 단위로 저장되어 있음
- 인덱스는 해당 테이블의 블록에 주소를 가지고 있음

3. 랜덤 액세스

- 인덱스를 스캔하여 테이블로 데이터를 찾아가는 방식
- 많은 양의 데이터를 읽을 경우 인덱스 스캔보다 FULL스캔이 더 효율적

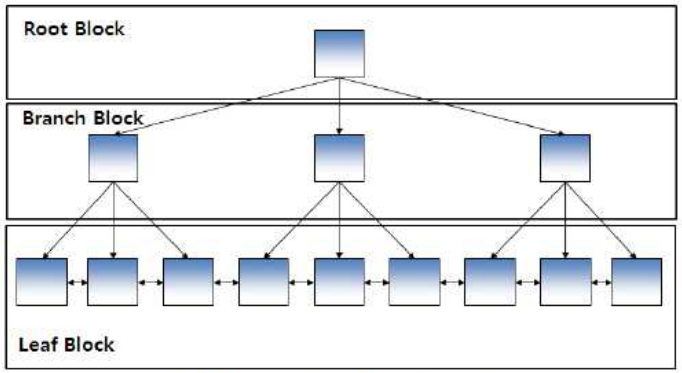
4. 인덱스 종류

- 단일 인덱스
- 결합 인덱스 (여러 칼럼을 묶어 결합 인덱스로 구성)

* 결합순서 *

① =조건으로 많이 쓰는 칼럼이 앞에 오는게 좋음
② BETWEEN 범위 지정하는 칼럼이 그 다음에 오는게 좋음
③ ID처럼 분별력이 높은 칼럼이 그 다음으로 오는게 좋음

5. 트리 기반 인덱스 : RDBMS에서 가장 일반적인 인덱스 (B-트리 인덱스)
- 일치검색, 범위검색 둘 다 적합
- 브랜치 블록(인터널 블록) : 하위 단계의 블록을 가리키는 포인터를 가짐
- 리프 블록 : 인덱스를 구성하는 칼럼의 데이터 + 해당 데이터에 대한 행의 위치를 가리키는 레코드 식별자, 양방향 링크를 가진다



[그림 Ⅱ-3-6] B-트리 인덱스 구조

6. 클러스터형 인덱스 (SQL Server)
- 인덱스는 저장 구조에 따라 클러스터형과 비클러스터형 인덱스로 구분
- 인덱스의 리프 페이지 = 데이터 페이지, 테이블 탐색에 필요한 레코드 식별자가 리프 페이지에 없음
- 리프페이지의 모든 로우(데이터)는 인덱스 키 칼럼순으로 물리적으로 정렬되어 저장됨
- 인덱스 키 칼럼과 나머지 칼럼을 리프에 같이 저장 → 테이블 랜덤 액세스 필요 X
- 클러스터형 인덱스의 리프 페이지를 탐색하면 해당 테이블의 모든 칼럼 값 바로 얻음

EmployeeID	LastName	FirstName	HireDate
1	Davolio	Nancy	1992-05-01 00:00:00.000
2	Fuller	Andrew	1992-08-14 00:00:00.000
3	Leverling	Janet	1992-04-01 00:00:00.000
4	peacock	Margaret	1993-05-03 00:00:00.000
5	Buchanan	Steven	1993-10-17 00:00:00.000
6	Suyama	Michael	1993-10-17 00:00:00.000
7	King	Robart	1994-01-02 00:00:00.000
8	Callahan	Laura	1994-03-05 00:00:00.000
9	Dodsworth	Anne	1994-11-15 00:00:00.000

[그림 Ⅱ-3-8] Employees_pk 클러스터형 인덱스

7. 비트맵 인덱스
- 질의 시스템 구현 시에 모두 알 수 없는 경우인 DW 및 AD-HOC 질의 환경을 위해 설계
- 하나의 인덱스 키 엔트리가 많은 행에 대한 포인터를 저장

8. 인덱스 스캔 방식 종류

① 전체 테이블 스캔 (FULL TABLE SCAN)
- 데이블의 모든 데이터를 읽으며 데이터 추출, 읽은 블록의 재사용성을 낮다고 판단해 메모리 버퍼에서 제거
- 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 유리

② 인덱스 스캔 (INDEX SCAN)
- 인덱스를 구성하는 칼럼의 값을 기반으로 데이터 추출
- 인덱스를 읽어 ROWID를 찾고 해당 데이터를 찾기 위해 테이블을 읽음
- 적은 데이터를 조회할 때 유리
- 랜덤 액세스에 의한 부하가 발생할 수 있고 중복 스캔 비효율이 발생

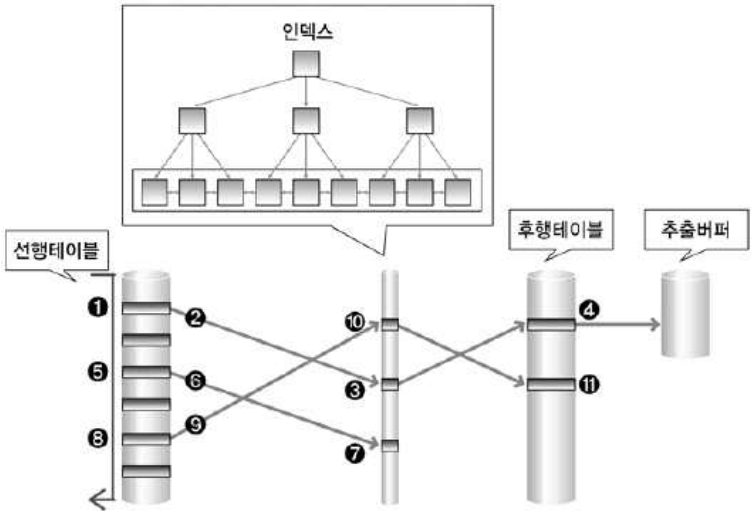
③ 전체 테이블 스캔과 인덱스 스킨 방식의 비교
- 전체 테이블 스캔 : 비효율적, 여러 블록씩. But 테이블 대부분 데이터 찾을 땐 유리
- 인덱스 스캔 : 레코드 식별자 이용, 정확한 위치 알고 읽음. 한번의 I/O요청에 한 블록씩

제3절 조인 수행 원리

1. NL 조인 (주로 액세스 방식으로 데이터 읽음)
- 두 개의 테이블을 중첩된 반복문처럼 조인을 수행
- 반복문 외부(처음 테이블)에 있는 테이블 : 선행 테이블 / 외부 테이블
- 반복문 내부(두번째 테이블)에 있는 테이블 : 후행 테이블 / 내부 테이블
- 결과 행의 수가 적은 테이블을 선행 테이블로 선택한다

2. NL 조인 작업 방법

1) 선행 테이블에서 조건에 맞는 값을 찾는다
2) 선행 테이블의 조인 키를 가지고 후행 테이블 조인 키 확인
3) 후행 테이블의 인덱스에 선행 테이블의 조인 키 존재 확인
4) 인덱스에서 추출한 레코드 식별자를 이요하여 후행 테이블 액세스하여 버퍼에 저장
5) 앞의 작업을 선행 테이블에서 만족하는 키 값이 없을 때 까지 반복하여 수행



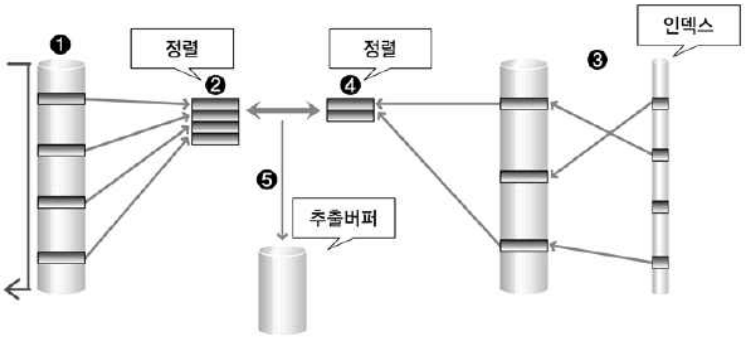
[그림 Ⅱ-3-12] NL Join

3. SORT MERGE JOIN (주로 스캔 방식으로 데이터 읽음)

- 조인 칼럼을 기준으로 데이터를 정렬하여 조인한다
- 넓은 범위의 데이터를 처리할 때 주로 사용
- 정렬 데이터가 많을 경우 디스크 I/O로 인한 부하 발생하여 성능이 떨어질수도
- 비동등 조인에 대해서도 조인이 가능하다
- 인덱스가 존재하지 않을 경우에도 사용할 수 있다

4. SOFT MERGE JOIN 작업 방법

1) 선행 테이블에서 조건에 맞는 행을 찾는다
2) 선행 테이블의 조인 키를 기준으로 정렬 작업 수행
3) 1-2번 작업을 반복 수행하여 모든 행을 찾아 정렬한다
4) 후행테이블에서도 같은 작업 진행
5) 정렬된 결과를 이용하여 조인을 수행하고 결과 값을 추출 버퍼에 저장



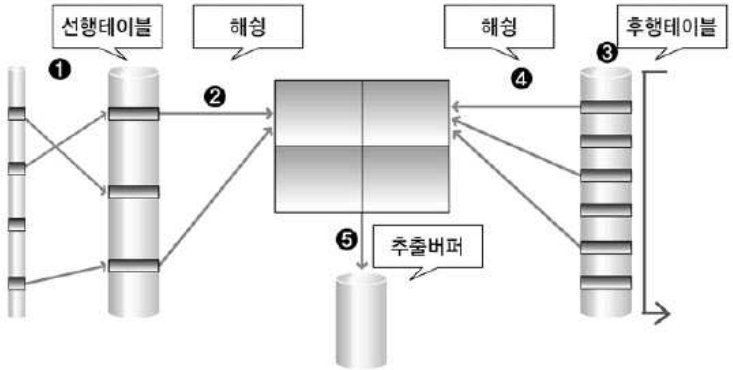
[그림 Ⅱ-3-13] Sort Merge Join

3. Hash JOIN

- 조인을 수행할 테이블의 조인 칼럼을 기준으로 해쉬 함수를 수행하여 서로 동일한 해쉬 값을 갖는 것들 사이에서 실제 값이 같은지 비교
- 조인 칼럼의 인덱스가 존재하지 않을 경우에도 사용가능
- '='로 수행하는 동등 조인만 가능
- 결과 행의 수가 적은 테이블을 선행 테이블로 사용

4. Hash JOIN 작업 방법

1) 선행테이블에서 조건에 만족하는 행을 찾음
2) 선행테이블의 조인 키를 기준으로 해쉬 함수를 적용하여 해쉬 테이블 생성
3) 1-2작업 반복하여 선행 테이블의 모든 조건에 맞는 행을 찾아 해시테이블 완성
4) 후행테이블에서 조건에 만족하는 행을 찾음
5) 후행테이블의 조인 키를 기준으로 해시함수 적용하여 해당 버킷을 찾음
6) 같은 버킷에 해당되면 조인에 성공하여 추출버퍼에 저장
7) 후행 테이블의 조건만큼 반복 수행하여 완료



[그림 Ⅱ-3-14] Hash Join