

1과목(10/50문항)

제1장 데이터모델링의 이해

제1절 데이터모델의 이해

1. 모델링의 이해

가. 모델링의 정의 : “다양한 현상을 표기법에 의해 표기하는 것”

나. ★ 특징 3가지: 추상화, 단순화, 명확화

다. 모델링의 3가지 관점: 데이터 관점, 프로세스 관점, 상관 관점

2. 데이터모델의 기본개념이해

가. 데이터모델링의 정의: “정보시스템을 구축하기 위한 데이터관점의 업무분석기법”

나. 데이터모델이 제공하는 기능

: 시스템 가시화, 시스템 구조와 행동 명세화, 시스템구축의 구조화된 틀 제공

: 문서화, 세부 사항은 숨기는 다양한 관점 제공, 상세 수준의 표현방법 제공

3. 데이터모델링의 중요성 및 유의점

[중요성]

가. 파급효과

나. 복잡한 정보요구사항의 간결한 표현

다. 데이터 품질

★

[유의점]

가. 중복 - 데이터베이스가 여러 장소에 같은 정보를 저장하지 않게 함

나. 비유연성 - 데이터정의를 데이터 사용프로세스와 분리

다. 비일관성 - ex) 신용상태에 대한 개신 없이 고객의 납부이력정보 개신

4. 데이터모델링의 3단계 진행

가. 개념적 데이터모델링 : 추상화 수준이 높고 업무 중심적이고 포괄적인 수준의 모델링 진행

나. 논리적 데이터모델링 : 시스템으로 구축하고자 하는 업무에 대해 Key, 속성, 관계 등을 정확하게 표현, 높은 재사용

다. 물리적 데이터모델링 : 실제로 데이터베이스에 이식할 수 있도록 성능, 저장 등 물리적인 성격 고려 설계

5. 프로젝트 생명주기에서 데이터모델링

* 계획 단계 -> 개념적모델링 // 분석단계 -> 논리적 모델링 // 설계단계 -> 물리적 모델링

* 실제는: 분석단계에서 -> 개념적+논리적 모델링 // 설계단계 -> 물리적 모델링

6. 데이터 모델링에서 데이터독립성의 이해

가. 데이터 독립성의 필요성: 유지보수비용 증가, 데이터복잡도 증가, 데이터증복성 증가, 요구사항대응 저하

나. 데이터베이스 3단계구조: 외부단계 / 개념적단계 / 내부적단계

다. 데이터독립성 요소: 외부스키마 / 개념스키마 / 내부스키마

라. 두 영역의 데이터독립성: 논리적 독립성, 물리적 독립성

마. ★ 사상 ‘Mapping’ : “상호 독립적인 개념을 연결시켜주는 다리” / 외부적/개념적 사상 + 개념적/물리적 사상

7. 데이터모델링의 중요한 세가지 개념

가. 데이터모델링의 세가지 요소

- 1) 업무가 관여하는 어떤 것(Things)
- 2) 어떤 것이 가지는 성격(Attributes)
- 3) 업무가 관여하는 어떤 것 간의 관계(Relationships)

나. 단수와 집합(복수)의 명명

개념	복수/집합개념 & 타입/클래스	개별/단수개념 & 어커런스/인스턴트
어떤것(Thing)	엔터티타입(Entity Type)	엔터티(Entity)
	엔터티(Entity)	인스턴스(Instance) / 어커런스(Occurrence)
어떤것간의 연관(Relationships)	관계(Relationship)	페어링(Pairing)
어떤것의 성격(Attributes)	속성(Attribute)	속성값(Attribute Value)

8. 데이터모델링의 이해관계자

가. 이해관계자의 데이터 모델링 중요성 인식

나. 데이터 모델링의 이해관계자

9. 데이터모델의 표기법인 ERD의 이해

가. 데이터 모델 표기법: 엔터티를 사격형으로 표현, 관계를 마름모, 속성을 타원형으로 표현

나. ERD(Entity Relationship Diagram) 표기법을 이용하여 모델링하는 방법

엔터티그리기→엔터티 배치→엔터티간의 관계설정→관계명 기술→관계의 참여도 기술→관계의 필수여부기술

10. ★ 좋은 데이터모델의 요소

완전성 / 중복배제 / 업무규칙 / 데이터 재사용 / 의사소통 / 통합성

제2절 엔터티

1. 엔터티의 개념: “실체, 객체”

2. 엔터티와 인스턴스에 대한 내용과 표기법 →

3. 엔터티의 특징

가. 업무에서 필요로 하는 정보

나. 식별자에 의해 식별이 가능 해야함

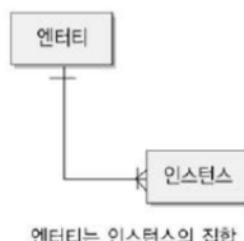
다. 인스턴스의 집합

라. 업무프로세스에 의해 이용

마. 속성을 포함

바. 관계의 존재

엔터티–인스턴스 ERD



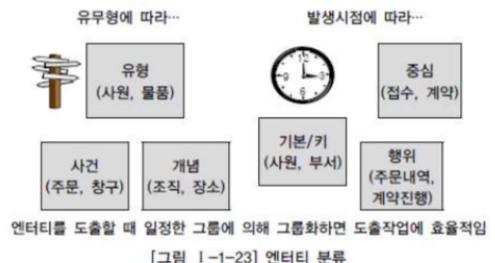
엔터티–인스턴스의 예

엔터티	인스턴스
과 목	수학
영어	영어
강사	이춘식
	조시형
사건	2010-001
	2010-002

【그림 I-1-15】엔터티와 인스턴스

4. ★ 엔터티의 분류

- 가. 유무형에 따른 분류: 유형 엔터티, 개념 엔터티, 사건 엔터티
- 나. 발생시점에 따른 분류: 기본 엔터티, 중심 엔터티, 행위 엔터티
- 다. 엔터티 분류방법의 예 ----->



5. 엔터티의 명명: 업무목적에 따라 생성되는 자연스러운 이름을 부여

제3절 속성

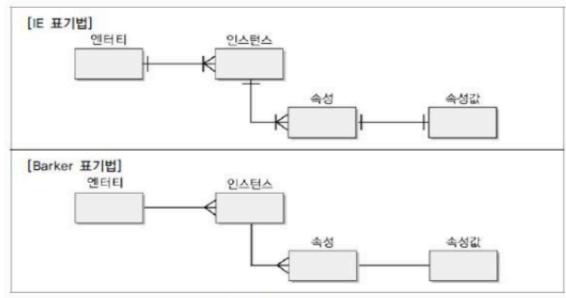
1. 속성의 개념: “업무에서 필요로 하는 인스턴스로 관리하고자하는 의미상 더 이상 분리되지 않는 최소의 데이터 단위”

2. 엔터티, 인스턴스와 속성, 속성값에 대한 내용과 표기법

가. ★ 엔터티, 인스턴스, 속성, 속성값의 관계

- 한 개의 엔터티는 두개 이상의 인스턴스 집합이어야 한다.
- 한 개의 엔터티는 두개 이상의 속성을 갖는다.
- 한 개의 속성은 한 개의 속성값을 갖는다.

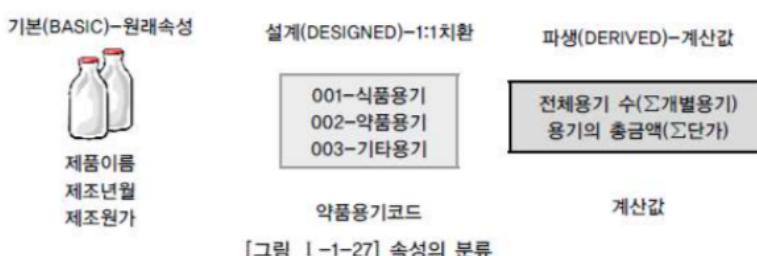
나. 속성의 표기법: IE표기법, Barker 표기법



3. 속성의 특징: 하나의 속성에는 한 개의 값. 하나의 속성에 여러 개의 값이 있는 다중값일 경우 별도의 엔터티를 이용하여 분리

4. 속성의 분류

가. ★ 속성의 특성에 따른 분류: 기본속성, 설계속성, 파생속성



[그림 1-1-27] 속성의 분류

나. 엔터티 구성방식에 따른 분류

- 엔터티를 식별할 수 있는 속성을 PK속성,
- 다른 엔터티와의 관계에서 포함된 속성을 FK속성,
- 엔터티에 포함되어 있고 PK,FK에 포함되지 않은 속성을 일반 속성이라 한다.
- 또한 의미를 조각 수 있는지에 따라 단순형, 복합형으로 분류할 수 있다.
- 속성 하나에 한 개의 값 → 단일값(Single Value) / 여러 개의 값 → 다중값(Multi Value)

5. ★ 도메인: “각 속성이 가질 수 있는 값의 범위”

→ 엔터티 내에서 속성에 대한 데이터타입과 크기 그리고 제약사항을 지정하는 것

제4절 관계

1. 관계의 개념

가. 관계의 정의

“엔터티의 인스턴스 사이의 논리적인 연관성으로서 존재의 형태로서나 행위로서 서로에게 연관성이 부여된 상태”

나. 관계의 패어링: “패어링은 엔터티안에 인스턴스가 개별적으로 관계를 가지는 것이고, 이것의 집합을 관계로 표현”

다. 관계의 분류

- ERD: 존재에 의한 관계 / 행위에 의한 관계

- UML(Unified Modeling Language): 연관 관계 / 의존관계

2. 관계의 표기법

가. 관계명: 관계의 이름

나. 관계 차수: 1:1, 1:M, M:N

다. 관계선택사양: 필수관계, 선택관계

4. 관계의 정의 및 읽는 방법

가. 관계체크사항

나. 관계읽기



제5절 식별자

1. 식별자 개념

- 하나의 엔터티에 구성되어있는 여러 개의 속성 중에 엔터티를 대표할 수 있는 속성을 의미하며

- 하나의 엔터티는 반드시 하나의 유일한 식별자가 존재해야 한다.

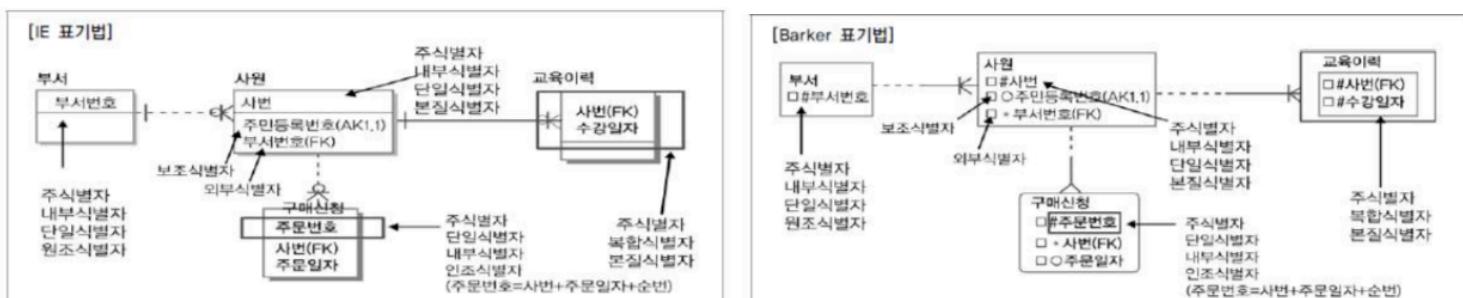
2. ★ 식별자의 특징: 유일성, 최소성, 불변성, 존재성

3. ★ 식별자분류 및 표기법

가. 식별자 분류: 주식별자 보조식별자 / 내부식별자와 외부식별자 / 단일식별자와 복합식별자,

본질식별자와 인조식별자

나. 식별자 표기법

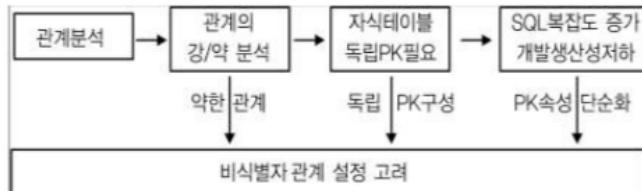


4. 주식별자 도출기준

- 가. 해당업무에서 자주 이용되는 속성을 주식별자로 지정하도록함
- 나. 명칭, 내역등과같이 이름으로 기술되는 것을 피함
- 다. 속성의 수가 많아지지 않도록함

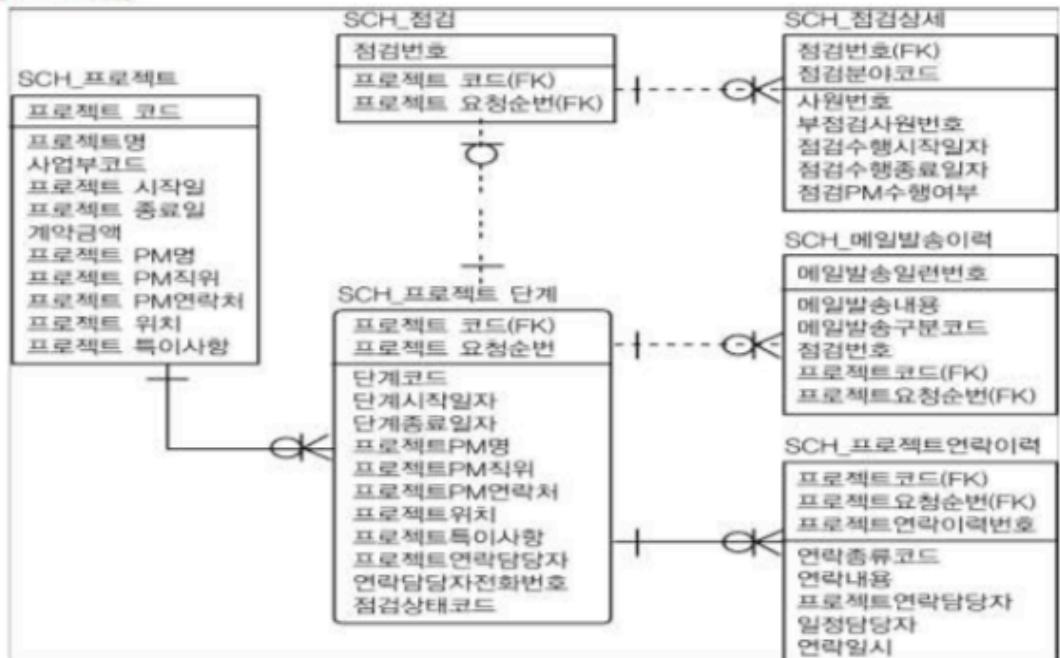
5. 식별자관계와 비식별자 관계에 따른 식별자

- 가. 식별자관계와 비식별자 관계의 결정: 외부식별자는 Foreign Key역할을 한다.
- 나. 식별자 관계: “자식엔터티의 주식별자로 부모의 주식별자가 상속이 되는 경우”
Null값이 오면 안되므로 반드시 부모엔터티가 생성되어야 자기자신의 엔터티생성
- 다. 비식별자 관계: “부모 엔터티로부터 속성으로 받았지만 자식엔터티의 주식별자로 사용하지않고
일반적인 속성으로만 사용하는 경우”
- 라. 식별자관계로만 설정할 경우의 문제점: 주식별자 속성이 지속적으로 증가, 복잡성과 오류 가능성 유발
- 마. 비식별자관계로만 설정한 경우의 문제점: 쓸데없이 부모엔터티까지 찾아가야 하는 경우가 발생한다
- 바. 식별자관계와 비식별자관계 모델링: 비식별자관계 선택프로세스, 식별자와 비식별자 관계비교,
식별자와 비식별자를 적용한 데이터 모델



[그림 1-1-53] 비식별자관계 설정 고려사항

[IE 표기법]



제2장 데이터 모델과 성능

제1절 성능데이터모델링의 개요

1. 성능데이터모델링의 정의

데이터모델링을 할 때 어떤 작업유형에따라 성능향상을 도모해야하는지

목표를 분명하게 해야 정확한 성능향상 모델링을 할 수 있다.

2. 성능 데이터모델링 수행 시점

분석/설계단계에서 데이터베이스 처리 성능을 향상시킬 수 있는 방법을 고려해야 한다

3. 성능 데이터모델링 고려사항

정규화..반정규화...

제2절 정규화와 성능

1. 정규화를 통한 성능향상전략

- ★“데이터에 대한 중복성을 제거하여준다”
- 일반적으로 정규화가 잘되어있으면 입력/수정/삭제의 성능이 향상되고 반정규화를 많이 하면 조회의 성능이 향상된다
- 중복속성에 대한 분리가 1차 정규화의 정의임. 로우단위의 대상, 칼럼 단위로 중복도 해당

2. 반정규화된 테이블의 성능저하사례1

3. 반정규화된 테이블의 성능저하사례2

4. 반정규화된 테이블의 성능저하사례3

5. 반정규화된 테이블의 성능저하사례4

6. ★ 함수적 종속성에 근거한 정규화 수행 필요

- 함수의 종속성(Functional Dependency)은 데이터들이 어떤 기준값에 의해 종속되는 현상을 지칭 하는 것이다.
- 결정자 예) 주민등록번호 // 종속자 예) 이름, 출생지, 주소

제3절 반정규화와 성능

1. 반정규화를 통한 성능향상 전략

가. 반정규화의 정의

“정규화된 엔터티, 속성, 관계에 대해 시스템의 성능향상과 개발과 운영의

단순화를 위해 중복, 통합, 분리 등을 수행하는 데이터모델링 기법”

조인으로 인한 성능저하가 예상될 때 등

나. 반정규화의 적용방법: 반정규화대상 조사, 다른 방법 유도 검토, 반정규화 적용

2. 반정규화 기법

- 가. 테이블 반정규화: 테이블병합, 테이블분할, 테이블추가
- 나. 칼럼 반정규화: 중복칼럼 추가, 파생칼럼 추가, 이력테이블칼럼 추가, PK에 의한 칼럼 추가, 응용시스템 오작동을 위한 칼럼 추가
- 다. 관계 반정규화: 중복관계추가

3. 정규화가 잘 정의된 데이터모델에서 성능이 저하될 수 있는 경우

4. 정규화가 잘 정의된 데이터모델에서 성능이 저하된 경우

제4절 대량데이터에 따른 성능

1. 대량데이터 발생에 따른 테이블 분할 개요

- 로우 체이닝(Row Chaining) - “로우길이가 너무 길어서 데이터블록하나에 데이터가 모두 저장되지 않고 두개 이상 저장”
- 로우 마이그레이션(Row Migration) - “데이터블록에서 수정이 발생하면 수정된 데이터를 해당 데이터블록에서 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식”

2. 한 테이블에 많은 수의 칼럼을 가지고 있는 경우

3. 대량 데이터 저장 및 처리로 인한 성능

- 가. RANGE PARTITION 적용
- 나. LIST PARTITION 적용
- 다. HASH PARTITION 적용

4. 테이블에 대한 수평 분할/수직 분할의 절차

제5절 데이터베이스 구조와 성능

1. 슈퍼타입 / 서브타입 모델의 성능고려방법

가. 슈퍼/서브타입 데이터모델의 개요

“Extended ER모델”, 업무를 구성하는 데이터의 특징을 공통과 차이점의 특징을

고려하여 효과적으로 표현할 수 있기 때문에 자주 쓰임

(공통은 슈퍼타입으로 모델링하고, 공통으로부터 상속받아 다른 엔터티와 차이가 있는 속성은 별도의 서브엔터티 구분)

나. 슈퍼/서브타입 데이터 모의 변환

다. 슈퍼/서브타입데이터모델의 변환 기술

라. 슈퍼/서브타입데이터모델의 변환타입비교

OneToOne Type / Plus Type / Single Type

2. ★ 인덱스 특성을 고려한 PK/FK 데이터베이스 성능 향상

- 일반적으로 프로젝트에서는 PK/FK 칼럼 순서의 중요성을 인지하지 못한 채로

데이터 모델링이 되어 있는 그 상태대로 바로 DDL을 생성함으로써

데이터베이스 데이터처리 성능에 문제를 유발하는 경우가 빈번하게 발생이 된다.

가. ★ PK/FK 칼럼 순서와 성능 개요

- 인덱스의 특징은 여러 개의 속성이 하나의 인덱스로 구성되어 있을 때

앞쪽에 위치한 속성의 값이 비교자로 있어야 인덱스가 좋은 효율을 나타낼 수 있다.

앞쪽에 위치한 속성값이 가급적 '=' 아니면 최소한 범위 BETWEEN이 들어와야 됨

나. PK칼럼의 순서를 조정하지 않으면 성능이 저하 이유

- PK의 순서를 인덱스 특징에 맞게 고려하지 않고 바로 그대로 생성하게 되면,

테이블에 접근하는 트랜잭션의 특징에 효율적이지 않은 인덱스가 생성되어 있으므로

인덱스의 범위를 넓게 이 용하거나 Full Scan을 유발하게 되어 성능이 저하된다고 정리할 수 있다.

다. PK순서를 잘못 지정하여 성능이 저하된 경우-간단한 오류

라. PK순서를 잘못 지정하여 성능이 저하된 경우-복잡한 오류

3. 물리적인 테이블에 FK제약이 걸려있지 않을 경우 인덱스 미생성으로 성능저하

- 물리적인 테이블에 FK 제약 걸었을 때는 반드시 FK 인덱스를 생성하도록 하고

FK 제약이 걸리지 않았을 경우에는 FK 인덱스를 생성하는 것을 기본정책으로 하되

발생되는 트랜잭션에 의해 거의 활용되지 않았을 때에만 FK 인덱스를 지우는 방법으로 하는 것이 적절한 방법

제6절 분산데이터베이스와 성능 →

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.

1. 분산데이터베이스의 개요: “데이터베이스를 연결하는 빠른 네트워크 환경을 이용하여 데이터베이스를

여러 지역 여러 노드로 위치시켜 사용성/성능 등을 극대화시킨 데이터베이스”

2. 분산데이터베이스의 투명성: 분할, 위치, 지역 사상, 중복, 장애, 병행

3. 분산데이터베이스의 적용방법 및 장단점

가. 분산데이터베이스 적용방법

나. 분산데이터베이스 장단점

4. 분산데이터베이스의 활용 방향성

5. 데이터베이스 분산구성의 가치

6. 분산데이터베이스의 적용기법

가. 테이블위치분산

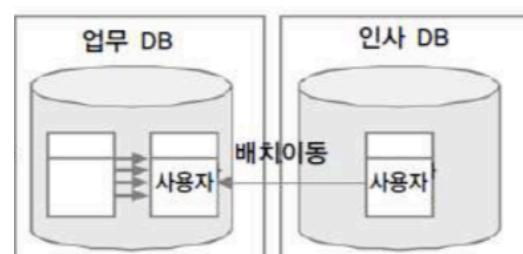
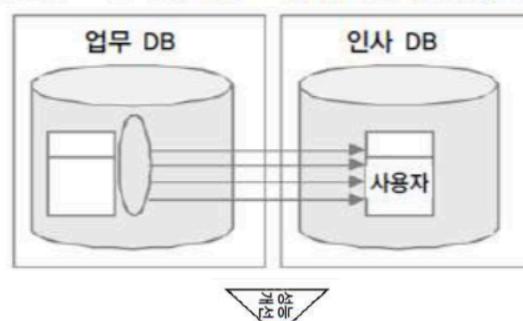
나. 테이블분할분산 - 수평분할, 수직분할

다. 테이블복제분산 - 부분복제, 광역복제

라. 테이블요약분산 - 분석요약, 통합요약

7. 분산데이터베이스를 적용하여 성능이 향상된 사례 →

[그림 I-2-55] 업무 특성에 따른 분산환경 구성



제1장 SQL 기본

제1절 관계형 데이터베이스 개요

1. 데이터베이스: “특정기업이나 조직 또는 개인이 필요에 의해 데이터를 일정한 형태로 저장한 것”

2. ★ SQL: SQL은 관계형데이터베이스에서 데이터 정의, 데이터 조작, 데이터제어를 하기 위해 사용하는 언어

- 데이터조작어(DML) – SELECT, INSERT, UPDATE, DELETE(NOT AUTO COMMIT)
- 데이터정의어(DDL) – CREATE, ALTER, DROP, RENAME(AUTO COMMIT)
- 데이터제어어(DCL) – GRANT, REVOKE
- 트랜잭션제어어(TCL) – COMMIT, ROLLBACK

SQL 문장들의 종류

명령어의 종류	명령어	설명
데이터 조작어 (DML: Data Manipulation Language)	SELECT	데이터베이스에 들어 있는 데이터를 조회하거나 검색하기 위한 명령어를 말하는 것으로 RETRIEVE라고도 한다.
	INSERT	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.
	UPDATE DELETE	
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.

3. TABLE: 테이블은 하나 이상의 칼럼을 가져야 한다

관계형 데이터베이스에서는 모든 데이터를 칼럼과 행의 2차원 구조로 나타낸다.

4. ERD(Entity Relationship Diagram): “관계의 의미를 직관적으로 표현할 수 있는 수단” / 구성요소 3개는 → 엔티티, 관계, 속성

제2절 DDL

1. 데이터 유형

CHARACTER(고정길이 문자열정보), VARCHAR(), NUMERIC, DATE

[표 II-1-7] 자주 쓰이는 데이터 유형

데이터 유형	설명
CHARACTER(s)	<ul style="list-style-type: none">- 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)	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 날짜와 시각 정보 (Oracle은 DATE로 표현, SQL Server는 DATETIME으로 표현)- Oracle은 1초 단위, SQL Server는 3.33ms(millisecond) 단위 관리

2. CREATE TABLE

SQL>>

```
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));
```

- ★ 제약조건(CONSTRAINT): “사용자가 원하는 조건의 데이터만 유지하기 위한 방법”

CONSTRAINT 종류: PRIMARY KEY, UNIQUE KEY, NOT NULL, CHECK, FOREIGN KEY

NULL은 “아직 정의되지 않은 미지의 값”, “현재 데이터를 입력하지 못하는 경우”

DEFAULT는 데이터입력 시에 칼럼의 값이 지정되어 있지 않을 경우 기본값을 설정 가능

- 생성된 테이블 구조 확인: Oracle-“DESCRIBE 테이블명;”, SQL Server-“sp_help ‘dbo.테이블명’”

- SELECT 문장을 통한 테이블 생성 사례: (Create Table ~ As Select~), (Select ~ Into ~)

- 기존테이블의 제약조건 중 NOT NULL만 새로운 복제테이블에 적용됨

3. ALTER TABLE

SQL>>

```
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;
```

- DROP COLUMN: 데이터가 있거나 없거나 모두 삭제 가능. 한번에 하나의 칼럼만 삭제 가능.
- MODIFY COLUMN, RENAME COLUMN
- DROP CONSTRAINT: 테이블 생성 시 부여했던 제약조건을 삭제하는 명령어
- ADD CONSTRAINT: 테이블 생성 이후에 필요에 의해서 제약조건을 추가

4. RENAME TABLE

SQL>>

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

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

5. DROP TABLE

SQL>>

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

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

- 테이블의 모든 데이터 및 구조 삭제. CASCADE CONSTRAINT 옵션은 해당 테이블과
- 관계가 있었던 참조되는 제약조건에 대해서도 삭제한다는 뜻, 복구X

6. TRUNCATE TABLE

SQL>>

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

- 테이블 자체가 삭제되는 것이 아니고, 해당테이블에 들어있던 모든 행들이 제거되는 것(데이터만 제거)
- 기존에 사용하던 테이블의 모든 로우를 제거하기 위한 명령어

제3절 DML

1. INSERT

SQL>>

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); →(밸류 리스트)

2. UPDATE

SQL>>

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

UPDATE PLAYER SET BACK_NO = 99;

UPDATE PLAYER SET POSITION = 'MF'; →(문자값인 경우 ‘ ’ 사용)

3. DELETE

SQL>>

DELETE FROM 삭제를 원하는 정보가 들어있는 테이블명 WHERE 조건절;

DELETE FROM PLAYER; →(조건절이 없으면 전체 테이블 삭제)

4. SELECT

SQL>>

SELECT 칼럼명 FROM 테이블;

SELECT PLAYER_ID, PLAYER_NAME, TEAM_ID, POSITION FROM PLAYER;

SELECT DISTINCT POSITION FROM PLAYER; →(DISTINCT: 중복데이터를 1건으로 표시)

SELECT * FROM PLAYER; →(*: 모든 칼럼명 선택)

SELECT PLAYER_NAME AS 선수명 FROM PLAYER; →(AS: 칼럼명에 별명붙이고 별명으로 표시)

5. 산술연산자와 합성 연산자

SQL>>

SELECT PLAYER_NAME 이름, HEIGHT - WEIGHT "키-몸무게" FROM PLAYER;

- 우선순위를 위한 괄호 적용이 가능하다. 일반적으로 산술 연산을 사용하거나 특정 함수를 적용하게 되면

기존의 칼럼에 대해 새로운 의미를 부여한 것이므로 적절한 ALIAS를 새롭게 부여하는 것이 좋다.

```

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

1. ★ 트랜잭션 개요: “데이터베이스의 논리적 연산 단위” —> (계좌이체 트랜잭션)

- 트랜잭션은 데이터베이스의 논리적 연산단위이다. 트랜잭션(TRANSACTION)이란 밀접히 관련되어 분리될 수 없는 한 개 이상의 데이터베이스 조작을 가리킨다. 하나의 트랜잭션에는 하나 이상의 SQL 문장이 포함된다.
- 트랜잭션은 분할할 수 없는 최소의 단위이다. 그렇기 때문에 전부 적용하거나 전부 취소한다.

★ 트랜잭션 특성: 원자성(all or nothing), 일관성, 고립성, 지속성

[표 II-1-14] 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 개신한 데이터베이스의 내용은 영구적으로 저장된다.

★ [TCL(TRANSACTION CONTROL LANGUAGE)]

- 커밋(COMMIT): 올바르게 반영된 데이터를 데이터베이스에 반영시키는 것
- 롤백(ROLLBACK): 트랜잭션 시작 이전의 상태로 되돌리는 것
- 저장점(SAVEPOINT): 저장점 기능

2. COMMIT

```

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

```

- ★ Oracle은 DML문장 수행 후 사용자가 임의로 COMMIT 또는 ROLLBACK 수행해줘야 함
- ★ SQL Server는 기본적으로 AUTO COMMIT

3. ROLLBACK

Oracle SQL>>

UPDATE PLAYER SET HEIGHT = 100;

————>(480개의 행이 수정되었다.)

ROLLBACK;

————>(롤백이 완료되었다.)

SQL>>

BEGIN TRAN UPDATE PLAYER SET HEIGHT = 100;

ROLLBACK;

————>(롤백이 완료되었다.)

- 테이블 내 입력한 데이터나, 수정한 데이터, 삭제한 데이터에 대하여 COMMIT 이전으로 되돌려, 변경 사항을 취소할 수 있는데 데이터베이스에서는 룰백(ROLLBACK) 기능을 사용한다.

4. SAVEPOINT

SQL>>

SAVEPOINT 세이브포인트명;

SAVEPOINT SVPT1;

ROLLBACK TO SVPT1;

- 저장점(SAVEPOINT)을 정의하면 룰백(ROLLBACK)할 때 트랜잭션에 포함된 전체 작업을 룰백하는 것이 아니라 현 시점에서 SAVEPOINT까지 트랜잭션의 일부만 룰백할 수 있다.

제5절 WHERE 절

1. WHERE 조건절 개요: “자신이 원하는 자료만을 검색하기 위해 이용”

WHERE

SQL>>

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;

- 괄호로 묶은 연산이 제일 먼저 연산 처리된다.
- 연산자들 중에는 부정 연산자(NOT)가 먼저 처리되고,
- 비교 연산자(=,>,>=,<,<=), SQL 비교 연산자(BETWEEN a AND b, IN (list), LIKE, IS NULL)가 처리되고,
- 논리 연산자 중에서는 AND, OR의 순으로 처리된다.
- ★ 처리 순서: 부정 연산자 -> 비교 연산자 -> 논리 연산자

2. 연산자의 종류: 비교 연산자, SQL연산자, 논리 연산자, 부정 비교 연산자, 부정 SQL연산자

[표 II-1-15] 연산자의 종류

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.

2. SQL연산자: IN(list)연산자, LIKE연산자, BETWEEN a AND b 연산자, IS NULL연산자

[표 II-1-19] SQL 연산자의 종류

연산자	연산자의 의미
BETWEEN a AND b	a와 b의 값 사이에 있으면 된다.(a와 b의 값이 포함됨)
IN (list)	리스트에 있는 값 중에서 어느 하나라도 일치하면 된다.
LIKE '비교문자열'	비교 문자열과 형태가 일치하면 된다.
IS NULL	NULL 값인 경우

SQL>>

SELECT ENAME, JOB, DEPTNO FROM EMP WHERE (JOB, DEPTNO) IN ('MANAGER',20),('CLERK',30);
사원 테이블(EMP)에서 JOB이 MANAGER이면서 20번 부서에 속하거나,
JOB이 CLERK이면서 30번 부서에 속하는 사원의 정보(ENAME, JOB, DEPTNO)를
IN 연산자의 다중 리스트를 이용해 출력하라.

SELECT PLAYER_NAME, POSITION, BACK_NO, HEIGHT FROM PLAYER WHERE POSITION LIKE 'MF';

SELECT PLAYER_NAME, POSITION, BACK_NO, HEIGHT FROM PLAYER
WHERE PLAYER_NAME LIKE '장%';
“장”씨 성을 가진 선수들의 정보를 조회하는 WHERE 절

[표 II-1-20] 와일드 카드의 종류

와일드 카드	설명
%	0개 이상의 어떤 문자를 의미한다.
_	1개인 단일 문자를 의미한다.

“BETWEEN a AND b” → “a 이상 b 이하”

SELECT PLAYER_NAME, POSITION, BACK_NO, HEIGHT FROM PLAYER
WHERE HEIGHT BETWEEN 170 AND 180;
키가 170 센티미터 이상 180센티미터 이하인 선수들의 정보

3. ★ IS NULL / IS NOT NULL

- NULL(ASCII 00)은 값이 존재하지 않는 것으로 확정되지 않은 값을 표현할 때 사용한다.
따라서 어떤 값보다 크거나 작지도 않고 ‘ ’(공백, ASCII 32)이나 0(Zero, ASCII 48)과 달리 비교 자체가 불가능한 값인 것이다.
- 어떤 값과 비교할 수도 없으며, 특정 값보다 크다, 적다라고 표현할 수 없다.
- ★ NULL 값과의 수치연산은 NULL 값을 리턴한다.
- ★ NULL 값과의 비교연산은 거짓(FALSE)을 리턴한다.

SQL>>

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 테이블에서 포지션이 NULL값을 갖는 선수이름, 포지션, 팀ID를 출력하라!)

4. 논리 연산자

[표 II-1-21] 논리 연산자의 종류

연산자	연산자의 의미
AND	앞에 있는 조건과 뒤에 오는 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 된다. 즉, 앞의 조건과 뒤의 조건을 동시에 만족해야 하는 것이다.
OR	앞의 조건이 참(TRUE)이거나 뒤의 조건이 참(TRUE)이 되면 결과도 참(TRUE)이 된다. 즉, 앞뒤의 조건 중 하나만 참(TRUE)이면 된다.
NOT	뒤에 오는 조건에 반대되는 결과를 되돌려 준다.

“소속이 삼성블루윙즈이고 키가 170 센티미터 이상인 선수들의 자료를 조회”

SQL>>

```
SELECT PLAYER_NAME 선수이름, POSITION 포지션, BACK_NO 백넘버, HEIGHT 키 FROM PLAYER
WHERE TEAM_ID = 'K02' AND HEIGHT >= 170;
```

“소속이 삼성블루윙즈이거나 전남드래곤즈인 선수들 중에서 포지션이 MF인 선수들 자료를 조회”

SQL>>

```
SELECT PLAYER_NAME 선수이름, POSITION 포지션, BACK_NO 백넘버, HEIGHT 키 FROM PLAYER
WHERE TEAM_ID IN ('K02','K07') AND POSITION = 'MF';
```

“소속팀이 삼성블루윙즈이거나 전남드래곤즈에 소속된 선수들이어야 하고, 포지션이 미드필더(MF)이며 키는 170 센티미터 이상이고 180 이하인 선수들 자료를 조회”

SQL>>

```
SELECT PLAYER_NAME 선수이름, POSITION 포지션, BACK_NO 백넘버, HEIGHT 키 FROM PLAYER
WHERE (TEAM_ID = 'K02' OR TEAM_ID = 'K07') AND POSITION = 'MF'
AND HEIGHT >= 170 AND HEIGHT <= 180;
```

★ 논리 연산자들이 여러 개가 같이 사용되었을 때의 처리 우선순위는 (), NOT, AND, OR의 순서대로 처리된다.

5. 부정 연산자

[표 II-1-23] 부정 연산자 종류

종류	연산자	연산자의 의미
부정 논리 연산자	!=	같지 않다.
	^=	같지 않다.
	◊	같지 않다. (ANSI/ISO 표준, 모든 운영체계에서 사용가능)
	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, b값을 포함하지 않는다)
	NOT IN (list)	list 값과 일치하지 않는다.
	IS NOT NULL	NULL 값을 갖지 않는다.

“삼성블루윙즈 소속인 선수들 중에서 포지션이 미드필더(MF)가 아니고,
키가 175 센티미터 이상 185 센티미터 이하가 아닌 선수들의 자료를 찾아본다.”

SQL>>

```
SELECT PLAYER_NAME 선수이름, POSITION 포지션, BACK_NO 백넘버, HEIGHT 키 FROM PLAYER  
WHERE TEAM_ID = 'K02' AND POSITION  $\neq$  'MF' AND HEIGHT NOT BETWEEN 175 AND 185;
```

6. ★ ROWNUM & TOP: SQL처리결과 집합의 각 행에 대해 임시로 부여되는 일련번호

Oracle – ROWNUM

- Oracle의 ROWNUM은 칼럼과 비슷한 성격의 Pseudo Column으로써

SQL 처리 결과 집합의 각 행에 대해 임시로 부여되는 일련번호이며,

테이블이나 집합에서 원하는 만큼의 행만 가져오고 싶을 때

WHERE 절에서 행의 개수를 제한하는 목적으로 사용한다.

Oracle SQL>>

“MY_TABLE이라는 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  $\leq$  3;
```

SQL – TOP

- SQL Server는 TOP 절을 사용하여 결과 집합으로 출력되는 행의 수를 제한할 수 있다.

- Expression: 반환할 행의 수를 지정하는 숫자이다.

- PERCENT: 쿼리 결과 집합에서 처음 Expression%의 행만 반환됨을 나타낸다.

- WITH TIES : ORDER BY 절이 지정된 경우에만 사용할 수 있으며,

TOP N(PERCENT)의 마지막 행과 같은 값이 있는 경우 추가 행이 출력되도록 지정할 수 있다.

SQL>>

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

“PLAYER 테이블에서 1~5행까지의 PLAYER_NAME 을 출력하라”

```
SELECT TOP(5) PLAYER_NAME FROM PLAYER;
```

- SQL 문장에서 ORDER BY 절이 사용되지 않으면

Oracle의 ROWNUM과 SQL Server의 TOP 절은 같은 기능을 하지만,

ORDER BY 절이 같이 사용되면 기능의 차이가 발생한다.

이 부분은 1장 8절 ORDER BY 절에서 설명하도록 한다.

제6절 함수

1. 내장 함수 개요: 함수는 입력되는 값이 많아도 출력은 하나만 됨(M:1)

[표 II-1-26] 단일행 문자형 함수 사례

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'

2. 문자형 함수

SQL>>

[표 II-1-25] 단일행 문자형 함수의 종류

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, MySQL에서 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 생략 되면 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 제거 할 수 있다.

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

“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)

3. 숫자형 함수

[표 II-1-27] 단일행 숫자형 함수 종류

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.

* 주: Oracle 함수/SQL Server 함수 표시 '/' 없는 경우 공통 함수

[표 II-1-28] 단일행 숫자형 함수 사례

숫자형 함수 사용	결과 값 및 설명
ABS(-15)	15
SIGN(-20)	-1
SIGN(0)	0
SIGN(+20)	1
MOD(7,3) / 7%3	1
CEIL(38,123) / CEILING(38,123)	39
CEILING(-38,123)	-38
FLOOR(38,123)	38
FLOOR(-38,123)	-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. 날짜형 함수

[표 II-1-29] 단일행 날짜형 함수 종류

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

[표 II-1-30] 단일행 날짜형 데이터 연산

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. 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

[표 II-1-33] 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.

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

가. ★ 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 해당 시스템에서 의미 없는 문자로 바꾼다.

[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

나. NULL과 공집합

- SELECT 1 FROM DUAL WHERE 1 = 2; 와 같은 조건이 대표적인 공집합을 발생시키는 쿼리이며,
위와 같이 조건에 맞는 데이터가 한 건도 없는 경우를 공집합이라고 하고, NULL 데이터와는 또 다르게 이해해야 한다.

다. NULLIF(표현식1, 표현식2)

- 표현식1이 표현식2와 같으면 NULL을, 같지 않으면 표현식1을 리턴한다

SQL>>

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

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

라. ★ COALESCE(표현식1,표현식2,…)

- 임의의 개수 표현식에서 NULL이 아닌 최초의 표현식을 나타낸다
- COALESCE 함수는 인수의 숫자가 한정되어 있지 않으며,
임의의 개수 EXPR에서 NULL이 아닌 최초의 EXPR을 나타낸다.
만일 모든 EXPR이 NULL이라면 NULL을 리턴한다.

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 함수중 하나
- GROUP BY 절은 행들을 소그룹화 한다.
- SELECT 절, HAVING 절, ORDER BY 절에 사용할 수 있다.

[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.

2. GROUP BY

- ROLLUP이나 CUBE에 의한 소계가 계산된 결과에는 GROUPING(EXPR)=1이 표시됨
- 그외의 결과에는 GROUPING(EXPR)=0이 표시된다

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
```

3. HAVING

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

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;
```

4. CASE 표현을 활용한 월별 데이터 집계

- “집계 함수(CASE())~GROUP BY” 가능한은, 모델링의 제1정규화로 인해 반복되는 칼럼의 경우
구분 칼럼을 두고 여러 개의 레코드로 만들어진 집합을, 정해진 칼럼 수만큼 확장해서 집계 보고서를 만드는 유용한 기법이다.

5. 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 사용하면 된다.

제8절 ORDER BY

1. ★ ORDER BY

- ORDER BY 절은 SQL 문장으로 조회된 데이터들을 다양한 목적에 맞게 특정 칼럼을 기준으로 정렬하여 출력하는데 사용
- ORDER BY 절에 칼럼(Column)명 대신에 SELECT 절에서 사용한 ALIAS 명이나 칼럼 순서를 나타내는 정수도 사용 가능
- 기본적으로 오름차순

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;
```

2. SELECT 문장 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

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

1. JOIN 개요

“두개 이상의 테이블들을 연결 또는 결합하여 데이터를 출력하는 것”

2. EQUI JOIN

- 두 개의 테이블 간에 칼럼 값들이 서로 정확하게 일치하는 경우에 사용되는 방법
- 대부분 PK ↔ FK의 관계를 기반으로 한다. 그러나 반드시 PK ↔ FK의 관계로만 EQUI JOIN 이 성립하는 것은 아니다.

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;
```

3. Non EQUI JOIN

- 두 개의 테이블 간에 칼럼 값들이 서로 정확하게 일치하지 않는 경우에 사용된다.
- Non EQUI JOIN의 경우에는 “=” 연산자가 아닌 다른(Between, >, >=, <, <= 등) 연산자들을 사용하여 JOIN을 수행
- 4.3개이상 TABLE JOIN

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;
```

제1절 표준 조인

1. STANDAR SQL

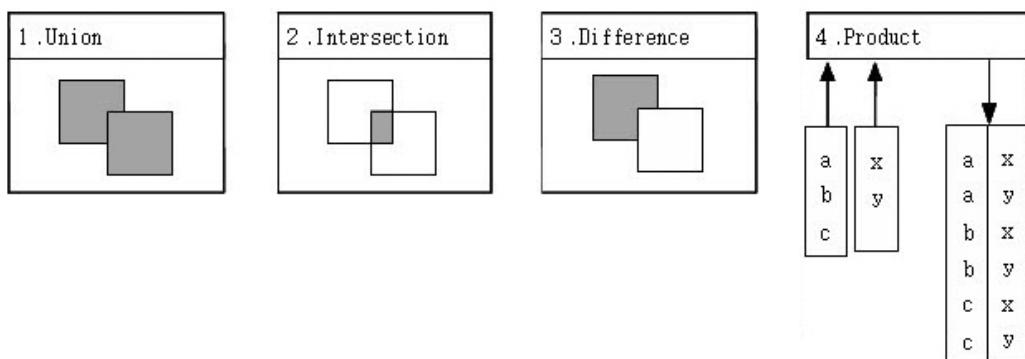
가. ★ 일반 집합연산자 → 현재의 SQL

UNION → UNION

INTERSECTION → INTERSECT

DIFFERENCE → EXCEPT(Oracle은 MINUS)

PRODUCT → CROSS JOIN(CARTESIAN PRODUCT)



[그림 II -2-1] E.F.CODD 일반 집합 연산자

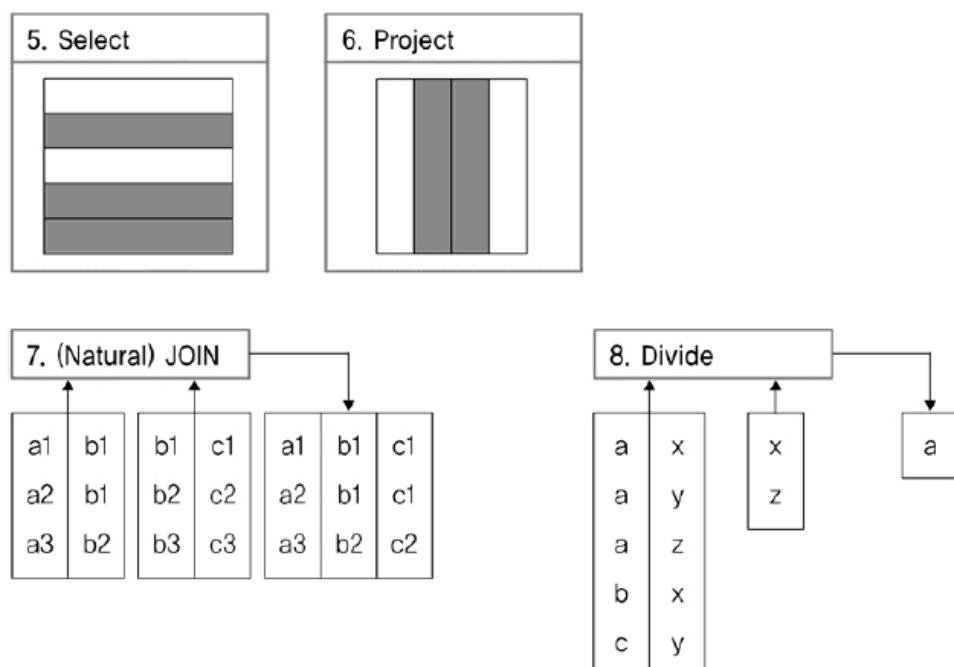
나. 순수관계연산자 → 현재의 SQL

SELECT→WHERE

PROJECT→SELECT

(NATURAL)JOIN→다양한JOIN

DIVIDE→현재 사용하지 않음



[그림 II -2-2] E.F.CODD 순수 관계 연산자

2. FROM절 JOIN 형태

ANSI/ISO SQL에서 표시하는 FROM 절의 JOIN 형태는 다음과 같다.

- INNER JOIN – NATURAL JOIN – USING 조건절 – ON 조건절 – CROSS JOIN – OUTER JOIN

3. ★ INNER JOIN

- ‘내부 JOIN’이라고 함. JOIN 조건에서 동일한 값이 있는 행만 반환한다.
- DEFAULT옵션이므로 생략이 가능하지만, CROSS JOIN, OUTER JOIN과는 같이 사용할 수 없다.
- USING 조건절이나 ON조건절을 필수적으로 사용해야 한다.

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;

4. ★ NATURAL JOIN

- 두테이블간의 동일한 이름을 갖는 모든 칼럼들에 대해 EQUI(=)JOIN을 수행한다
- 추가로 USING조건절, ON조건절, WHERE절에서 JOIN조건을 정의할 수 없다
- JOIN에 사용된 칼럼들은 같은 데이터 유형이어야 한다
- ALIAS나 테이블명과 같은 접두사를 붙일 수 없다

SQL>>

“사원 번호와 사원 이름, 소속부서 코드와 소속부서 이름을 출력하시오.”

SELECT DEPTNO, EMPNO, ENAME, DNAME FROM EMP NATURAL JOIN DEPT;

5. ★ USING 조건절

- 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 =JOIN을 할 수 있다.
- SQL Server에서는 지원하지 않는다
- JOIN칼럼에 대해서는 ALIAS나 테이블이름과 같은 접두사를 붙일 수 없다

Oracle SQL>>

SELECT * FROM DEPT JOIN DEPT_TEMP USING (DEPTNO);

6. ★ ON 조건절

- 칼럼 명이 다르더라도 JOIN 조건을 사용할 수 있는 장점이 있다
- WHERE 검색 조건은 충돌 없이 사용할 수 있다
- ON 조건절에서 사용된 괄호는 옵션사항이다
- ★ ALIAS나 테이블명과 같은 접두사를 사용해야 한다

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);
```

7. CROSS JOIN

- CARTESIAN PRODUCT / “JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합”

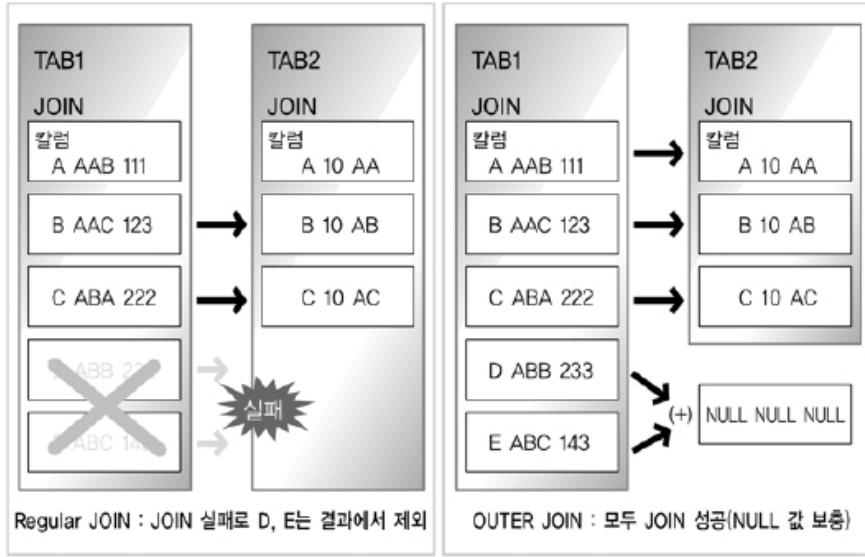
SQL>>

“사원 번호와 사원 이름, 소속부서 코드와 소속부서 이름을 찾아본다.”

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

8. ★ OUTER JOIN

- JOIN 조건에서 동일한 값이 없는 행도(NULL값도) 출력된다.
- USING 조건 절이나 ON 조건 절을 필수적으로 사용해야한다



[그림 II-2-3] OUTER JOIN 설명

가. LEFT OUTER JOIN

- 조인 수행 시 좌측 테이블에 해당하는 데이터를 먼저 읽은 후, 우측테이블에서 JOIN 대상 데이터를 읽어온다
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)
```

나. RIGHT OUTER JOIN

- LEFT OUTER JOIN와 반대로 우측 테이블이 기준이 되어 결과 생성

다. FULL OUTER JOIN

- 합집합 개념으로 LEFT와 RIGHT를 모두 읽어 온다

SQL>>

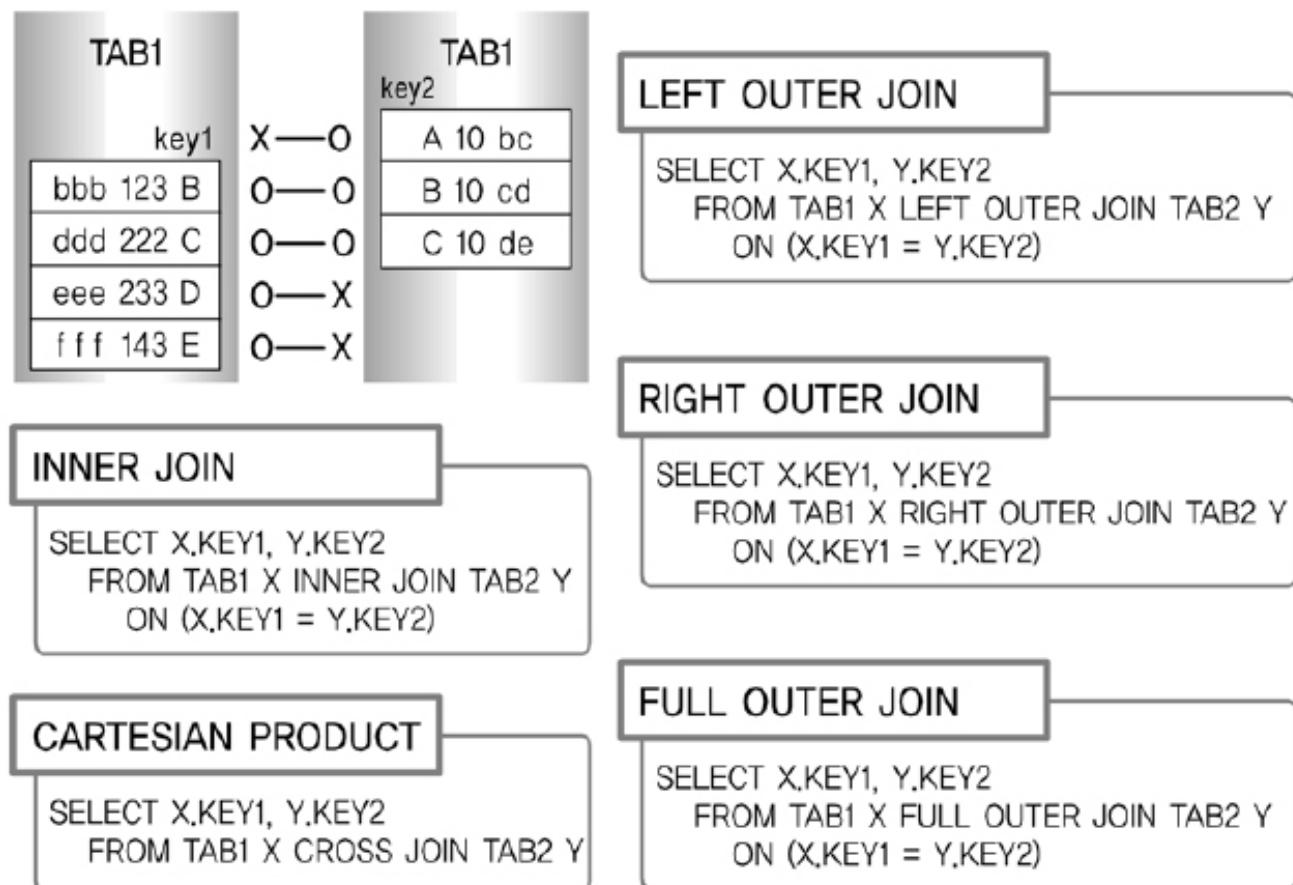
```
UPDATE DEPT_TEMP SET DEPTNO = DEPTNO + 20;
```

→(먼저 업데이트)

```
SELECT * FROM DEPT_TEMP;
```

→(업데이트한 모두를 읽어온다)

9. INNER vs OUTER 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-NONE, E-NONE 인 4건이 출력된다.

세 번째, RIGHT OUTER JOIN의 결과는 다음과 같다.

TAB2를 기준으로 키 값 조합이 NONE-A, B-B, C-C 인 3건이 출력된다.

네 번째, FULL OUTER JOIN의 결과는 다음과 같다.

양쪽 테이블을 기준으로 키 값 조합이 NONE-A, B-B, C-C, D-NONE, E-NONE 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 $4 * 3 = 12$ 건이 추출됨

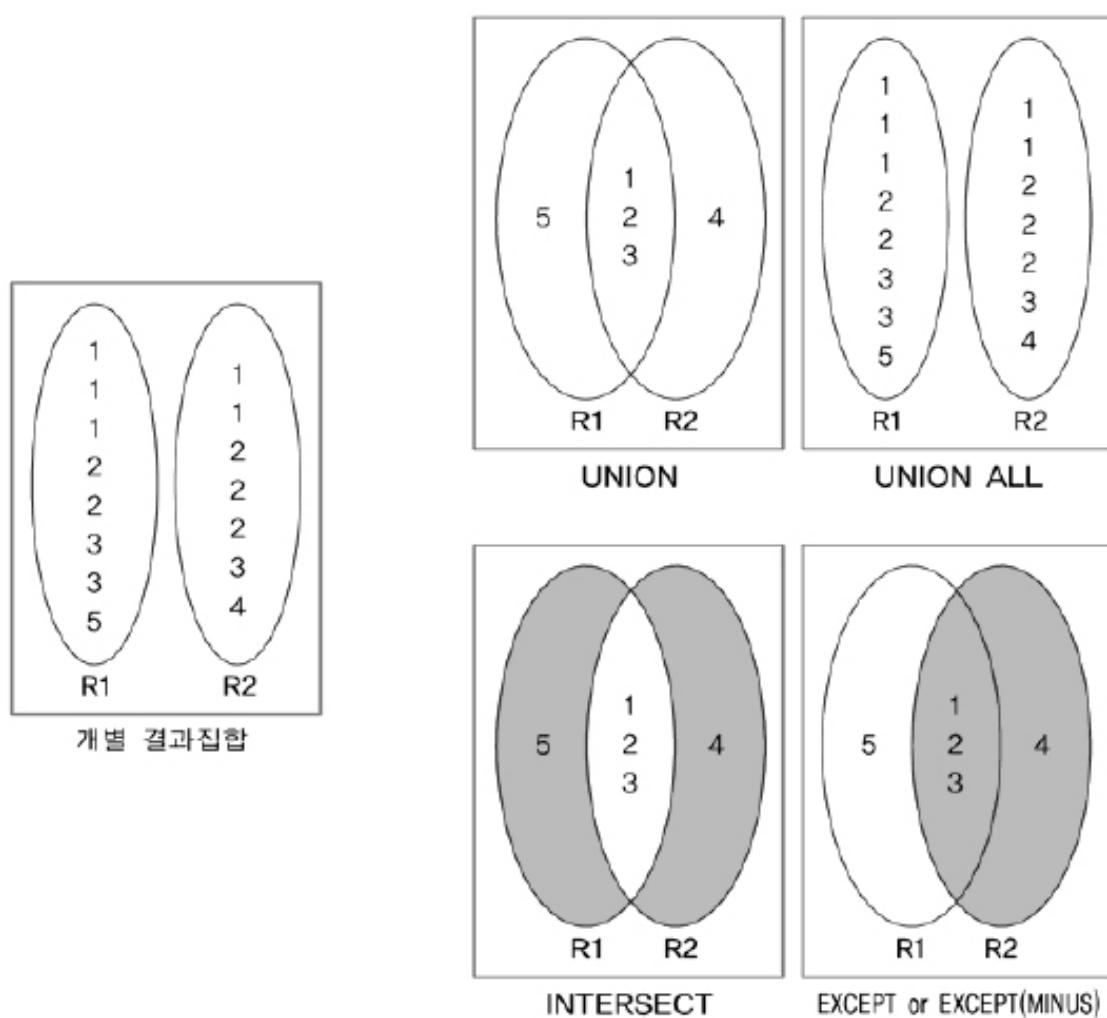
키 값 조합이 B-A, B-B, B-C, C-A, C-B, C-C, D-A, D-B, D-C, E-A, E-B, E-C 인 12건이 출력된다.

제2절 집합 연산자

- 두개 이상의 테이블에서 조인을 사용하지 않고 연관된 데이터를 조회하는 방법 중 하나
- 집합연산자는 2개 이상의 질의 결과를 하나의 결과로 만들어준다
- SELECT절의 칼럼 수가 동일하고 동일 위치에 존재하는 칼럼의 데이터타입이 상호 호환 가능해야 한다

[표 II-2-1] 집합 연산자의 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)



[그림 II-2-5] 집합 연산자의 연산

제3절 계층형 질의와 셀프 조인

1. 계층형 질의

- 테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해서 계층형 질의(Hierarchical Query)를 사용한다.
- 엔터티를 순환관계 데이터 모델로 설계할 경우 계층형 데이터가 발생한다. (예: 조직, 사원, 메뉴 등)

가. ★ Oracle 계층형 질의

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

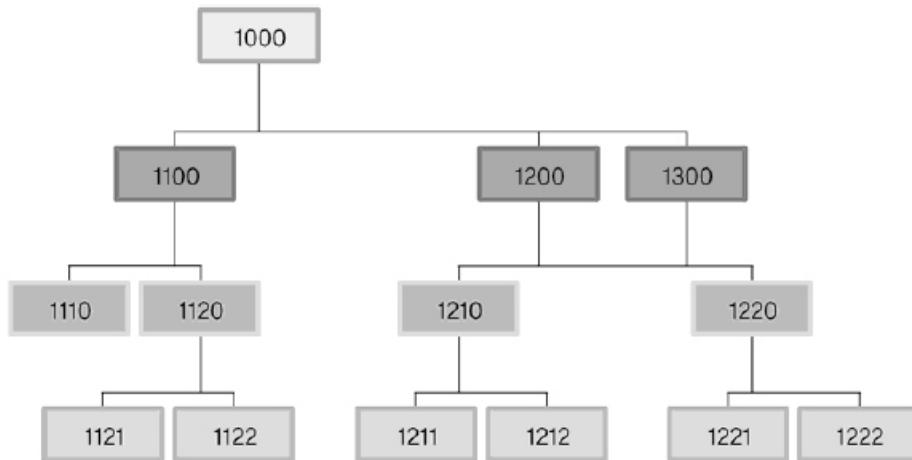
[그림 II-2-7] 계층형 질의 구문

[표 II-2-2] 계층형 질의에서 사용되는 가상 칼럼

가상 칼럼	설명
LEVEL	루트 데이터이면 1, 그 하위 데이터이면 2이다. 리프(Leaf) 데이터까지 1씩 증가 한다.
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.

나. ★ SQL Server 계층형 질의

CTE(Common Table Expression)를 재귀 호출



[그림 II-2-10] 조직도 예제

정리하자면 다음과 같다. 먼저, 앵커 멤버가 시작점이자 Outer 집합이 되어 Inner 집합인 재귀 멤버와 조인을 시작한다. 이어서, 앞서 조인한 결과가 다시 Outer 집합이 되어 재귀 멤버와 조인을 반복하다가 조인 결과가 비어 있으면 즉, 더 조인할 수 없으면 지금까지 만들어진 결과 집합을 모두 합하여 리턴한다.

2. 셀프 조인: “동일 테이블 사이의 조인”, 반드시 테이블 별칭(Alias)을 사용해야 한다.

제4절 서브 쿼리



[그림 II-2-12] 메인쿼리와 서브쿼리

- “하나의 SQL문안에 포함되어있는 또다른 SQL문” —>
- 서브 쿼리는 메인 쿼리의 칼럼을 모두 사용할 수 있지만, 메인 쿼리는 서브 쿼리의 칼럼을 사용할 수 없다.
- 서브쿼리를 괄호로 감싸서 사용한다. 단일행 또는 복수행 비교연산자와 함께 사용가능하다. ORDER BY를 사용하지 못한다

[표 II-2-4] 동작하는 방식에 따른 서브쿼리 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 얹혀진 데이터를 서브쿼리에서 조건이 맞는지 확인하고자 할 때 주로 사용된다.

[표 II-2-5] 반환되는 데이터의 형태에 따른 서브쿼리 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.

1. 단일행 서브 쿼리

- 단일행 비교연산자와 함께 사용할 때는 서브 쿼리의 결과 건수가 반드시 1건 이하여야 함

2. ★ 다중행 서브 쿼리

- 실행 결과가 여러건임. IN, ALL, ANY, SOME, EXISTS

[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 ANY 와 동일함)
EXISTS (서브쿼리)	서브쿼리의 결과를 만족하는 값이 존재하는지 여부를 확인하는 조건을 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;
```

3. 다중칼럼서브쿼리

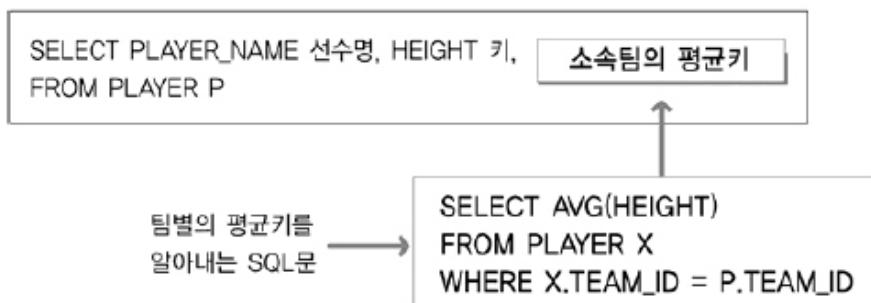
- “서브 쿼리의 결과로 여러 개의 칼럼이 반환되어 메인 쿼리의 조건과 동시에 비교되는 것”

4. 연관서브쿼리

- “서브쿼리내에 메인쿼리칼럼이 사용된 서브 쿼리” EXISTS는 항상 연관서브쿼리로 사용”

5. 그밖에 위치에서 사용하는 서브 쿼리

- 가. ★ SELECT 절에 서브 쿼리 사용하기 → 스칼라 서브쿼리(한 행, 한 칼럼만을 반환하는 서브 쿼리)



[그림 II-2-15] 스칼라 서브쿼리

- 나. ★ FROM 절에서 서브 쿼리 사용하기 → 인라인뷰(Inline View), 동적뷰(Dynamic View)

- 인라인뷰에서는 ORDER BY절을 사용할 수 있다. 인라인뷰에 먼저 정렬을 수행하고
- 정렬된 결과 중에서 일부데이터를 추출하는 것을 TOP-N 쿼리라고 한다
- ★ Oracle에서는 ROWNUM이라는 연산자를 통해 결과 중 일부데이터만 추출 가능

다. HAVING 절에서 서브쿼리사용하기

라. UPDATE문의 SET 절에서 사용하기

마. INSERT문의 VALUES절에서 사용하기

6. ★ 뷰

- 테이블은 실제로 데이터를 가지고있는 반면, 뷰는 실제데이터를 가지고있지 않다

[표 II-2-7] 뷰 사용의 장점

뷰의 장점	설명
독립성	테이블 구조가 변경되어도 뷰를 사용하는 응용 프로그램은 변경하지 않아도 된다.
편리성	복잡한 질의를 뷰로 생성함으로써 관련 질의를 단순하게 작성할 수 있다. 또한 해당 형태의 SQL문을 자주 사용할 때 뷰를 이용하면 편리하게 사용할 수 있다.
보안성	직원의 급여정보와 같이 숨기고 싶은 정보가 존재한다면, 뷰를 생성할 때 해당 칼럼을 빼고 생성함으로써 사용자에게 정보를 감출 수 있다.

제5절 그룹 함수

1. 데이터분석 개요

- ANSI/ISO SQL 표준은 데이터 분석을 위해서 다음 세 가지 함수를 정의하고 있다.
- AGGREGATE FUNCTION, GROUP FUNCTION, WINDOW FUNCTION

2. ★ ROLLUP 함수

- ROLLUP에 지정된 Grouping Columns의 List는 Subtotal을 생성하기 위해 사용되어지며, Grouping Columns의 수를 N이라고 했을 때 N+1 Level의 Subtotal이 생성된다.
- 계층 구조이므로 인수 순서가 바뀌면 수행 결과도 바뀜. 가능한 Subtotal만 생성
- GROUP BY의 확장된 형태

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 대하여 다차원 집계를 생성한다

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

- 원하는 부분의 소계만 손쉽게 추출할 수 있다. 인수는 평등한 관계이므로 인수의 순서 바뀌어도 결과는 같다

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	expr2 + 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. ★ WINDOW FUNCTION 개요

- OVER문구가 키워드로 필수 포함된다
- 분석 함수(ANALYTIC FUNCTION), 순위 함수(RANK FUNCTION)
- WINDOWING함수 2가지 종류(BETWEEN 사용 타입, BETWEEN 미사용 타입)

SQL>>

SELECT WINDOW_FUNCTION (ARGUMENTS)

OVER ([PARTITION BY 칼럼] [ORDER BY 절] [WINDOWING 절]) FROM 테이블 명;

2. 그룹 내 순위 함수

가. RANK 함수

나. ★ DENSE_RANK 함수: 동일한 순위를 하나의 건수로 취급, 중간 순위를 안 비움

다. ★ ROW_NUMBER 함수

- 동일한 값이라도 고유한 순위를 부여한다는 점이 RANK,DENSE_RANK와 다르다.

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;

3. 일반집계 함수

가. SUM 함수

나. MAX 함수

다. MIN 함수

라. AVG 함수

마. COUNT 함수

4. 그룹 내 행 순서 함수

가. FIRST_VALUE 함수: 파티션별 윈도우에서 가장 먼저 나온 값

나. LAST_VALUE 함수: 파티션별 윈도우에서 가장 나중에 나온 값

다. ★ LAG 함수: “현재 읽혀진 데이터의 이전 값을 알아내는 함수”

라. ★ LEAD 함수: “이후 값을 알아내는 함수”

5. 그룹 내 비율함수

가. RATIO_TO_REPORT 함수: 전체 SUM(칼럼)값에 대한 행별 칼럼 값의 백분율을 소수점으로 구함

나. PERCENT_RANK 함수: 제일 먼저 나오는 것을 0으로, 제일 늦게 나오는 것을 1로 하여,

값이 아닌 행의 순서별 백분율을 구한다.

다. CUME_DIST 함수: 전체건수에서 현재 행보다 작거나 같은 건수에 대한 누적백분율

라. NTILE 함수: 전체 건수를 ARGUMENT 값으로 N 등분한 결과

제7절 DCL

1. DCL 개요: 유저를 생성하고 권한을 제어할 수 있는 DCL(DATA CONTROL LANGUAGE) 명령어

[표 II-2-8] Oracle에서 제공하는 유저들

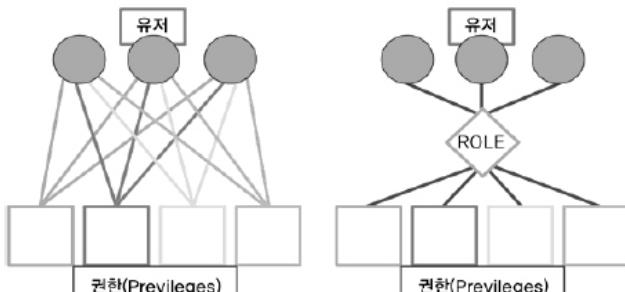
유저	역할
SCOTT	Oracle 테스트용 샘플 유저 Default 패스워드 : TIGER
SYS	DBA ROLE을 부여받은 유저
SYSTEM	데이터베이스의 모든 시스템 권한을 부여받은 DBA 유저 Oracle 설치 완료 시에 패스워드 설정

2. 유저와 권한

- 가. 유저 생성과 시스템 권한 부여: 룰(ROLE)을 이용하여 간편하고 쉽게 권한을 부여
나. OBJECT에 대한 권한 부여: 오브젝트 권한은 특정 오브젝트인 테이블, 뷰 등에 대한
SELECT, INSERT, DELETE, UPDATE 작업 명령어를 의미

3. ★ Role을 이용한 권한 부여

- 많은 데이터베이스에서 유저들과 권한들 사이에서 중개 역할을 하는 ROLE을 제공한다.
- 데이터베이스 관리자는 ROLE을 생성하고, ROLE에 각종 권한들을 부여한 후
ROLE을 다른 ROLE이나 유저에게 부여할 수 있다.
- ROLE에 포함되어 있는 권한들이 필요한 유저에게는 해당 ROLE만을 부여함으로써 빠르고 정확하게 필요한 권한을 부여



[그림 II-2-17] ROLE의 개념

[표 II-2-13] 데이터베이스 수준 역할명 (SQL Server 사례)

데이터베이스 수준 역할명	설명
db_accessadmin	Windows 로그인, Windows 그룹 및 SQL Server 로그인의 데이터베이스에 대한 액세스를 추가하거나 제거할 수 있다.
db_backupoperator	데이터베이스를 백업할 수 있다.
db_datareader	모든 사용자 테이블의 모든 데이터를 읽을 수 있다.
db_datawriter	모든 사용자 테이블에서 데이터를 추가, 삭제, 변경할 수 있다.
db_ddladmin	데이터베이스에서 모든 DDL 명령을 수행할 수 있다.
db_denydatareader	데이터베이스 내에 있는 사용자 테이블의 데이터를 읽을 수 없다.
db_denydatawriter	데이터베이스 내의 모든 사용자 테이블에 있는 데이터를 추가, 삭제, 변경할 수 없다.
db_owner	데이터베이스 내에 있는 모든 구성 및 유지관리 작업을 수행할 수 있고 데이터베이스를 삭제할 수도 있다.
db_securityadmin	역할 멤버 자격을 수정하고 사용 권한 관리를 할 수 있다. 이 역할에 보안 주체를 추가하면 원하지 않는 권한 상승이 설정될 수 있다.

제8절 절차형 SQL

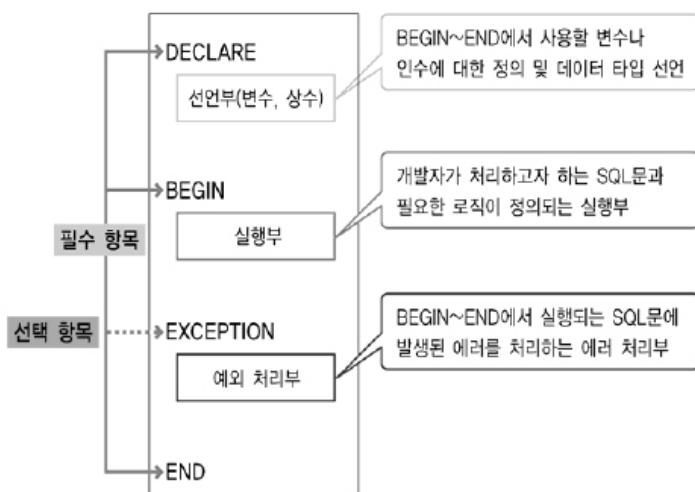
1. ★ 절차형 SQL 개요

- SQL에도 절차 지향적인 프로그램이 가능하도록 DBMS 벤더별로 PL(Procedural Language)/SQL(Oracle), SQL/PL(DB2), T-SQL(SQL Server) 등의 절차형 SQL을 제공하고 있다.
- 절차형 SQL을 이용하면 SQL문의 연속적인 실행이나 조건에 따른 분기처리를 이용하여 특정 기능을 수행하는 저장 모듈을 생성할 수 있다.

2. PL/SQL 개요

가. PL/SQL 특징: Oracle의 PL/SQL은 Block 구조로 되어있고 Block 내에는 DML 문장과 QUERY 문장, 그리고 절차형 언어(IF, LOOP) 등을 사용할 수 있으며, 절차적 프로그래밍을 가능하게 하는 트랜잭션 언어이다.

나. PL/SQL 구조



[그림 II-2-19] PL/SQL 블록 구조

다. PL/SQL 기본 문법

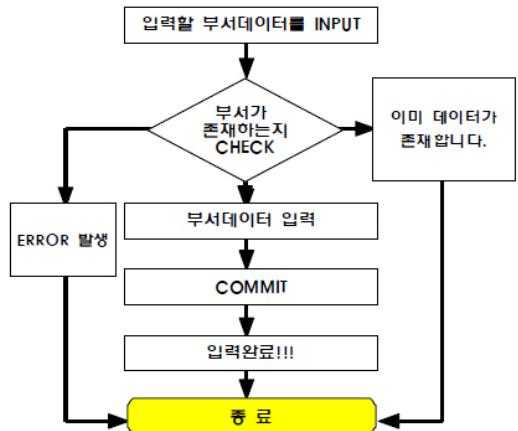
3. T-SQL 개요

가. T-SQL 특징: T-SQL은 근본적으로 SQL Server를 제어하기 위한 언어로서, T-SQL은 엄격히 말하면, MS사에서 ANSI/ISO 표준의 SQL에 약간의 기능을 더 추가해 보완적으로 만든 것

나. T-SQL 구조

다. T-SQL 기본 문법

4. Procedure의 생성과 활용



[그림 II-2-21] 부서 입력 FLOW CHART

5. User Defined Function의 생성과 활용

- 절차형 SQL을 로직과 함께 데이터베이스 내에 저장해 놓은 명령문의 집합을 의미한다.

6. ★ Trigger의 생성과 활용

- “특정한 테이블에 INSERT, UPDATE DELETE와 같은 DML문이 수행되었을 때,
데이터베이스에서 자동으로 동작하도록 작성된 프로그램”
- 즉 사용자가 직접 호출하여 사용하는 것이 아니고 데이터베이스에서 자동적으로 수행하게 된다.
Trigger는 테이블과 뷰, 데이터베이스 작업을 대상으로 정의할 수 있으며,
전체 트랜잭션 작업에 대해 발생되는 Trigger와 각 행에 대해서 발생되는 Trigger가 있다.
- Trigger는 데이터베이스에 의해 자동 호출되지만 결국
INSERT, UPDATE, DELETE 문과 하나의 트랜잭션 안에서 일어나는 일련의 작업들이라 할 수 있다.
- Trigger는 데이터베이스 보안의 적용, 유효하지 않은 트랜잭션의 예방, 업무 규칙 자동 적용 제공 등에 사용될 수 있다.

7. ★ 프로시저와 트리거의 차이

- 프로시저는 BEGIN ~ END 절 내에 COMMIT, ROLLBACK과 같은 트랜잭션 종료 명령어를 사용할 수 있지만,
데이터베이스 트리거는 BEGIN ~ END 절 내에 사용할 수 없다.

[표 II-2-19] 프로시저와 트리거의 차이점

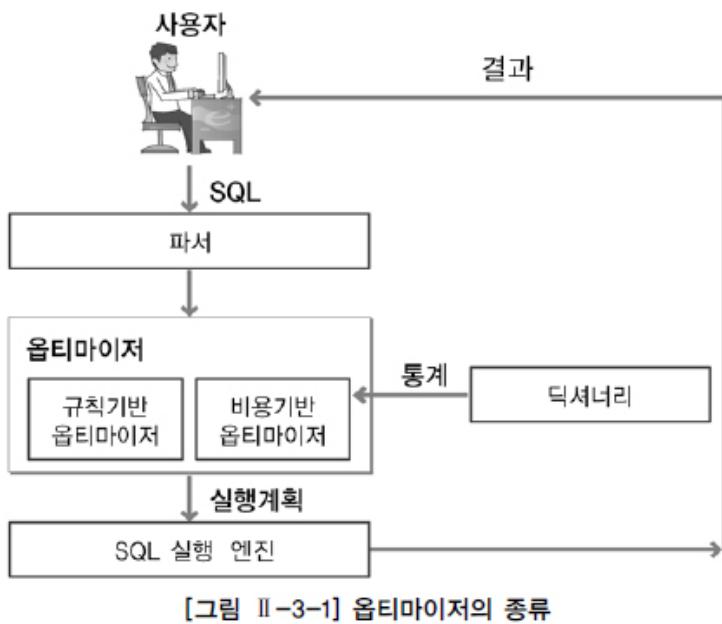
프로시저	트리거
CREATE Procedure 문법사용	CREATE Trigger 문법사용
EXECUTE 명령어로 실행	생성 후 자동으로 실행
COMMIT, ROLLBACK 실행 가능	COMMIT, ROLLBACK 실행 안됨

제3장 SQL 최적화 기본원리

제1절 옵티마이저와 실행계획

1. ★ 옵티마이저

- “다양한 실행방법들 중에서 최적의 실행방법을 결정하는 것이 옵티마이저의 역할”

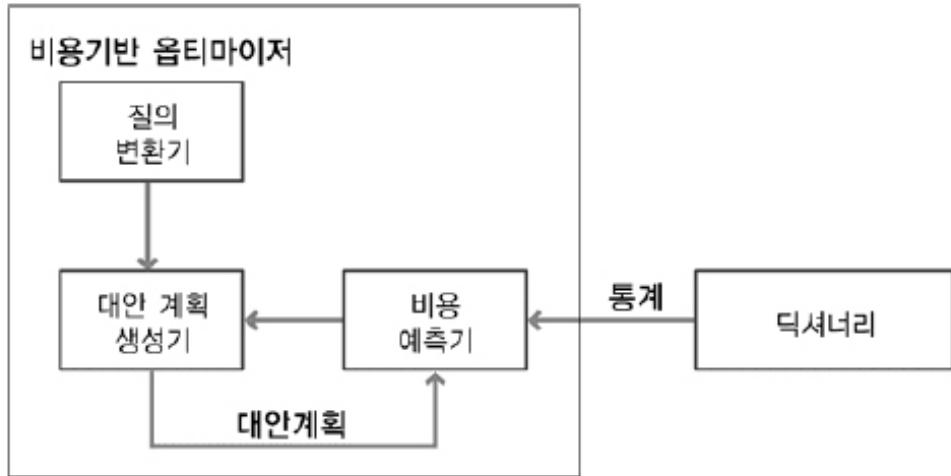


가. ★ 규칙기반 옵티마이저: 규칙(우선순위)을 가지고 실행계획을 생성한다.

순위	액세스 기법
1	Single row by rowid
2	Single row by cluster join
3	Single row by hash cluster key with unique or primary key
4	Single row by unique or primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite index
9	Single column index
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort merge join
13	MAX or MIN of indexed column
14	ORDER BY on indexed column
15	Full table scan

[그림 II-3-2] 규칙기반 옵티마이저의 규칙

나. ★ 비용기반 옵티マイ저: SQL문을 처리하는데 필요한 비용이 가장 적은 실행계획을 선택하는 방식



[그림 II-3-3] 비용기반 옵티マイ저의 구성 요소

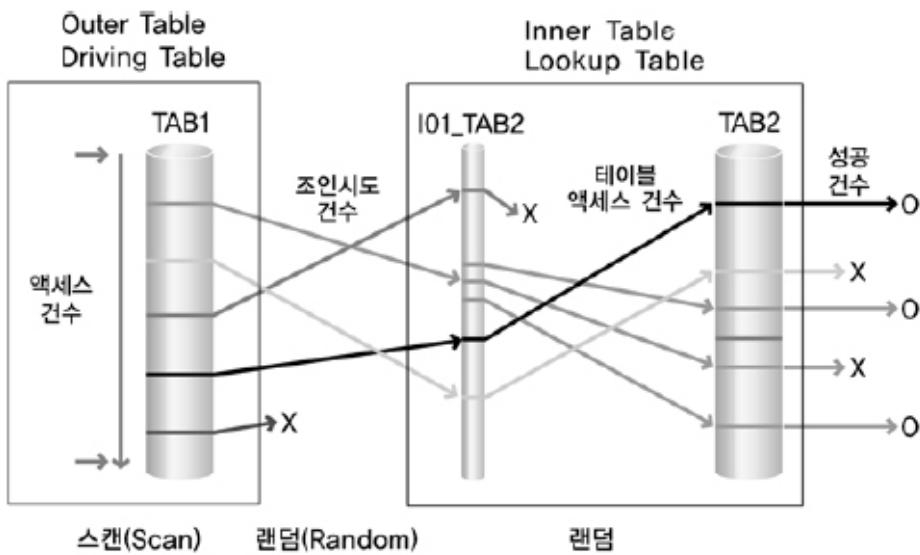
- 비용기반 옵티マイ저는 통계정보, DBMS 버전, DBMS 설정 정보 등의 차이로 인해 동일 SQL문도 서로 다른 실행계획이 생성될 수 있다.
- 또한 비용기반 옵티マイ저의 다양한 한계들로 인해 실행계획의 예측 및 제어가 어렵다는 단점이 있다.

2. ★ 옵티マイ저 실행계획

- “SQL에서 요구한 사항을 처리하기 위한 절차와 방법”
실행계획을 구성하는 요소에는 조인순서, 조인기법, 액세스기법, 최적화정보, 연산 등
- 동일한 SQL에 대해 결과를 낼 수 있는 다양한 처리 방법(실행계획)이 존재할 수 있지만 각 처리 방법마다 실행 시간(성능)은 서로 다를 수 있다.
옵티マイ저는 다양한 처리 방법들 중에서 가장 효율적인 방법을 찾아준다.

3. SQL 처리 흐름도

- “SQL의 내부적인 처리절차를 시각적으로 표현한 도표”



[그림 II-3-5] SQL 처리 흐름도

제2절 인덱스 기본

1. ★ 인덱스 특징과 종류

- “원하는 데이터를 쉽게 찾을 수 있도록 돋는 책의 찾아보기와 유사한 기능”

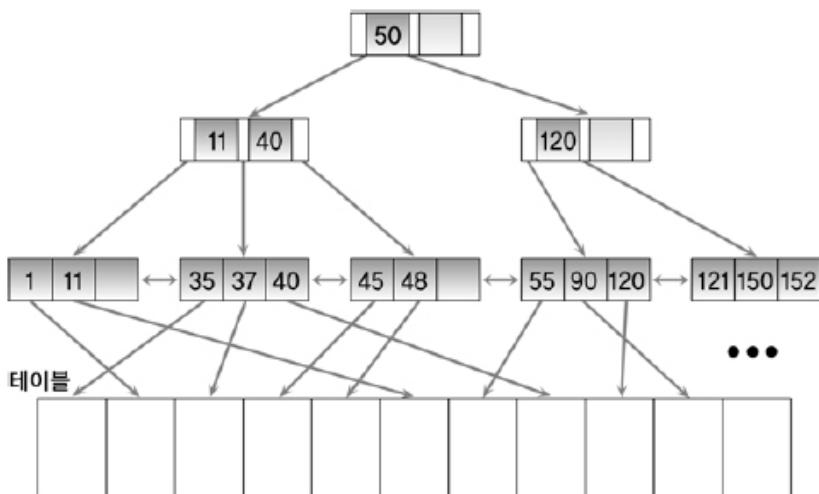
DML작업은 테이블과 인덱스를 함께 변경해야하기 때문에 느려질 수 있다

가. ★ 트리기반 인덱스

- DBMS에서 가장 일반적인 인덱스는 B-트리인덱스

- 리프블록은 인덱스를 구성하는 칼럼의 데이터와 해당데이터를 가지고 있는 행의 위치를 가리키는 레코드식별자로(RID, Record Identifier/Rowid) 구성되어있다.

‘=’로 검색하는 일치 검색과 ‘BETWEEN’등의 범위 검색 모두 적합함



[그림 II-3-7] B-트리 인덱스 검색

나. ★ SQL Server의 클러스터형 인덱스

- 클러스터형인덱스, 비클러스터형인덱스

- 클러스터형인덱스의 2가지 중요성

: 인덱스의 리프페이지가 곧 데이터페이지

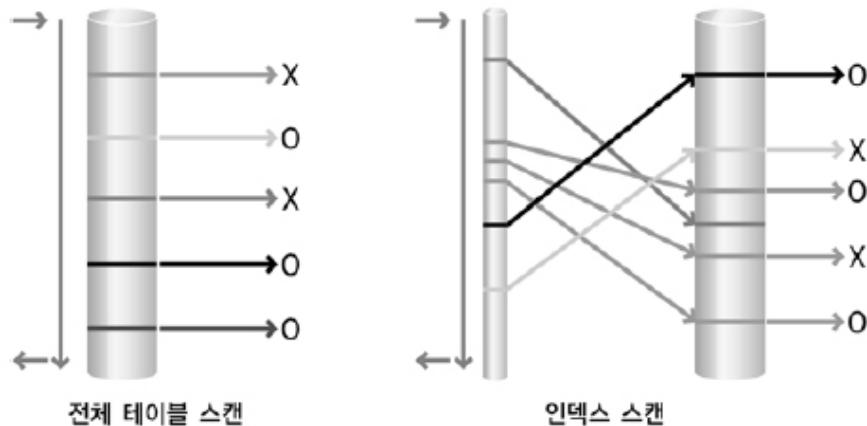
: 리프페이지의 모든 로우는 인덱스키칼럼순으로 물리적으로 정렬되어 저장됨

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

2. 전체 테이블 스캔과 인덱스 스캔

- 가. 전체 테이블 스캔
- 나. 인덱스 스캔
- 다. 전체테이블스캔과 인덱스 스캔방식의 비교

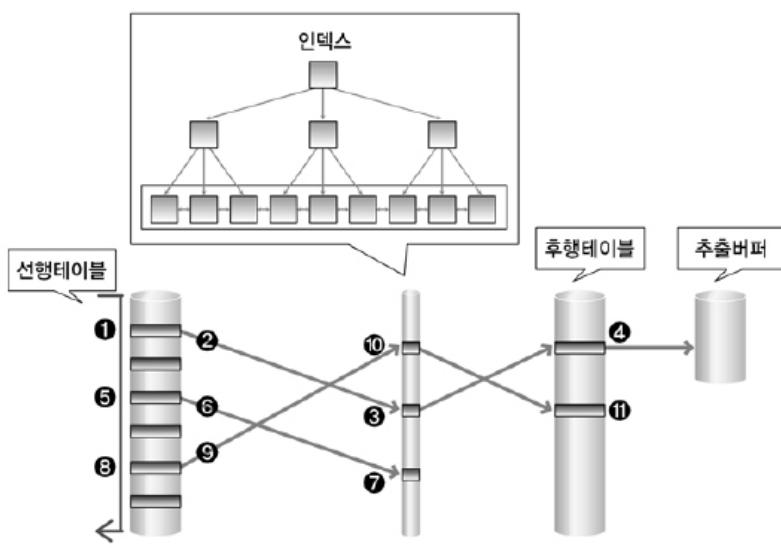


[그림 II-3-11] 전체 테이블 스캔과 인덱스 스캔에 대한 SQL 처리 흐름도 표현 예시

제3절 ★ 조인 수행 원리

1. ★ NL Join

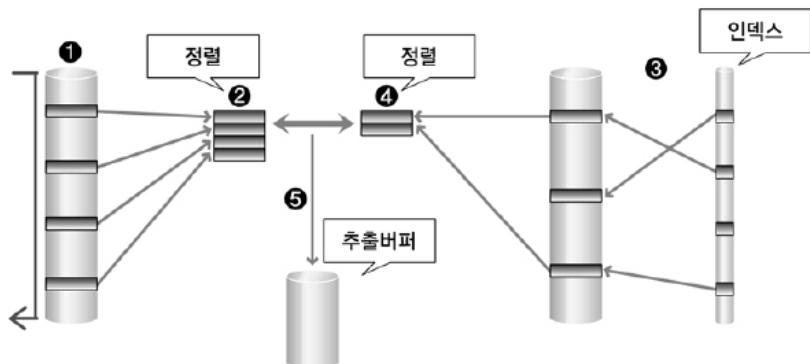
- NL Join은 프로그래밍에서 사용하는 중첩된 반복문과 유사한 방식으로 조인을 수행한다.
- 반복문의 외부에 있는 테이블을 선행 테이블 또는 외부 테이블(Outer Table)이라고 하고,
- 반복문의 내부에 있는 테이블을 후행 테이블 또는 내부 테이블(Inner Table)이라고 한다.
- 선행 테이블 또는 외부 테이블 → 후행 테이블 또는 내부 테이블
- 결과 행의 수가 적은 테이블을 조인 순서상 선행 테이블로 선택하는 것이 전체 일량을 줄임
- 조인이 성공하면 바로 조인 결과를 사용자에게 보여줌으로 온라인 프로그램에 적당



[그림 II-3-12] NL Join

2. ★ Sort Merge Join

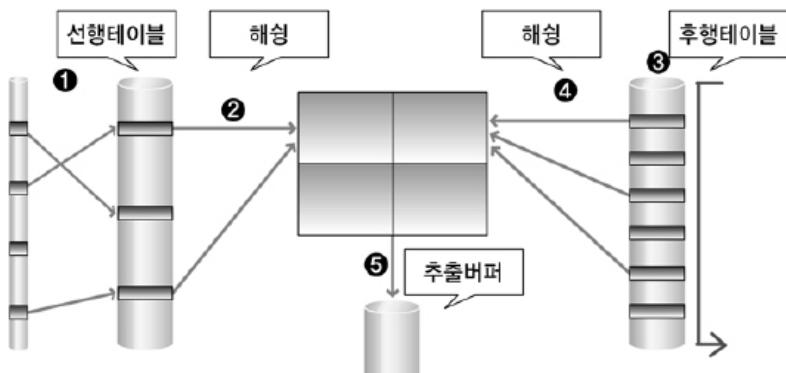
- 주로 스캔하는 방식으로 데이터를 읽음. 조인 칼럼 인덱스 없어도 사용가능 -> 단, 성능이 떨어질 수 있음
 - 조인 칼럼을 기준으로 데이터를 정렬하여 조인을 수행한다.
- NL Join은 주로 랜덤 액세스 방식으로 데이터를 읽는 반면
- Sort Merge Join은 주로 스캔 방식으로 데이터를 읽는다.
- Sort Merge Join은 랜덤 액세스로 NL Join에서 부담이 되던 넓은 범위의 데이터를 처리할 때 이용되던 조인 기법이다.



[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 같은지를 비교하면서 조인을 수행한다.
- Hash Join은 NL Join의 랜덤 액세스 문제점과 Sort Merge Join의 문제점인 정렬 작업의 부담을 해결 위한 대안으로 등장하였다.
- '='로 수행하는 동등 조인에서만 사용할 수 있다.
- 결과 행의 수가 적은 테이블을 선행 테이블로 사용하는 것이 좋다.



[그림 II-3-14] Hash Join

(end)