* 1. 모델링의 이해

1. 모델링의 이해

모델링의 특징 : 추상화, 단순화, 명확화

다시 정의하면 ‘현실세계를 추상화, 단순화, 명확화하기 위해 일정한 표기법에 의해 표현하는 기법’

모델링의 세 가지 관점

1) 데이터 관점 : 업무가 어떤 데이터와 관련이 있는지 또는 데이터 간의 관계는 무엇인지에 대해서 모델링하는 방법(What, Data)

2) 프로세스관점: 업무가 실제하고 있는 일은 무엇인지 또는 무엇을 해 야 하는지를 모델링하는 방법(How, Process)

3) 데이터와 프로세스의 상관 관점 : 업무가 처리하 는 일의 방법에 따라 데이터는 어떻게 영향을 받고 있는지 모델링하는 방법(Interaction)

데이터 모델링이란

- 정보시스템을 구축하기 위한 데이터 관점의 업무 분석 기법

- 현실세계의 데이터에 대해 약속된 표기법에 의해 표현하는 과정

- 데이터베이스를 구축하기 위한 분석/설계의 과정

데이터 모델링의 중요성

파급효과, 복잡한 정보 요구사항의 간결한 표현, 데이터 품질

데이터 모델링의 유의점

중복, 비유연성, 비일관성

데이터 모델링 3단계

1. 개념적 데이터 모델링

- 추상화 수준이 높고 업무 중심적이고 포괄적인 수준의 모델링 진행, 전사적 데이터 모델링, EA수립시 많이 이용

2. 논리적 데이터 모델링

- 시스템으로 구축하고자 하는 업무에 대해 Key, 속성, 관계 등을 정확하게 표현, 재사용성이 높음

3. 물리적 데이터 모델링

- 실제로 데이터베이스에 이식할 수 있도록 성능, 저장 등 물리적인 성격을 고려하여 설계

데이터 독립성 필요 -> 유지보수 비용증가, 데이터 중복성 증가,

데이터복잡도 증가, 요구사항 대응 저하

데이터 독립성 요소

외부스키마 : 사용자 관점, 접근하는 특성에 따른 스키마 구성

개념스키마 : 통합관점

내부스키마 : 물리적 저장 구조

* 논리적 독립성 : 개념 스키마가 변경되어도 외부 스키마에는 영향을 미치지 않도록

사용자 특성에 맞는 변경 가능

* 물리적 독립성 : 내부 스키마가 변경되어도 외부/개념 스키마는 영향을 받지 않도록

물리적 구조, 개념 구조가 서로 영향 없이 변경 가능

외부적/개념적 사상(논리적 사상) : 외부의 뷰와 개념적 뷰의 상호 관련성을 정의

개념적/내부적 사상(물리적 사상) : 개념적 뷰와 저장된 DB의 상호 관련성을 정의

데이터 모델링의 세 가지 요소 : Things, Attributes, Relationships



ERD 작업순서

① 엔터티를 그린다. ② 엔터티를 적절하게 배치한다. ③ 엔터티간 관계를 설정한다. ④ 관계명을 기술한다. ⑤ 관계의 참여도를 기술한다. ⑥ 관계의 필수여부를 기술한다.

좋은 데이터 모델의 요소

완전성, 중복배제, 업무규칙, 데이터 재사용, 의사소통, 통합성

2. 엔터티

“업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 집합적인 것(Thing)”

엔터티의 특정

- 반드시 해당 업무에서 필요하고 관리하고자 하는 정보이어야 한다. (ex 토익 응시 횟수)

- 유일한 식별자에 의해 식별이 가능해야 한다.

- 영속적으로 존재하는 인스턴스의 집합 이어야 한다. (3개 이상)

- 엔터티는 업무 프로세스에 의해 이용 되어야 한다.

- 엔터티는 다른 엔터티와 최소 한 개 이상의 관계가 있어야 한다.

엔터티의 분류

1. 유무형에 따른 분류

1. 유형엔터티(물품,사원) 2. 개념엔터티(조직,보험), 3. 사건엔터티(주문,청구,미납)

2. 발생시점에 따른 분류

1. 기본엔터티(사원,부서,고객) 2. 중심엔터티(계약,주문) 3. 행위엔터티(주문목록)

3. 속성

엔터티, 인스턴스, 속성, 속성값의 관계

- 한 개의 엔터티는 두 개 이상의 인스턴스의 집합이어야 한다.

- 한 개의 엔터티는 두 개 이상의 속성을 갖는다.

- 한 개의 속성은 한 개의 속성값을 갖는다.

속성의 특성에 따른 분류

- 기본속성

업무로부터 추출한 모든 속성이 여기에 해당하며 엔터티에 가장 일반적이고 많은 속성 을 차지한다. 코드성 데이터, 엔터티를 식별하기 위해 부여된 일련번호, 그리고 다른 속성을 계산하 거나 영향을 받아 생성된 속성을 제외한 모든 속성은 기본속성

- 설계속성

설계속성은 업무상 필요한 데이터 이외에 데이터 모델링을 위해, 업무를 규칙화하기 위해 속성을 새 로 만들거나 변형하여 정의하는 속성

- 파생속성

파생속성은 다른 속성에 영향을 받아 발생하는 속성으로서 보통 계산된 값들이 이에 해당

엔터티 구성방식에 따른 분류

PK, FK 일반속성(PK,FK)를 제외한 나머지

각 속성은 가질 수 있는 값의 범위가 있는데 이를 그 속성의 도메인 (Domain)이라 하며, 엔터티 내에서 속성에 대한 데이터타입과 크기 그리고 제약사항을 지정하는 것이다.

속성의 명칭 부여

- 해당업무에서 사용하는 이름을 부여 한다.

- 서술식 속성명은 사용하지 않는다.

- 약어사용은 가급적 제한한다.

- 전체 데이터모델에서 유일성 확보하는 것이 좋다.

4. 관계

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

ERD에서는 존재적 관계와 행위에 의한 관계를 구분하지 않지만 클래스다이어그램에서는 이것을 구분하여 연관관계와 의존 관계로 표현한다.

관계의 분류

1. 존재에 의한 관계 : 부서-사원 -> 소속한다.
2. 행위에 의한 관계 : 고객-주문 -> 주문한다.

관계의 표기법

- 관계명(Membership) : 관계의 이름

- 관계차수(Cardinality) : 1:1, 1:M, M:N

- rhksrPtjsxortkdid (Optionality) : 필수관계 선택관계

관계 읽기

- 기준(Source) 엔터티를 한 개(One) 또는 각 (Each)으로 읽는다.

- 대상(Target) 엔터티의 관계참여도 즉 개수(하나, 하나 이상)를 읽는다.

- 관계선택사양과 관계명을 읽는다.



5. 식별자

하나의 엔터티에 구성되어 있는 여러 개의 속성 중에 엔터티를 대표할 수 있는 속성을 의미하며 하 나의 엔터티는 반드시 하나의 유일한 식별자가 존재해야 한다.

식별자라는 용어는 업무적으로 구분이 되는 정보로 생각할 수 있으므로 논리 데이터 모델링 단계에서 사용하고 키는 데이터베이스 테이블에 접근을 위한 매개체로서 물리 데이터 모델링 단계에서 사용한다.

주식별자의 특징

- 유일성 : 주식별자에 의해 엔터티내에 모든 인스턴스들을 유일하게 구분함

- 최소성 : 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함

- 불변성 : 주식별자가 한 번 특정 엔터티에 지정되면 그 식별자의 값은 변하지 않아야 함

- 존재성 : 주식별자가 지정되면 반드시 데이터 값이 존재 (Null 안됨)

식별자의 종류

- 엔터티 내에서 대표성을 가지는가에 따라 주식별자 (Primary Identifier)와 보조식별자 (Alternate Identitifer)로 구분

- 엔터티 내에서 스스로 생성되었는지 여부에 따라 내부식별자와 외부식별자(Foreign Identifier)로 구분

- 단일 속성으로 식별이 되는가에 따라 단일식별자 (Single Identifier)와 복합식별자(Composit Identifier)로 구분

- 원래 업무적으로 의미가 있던 식별자 속성을 대체하여 일련번호와 같이 새롭게 만든 식별자를 구분하기 위해 본직식별자와 인조식별자로 구분

* 1. 데이터 모델과 성능

성능 데이터 모델링 : DB 성능 향상을 목적으로 설계 단계의 데이터 모델링 때부터 성능과 관련된 사항이 데이터 모델링에 반영될 수 있도록 하는 것

수행 시점 : 빠를수록 좋다.

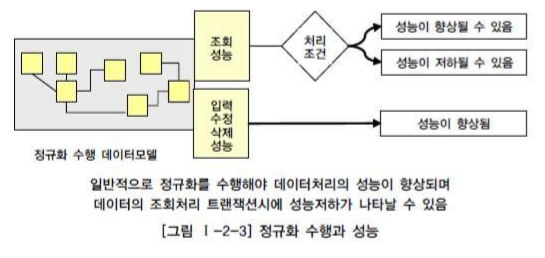
고려사항

1. 데이터 모델링을 할 때 정규화를 정확하게 수행한다.
2. 데이터베이스 용량산정을 수행한다.
3. 데이터베이스에 발생되는 트랜잭션의 유형을 파악한다.
4. 용량과 트랜잭션의 유형에 따라 반정규화를 수행한다.
5. 이력모델의 조정, PK/FK 조정, 슈퍼타입/서브타입 조정 등을 수행한다.
6. 성능관점에서 데이터 모델을 검증한다.

1. 정규화와 성능

데이터베이스에서 데이터를 처리할 때 성능이라고 하면 1.조회 성능과 2.입력/수정/삭제 성능의 두 부류로 구분된다. 이 두 가지 성능이 모두 우수하면 좋겠지만 데이터 모델을 구성하는 방식에 따라 두 성능이 Trade-Off 되어 나타나는 경우가 많이 있다.

정규화를 수행한다는 것은 데이터를 결정하는 결정자에 의해 함수적 종속을 가지고 있는 일반속성을 의존자로 하여 입력/수정/삭제 이상을 제거하는 것이다. 데이터의 중복속성을 제거하고 결정자에 의해 동일한 의미의 일반속성이 하나의 테이블로 집약되므로 한 테이블의 데이터 용량이 최소화되는 효과가 있다. 따라서 정규화된 테이블은 데이터를 처리할 때 속도가 빨라질 수도 있고 느려질 수도 있는 특성이 있다.

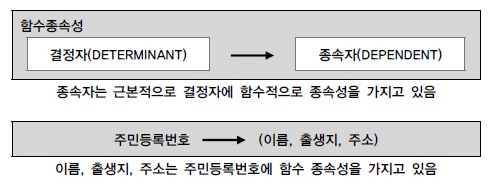


1차 정규화 : 어떤 속성이 다수값 존재시 M:1관계로 새로 엔터티 추가

2차 정규화 : 식별자 아닌 전체 속성은 식별자 전체 속성에 완전 종속

3차 정규화 : 식별자 제외한 속성간에 종속이 존재하면 안됨

**함수적 종속성(Functional Dependency)에 근거한 정규화 수행 필요**



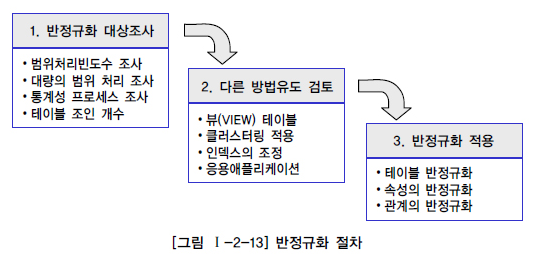
2. 반정규화와 성능

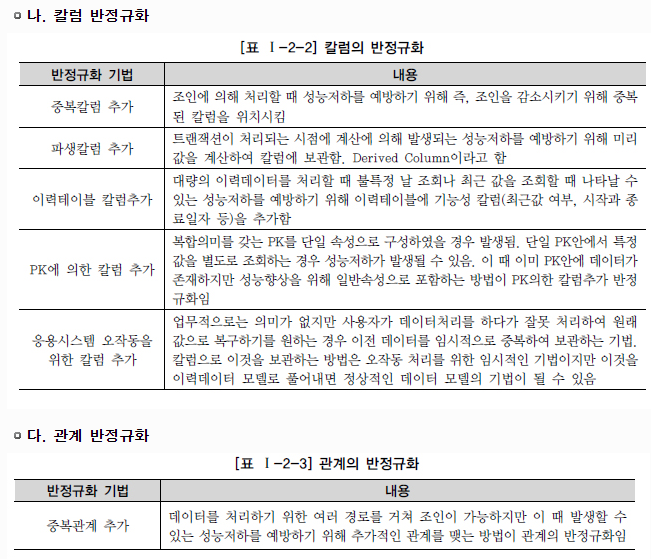
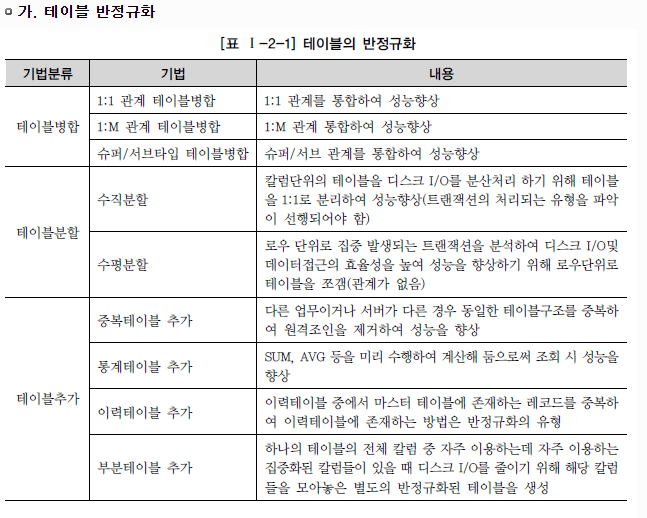
반정규화를 정의하면 정규화된 엔터티, 속성, 관계에 대해 시스템의 성능향상과 개발(Development)과 운영(Maintenance)의 단순화를 위해 중복, 통합, 분리 등을 수행하는 데이터 모델링의 기법

정규화만을 수행하면 엔터티의 갯수가 증가하고 관계가 많아져 일부 여러 개의 조인이 걸려야만 데이터를 가져오는 경우가 있다. 이러한 경우 업무적으로 조회에 대한 처리성능이 중요하다고 판단될 때 부분적으로 반정규화를 고려

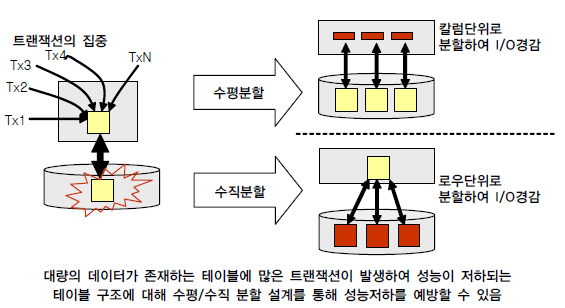
반정규화를 기술적으로 수행하지 않는 경우, 1. 성능이 저하된 데이터베이스가 생성 2. 구축단계나 시험단계에서 반정규화를 적용할 때 수정에 따른 노력비용이 많이 들게 된다.

반정규화를 적용할 때는 기본적으로 데이터 무결성이 깨질 가능성이 많이 있기 때문에 반드시 데이터 무결성을 보장할 수 있는 방법을 고려한 이후에 반정규화를 적용





3. 대량 데이터에 따른 성능



수평 분할 -> 행을 기준으로 분할 수직 분할 -> 열을 기준으로 분할

한 테이블에 많은 수의 칼럼을 가지고 있는 경우

이렇게 많은 칼럼을 가지고 있는 테이블에 대해서는 트랜잭션이 발생될 때 어떤 칼럼에 대해 집중적으로 발생하는지 분석하여 테이블을 쪼개어 주면 디스크 I/O가 감소하게 되어 성능이 개선

대량 데이터 저장 및 처리로 인한 성능

Oracle의 경우 크게 LIST PARTITION(특정값 지정), RANGE PARTITION(범위), HASH PARTITION(해쉬적용), COMPOSITE PARTITION(범위와 해쉬가 복합) 등이 가능

RANGE PARTITION : 하나의 테이블에 접근하면 내부적으로 RANGE로 구분된 테이블에서 트랜잭션 처리 => 요금1(1000만건), 요금2(1000만건), …

LIST PARTITION : 하나의 테이블에 접근하면 내부적으로 LIST로 구분된 파티션테이블에서 트랜잭션 처리 => 고객\_서울(1000만건), 고객\_제주(100만건), …

HASHING PARTITION : 지정된 HASH 조건에 따라 해슁 알고리즘이 적용되어 테이블이 분리되며 설계자는 테이블에 데이터가 정확하게 어떻게 들어갔는지 알 수 없다. 역시 성능향상을 위해 사용하며 데이터 보관주기에 따라 쉽게 삭제하는 기능은 제공될 수 없다.

테이블에 대한 수평분할/수직분할의 절차

1) 데이터 모델링을 완성한다.  
2) 데이터베이스 용량산정을 한다.  
3) 대량 데이터가 처리되는 테이블에 대해서 트랜잭션 처리 패턴을 분석한다.  
4) 칼럼 단위로 집중화된 처리가 발생하는지, 로우단위로 집중화된 처리가 발생되는지 분석하여 집중화된 단위로 테이블을 분리하는 것을 검토한다.

4. 데이터베이스 구조와 성능

슈퍼타입 모델링 : 사원 테이블

장점: 전체 사원에 대한 검색이 쉽다, 무결성에 유리하다.

단점: 서브타입별 속성이 많다면 지나치게 NULL이 많이 발생한다.(공간의 낭비)

서브타입 모델링 : 정규직 테이블 + 임시직 테이블

장점: NULL이 없거나 줄어들고(공간의 낭비가 적다), 정규직 사원과 임시직 사원을 별개로 처리하는 업무가 많을 때 효율이 좋다.

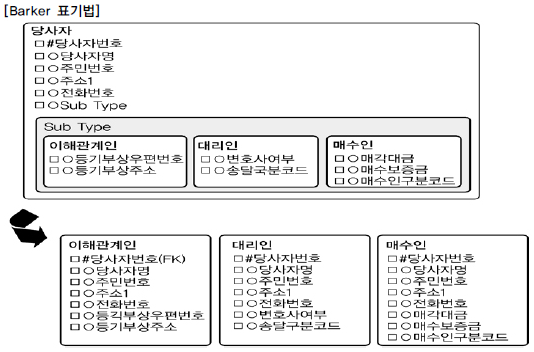
단점: UNION 함수를 쓸 때 중복제거 되기 때문에, PK의 유일성(Unique)을 잘 관리해야 한다. INTERSECT 연산 시 양쪽 테이블에 중복 사원이 안 나오도록 무결성을 유지하는 것이 중요하다. 비효율적인 조인이나 합집합, 교집합이 발생할 수 있다.(속도 저하)

슈퍼/서브타입 모델 => 1:1 타입, 슈퍼+서브 타입, All In One 타입

데이터의 양과 해당 테이블에 발생되는 트랜잭션의 유형에 따라

1) 개별로 발생되는 트랜잭션에 대해서는 개별 테이블로 구성

2) 슈퍼타입+서브타입에 대해 발생되는 트랜잭션에 대해서는 슈퍼타입+서브타입 테이블로 구성



3) 전체를 하나로 묶어 트랜잭션이 발생할 때는 하나의 테이블로 구성

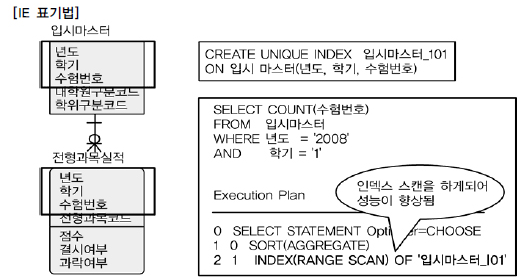
인덱스 특성을 고려한 PK/FK 데이터베이스 성능 향상

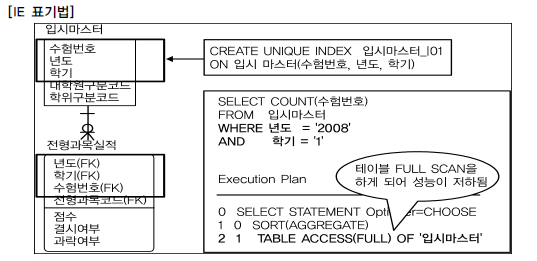
PK순서를 결정하는 기준은 인덱스 정렬구조를 이해한 상태에서 인덱스를 효율적으로 이용할 수 있도록 PK순서를 지정해야 한다.

**PK칼럼의 순서를 조정하지 않으면 성능이 저하되는 이유**

데이터 모델링에서 엔터티를 설계하면 그에 따라 DDL이 생성이 되고 생성된 DDL에 따라 인덱스가 생성 => 정렬 순서가 결정

정렬 순서를 바꿔주면 성능이 올라갈 수 있다.

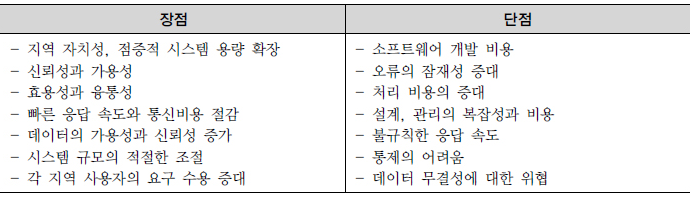




5. 분산 데이터베이스와 성능

분산데이터베이스가 되기 위해서는 6가지 투명성(Transparency)을 만족해야 한다.

1) 분할 투명성(단편화) : 하나의 논리적 Relation이 여러 단편으로 분할되어 각 단편의 사본이 여러 site에 저장  
2) 위치 투명성 : 사용하려는 데이터의 저장 장소 명시 불필요. 위치정보가 System Catalog에 유지되어야 함  
3) 지역사상 투명성 : 지역DBMS와 물리적 DB사이의 Mapping 보장. 각 지역시스템 이름과 무관한 이름 사용 가능  
4) 중복 투명성 : DB 객체가 여러 site에 중복 되어 있는지 알 필요가 없는 성질  
5) 장애 투명성 : 구성요소(DBMS, Computer)의 장애에 무관한 Transaction의 원자성 유지  
6) 병행 투명성 : 다수 Transaction 동시 수행시 결과의 일관성 유지, Time Stamp, 분산 2단계 Locking을 이용 구현



데이터를 분산 환경으로 구성하였을 때 가장 핵심적인 가치는 바로 통합된 데이터베이스에서 제공할 수 없는 빠른 성능을 제공한다는 점이다. 원거리 또는 다른 서버에 접속하여 처리하므로 인해 발생되는 네트워크 부하 및 트랜잭션 집중에 따른 성능 저하의 원인을 분산된 데이터베이스 환경을 구축하므로 빠른 성능을 제공하는 것이 가능

분산 데이터베이스의 적용 기법

테이블 위치 분산

테이블 분할 분산(수평, 수직)

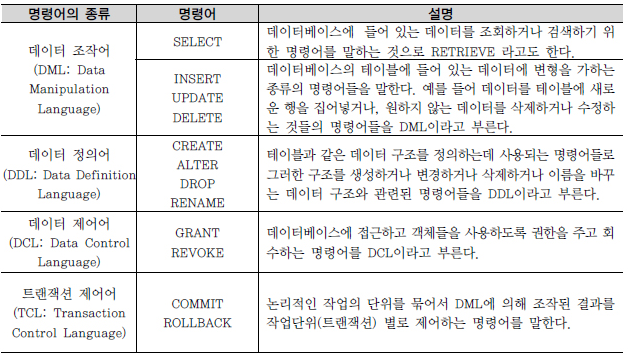
테이블 복제 분산(부분, 광역)

테이블 요약 분산

2-1. SQL 기본

1. 개요

SQL : Structured Query Language



테이블 : 행과 칼럼의 2차원 구조를 가진 데이터의 저장 장소

칼럼/열 : 2차원 구조를 가진 테이블에서 세로 방향으로 이루어진 하나하나의 특정 속성

행 : 2차원 구조를 가진 테이블에서 가로 방향으로 이루어진 연결된 데이터

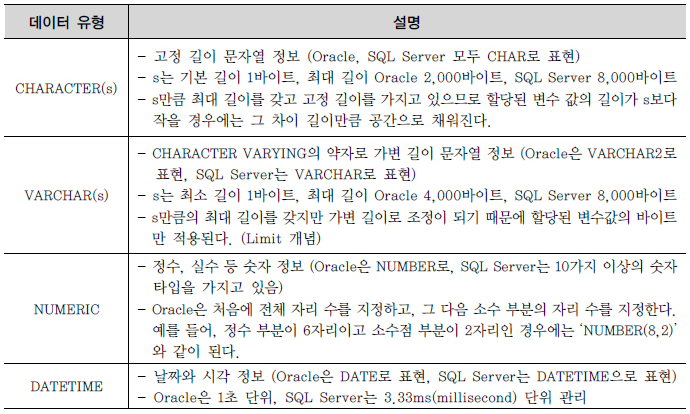
정규화 : 테이블을 분할하여 데이터의 정합성을 확보하고, 불필요한 중복을 줄이는 프로세스

기본 키(PK) : 테이블에 존재하는 각 행을 한 가지 의미로 특정할 수 있는 한 개 이상의 칼럼

외부 키(FK) : 다른 테이블의 기본 키로 사용되고 있는 관계를 연결하는 칼럼

ERD : Entity Relationship Diagram => IE 표기법, Barker 표기법

2. DDL(데이터 정의어)



CREATE TABLE

CREATE　TABLE　테이블이름 ( 칼럼명1 DATATYPE [DEFAULT 형식], 칼럼명2 DATATYPE [DEFAULT 형식], 칼럼명2 DATATYPE [DEFAULT 형식] ) ;

- 테이블명은 객체를 의미할 수 있는 적절한 이름을 사용한다. 가능한 단수형을 권고한다.

- 테이블 명은 다른 테이블의 이름과 중복되지 않아야 한다.

- 한 테이블 내에서는 칼럼명이 중복되게 지정될 수 없다.

- 테이블 이름을 지정하고 각 칼럼들은 괄호 "( )" 로 묶어 지정한다.

- 각 칼럼들은 콤마 ","로 구분되고, 테이블 생성문의 끝은 항상 세미콜론 ";"으로 끝난다.

- 칼럼에 대해서는 다른 테이블까지 고려하여 데이터베이스 내에서는 일관성 있게 사용하는 것이 좋다.(데이터 표준화 관점)

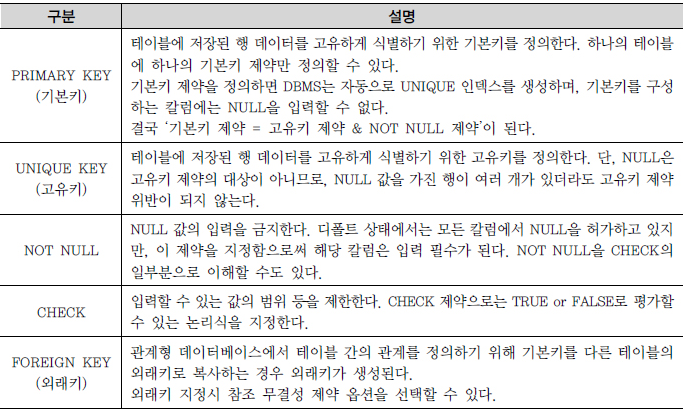
- 칼럼 뒤에 데이터 유형은 꼭 지정되어야 한다.

- 테이블명과 칼럼명은 반드시 문자로 시작해야 하고, 벤더별로 길이에 대한 한계가 있다.

- 벤더에서 사전에 정의한 예약어(Reserved word)는 쓸 수 없다.

- A-Z, a-z, 0-9, \_, $, # 문자만 허용된다.

제약 조건



* NULL 의미

NULL(ASCII 코드 00번)은 공백(BLANK, ASCII 코드 32번)이나 숫자 0(ZERO, ASCII 48)과는 전혀 다른 값이며, 조건에 맞는 데이터가 없을 때의 공집합과도 다르다. ‘NULL’은 ‘아직 정의되지 않은 미지의 값’이거나 ‘현재 데이터를 입력하지 못하는 경우’를 의미한다.

* DEFAULT 의미

데이터? 기본값(DEFAULT)을 사전에 설정할 수 있다. 데이터 입력시 명시된 값을 지정하지 않은 경우에 NULL 값이 입력되고, DEFAULT 값을 정의했다면 해당 칼럼에 NULL 값이 입력되지 않고 사전에 정의된 기본 값이 자동으로 입력된다.

[예제] Oracle CREATE TABLE TEAM ( TEAM\_ID CHAR(3) NOT NULL, REGION\_NAME VARCHAR2(8) NOT NULL, TEAM\_NAME VARCHAR2(40) NOT NULL, E\_TEAM\_NAME VARCHAR2(50), ORIG\_YYYY CHAR(4), STADIUM\_ID CHAR(3) NOT NULL, ZIP\_CODE1 CHAR(3), ZIP\_CODE2 CHAR(3), ADDRESS VARCHAR2(80), DDD VARCHAR2(3), TEL VARCHAR2(10), FAX VARCHAR2(10), HOMEPAGE VARCHAR2(50), OWNER VARCHAR2(10), CONSTRAINT TEAM\_PK PRIMARY KEY (TEAM\_ID), CONSTRAINT TEAM\_FK FOREIGN KEY (STADIUM\_ID) REFERENCES STADIUM(STADIUM\_ID) ); 테이블이 생성되었다.

[예제] SQL Server CREATE TABLE TEAM ( TEAM\_ID CHAR(3) NOT NULL, REGION\_NAME VARCHAR(8) NOT NULL, TEAM\_NAME VARCHAR(40) NOT NULL, E\_TEAM\_NAME VARCHAR(50), ORIG\_YYYY CHAR(4), STADIUM\_ID CHAR(3) NOT NULL, ZIP\_CODE1 CHAR(3), ZIP\_CODE2 CHAR(3), ADDRESS VARCHAR(80), DDD VARCHAR(3), TEL VARCHAR(10), FAX VARCHAR(10), HOMEPAGE VARCHAR(50), OWNER VARCHAR(10), CONSTRAINT TEAM\_PK PRIMARY KEY (TEAM\_ID), CONSTRAINT TEAM\_FK FOREIGN KEY (STADIUM\_ID) REFERENCES STADIUM(STADIUM\_ID) ); 테이블이 생성되었다.

생성된 테이블 구조 확인

Oracle의 경우 “DESCRIBE 테이블명;” 또는 간략히 “DESC 테이블명;”

SQL Server의 경우 “exec sp\_help ‘dbo.테이블명’”

ALTER TABLE

1. ADD COLUMN

[예제] Oracle ALTER TABLE PLAYER ADD (ADDRESS VARCHAR2(80));

[예제] SQL Server ALTER TABLE PLAYER ADD ADDRESS VARCHAR(80);

2. DROP COLUMN

[예제] Oracle ALTER TABLE PLAYER DROP COLUMN ADDRESS;

[예제] SQL Server ALTER TABLE PLAYER DROP COLUMN ADDRESS;

3. MODIFY COLUMN

[Oracle] ALTER TABLE 테이블명 MODIFY (칼럼명1 데이터 유형 [DEFAULT 식] [NOT NULL],

칼럼명2 데이터 유형 …);

[SQL Server] ALTER TABLE 테이블명 ALTER (칼럼명1 데이터 유형 [DEFAULT 식] [NOT NULL], 칼럼명2 데이터 유형 …);

- 해당 칼럼의 크기를 늘릴 수는 있지만 줄이지는 못한다. 이는 기존의 데이터가 훼손될 수 있기 때문이다.

- 해당 칼럼이 NULL 값만 가지고 있거나 테이블에 아무 행도 없으면 칼럼의 폭을 줄일 수 있다.

- 해당 칼럼이 NULL 값만을 가지고 있으면 데이터 유형을 변경할 수 있다.

- 해당 칼럼의 DEFAULT 값을 바꾸면 변경 작업 이후 발생하는 행 삽입에만 영향을 미치게 된다.

- 해당 칼럼에 NULL 값이 없을 경우에만 NOT NULL 제약조건을 추가할 수 있다.

4. DROP COSTRAINT

ALTER TABLE 테이블명 DROP CONSTRAINT 제약조건명; 둘 다 동일

5. ADD COSTRAINT

ALTER TABLE 테이블명 ADD CONSTRAINT 제약조건명 제약조건 (칼럼명); 둘 다 동일

RENAME TABLE

[예제] Oracle RENAME TEAM TO TEAM\_BACKUP;

[예제] SQL Server sp\_rename 'dbo.TEAM','TEAM\_BACKUP'; 주의: 엔터티 이름 부분을 변경하면 스크립트 및 저장 프로시저를 손상시킬 수 있다.

DROP TABLE

DROP TABLE 테이블명 [CASCADE CONSTRAINT]; 둘 다 동일

CASCADE CONSTRAINT 옵션은 해당 테이블과 관계가 있었던 참조되는 제약조건에 대해서도 삭제한다는 것

TRUNCATE TABLE

TRUNCATE TABLE은 테이블 자체가 삭제되는 것이 아니고, 해당 테이블에 들어있던 모든 행들이 제거되고 저장 공간을 재사용 가능하도록 해제한다. 테이블 구조를 완전히 삭제하기 위해서는 DROP TABLE을 실행하면 된다.

TRUNCATE TABLE PLAYER; 둘 다 동일

3. DML(데이터 조작어)

INSERT

▶ INSERT INTO 테이블명 (COLUMN\_LIST)VALUES (COLUMN\_LIST에 넣을 VALUE\_LIST); ▶ INSERT INTO 테이블명VALUES (전체 COLUMN에 넣을 VALUE\_LIST);

 첫 번째 유형은 테이블의 칼럼을 정의할 수 있는데, 이때 칼럼의 순서는 테이블의 칼럼 순서와 매치할 필요는 없으며, 정의하지 않은 칼럼은 Default로 NULL 값이 입력된다. 단, Primary Key나 Not NULL 로 지정된 칼럼은 NULL이 허용되지 않는다. 두 번째 유형은 모든 칼럼에 데이터를 입력하는 경우로 굳이 COLUMN\_LIST를 언급하지 않아도 되지만, 칼럼의 순서대로 빠짐없이 데이터가 입력되어야 한다.

UPDATE

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

DELETE

DELETE [FROM] 삭제를 원하는 정보가 들어있는 테이블명;

 데이터베이스는 DDL 명령어와 DML 명령어를 처리하는 방식에 있어서 차이를 보인다. DDL(CREATE, ALTER, RENAME, DROP) 명령어인 경우에는 직접 데이터베이스의 테이블에 영향을 미치기 때문에 DDL 명령어를 입력하는 순간 명령어에 해당하는 작업이 즉시(AUTO COMMIT) 완료된다. 하지만 DML(INSERT, UPDATE, DELETE, SELECT) 명령어의 경우, 조작하려는 테이블을 메모리 버퍼에 올려놓고 작업을 하기 때문에 실시간으로 테이블에 영향을 미치는 것은 아니다. 따라서 버퍼에서 처리한 DML 명령어가 실제 테이블에 반영되기 위해서는 COMMIT 명령어를 입력하여 TRANSACTION을 종료해야 한다.

그러나 SQL Server의 경우는 DML의 경우도 AUTO COMMIT으로 처리되기 때문에 실제 테이블에 반영하기 위해 COMMIT 명령어를 입력할 필요가 없다.

테이블의 전체 데이터를 삭제하는 경우, 시스템 활용 측면에서는 삭제된 데이터를 로그로 저장하는 DELETE TABLE 보다는 시스템 부하가 적은 TRUNCATE TABLE을 권고한다. 단, TRUNCATE TABLE의 경우 삭제된 데이터의 로그가 없으므로 ROLLBACK이 불가능하므로 주의해야 한다. 그러나 SQL Server의 경우 사용자가 임의적으로 트랜잭션을 시작한 후 TRUNCATE TABLE을 이용하여 데이터를 삭제한 이후 오류가 발견되어, 다시 복구를 원할 경우 ROLLBACK 문을 이용하여 테이블 데이터를 원 상태로 되돌릴 수 있다.

SELECT

- ALL : Default 옵션이므로 별도로 표시하지 않아도 된다. 중복된 데이터가 있어도 모두 출력한다. - DISTINCT : 중복된 데이터가 있는 경우 1건으로 처리해서 출력한다.

[예제] SELECT PLAYER\_ID, PLAYER\_NAME, TEAM\_ID, POSITION, HEIGHT, WEIGHT, BACK\_NO FROM PLAYER;

합성 연산자

- 문자와 문자를 연결하는 경우 2개의 수직 바(||)에 의해 이루어진다. (Oracle)

- 문자와 문자를 연결하는 경우 + 표시에 의해 이루어진다. (SQL Server)

- 두 벤더 모두 공통적으로 CONCAT (string1, string2) 함수를 사용할 수 있다.

- 칼럼과 문자 또는 다른 칼럼과 연결시킨다.

- 문자 표현식의 결과에 의해 새로운 칼럼을 생성한다.

4.TCL(트랜잭션 제어어)

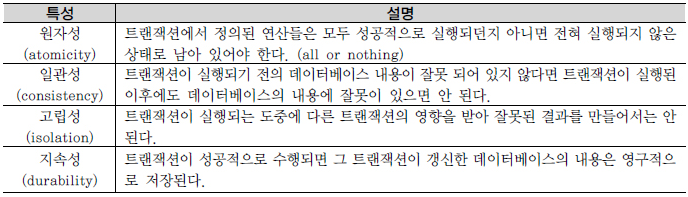
트랜잭션은 데이터베이스의 논리적 연산단위이다. 트랜잭션(TRANSACTION)이란 밀접히 관련되어 분리될 수 없는 한 개 이상의 데이터베이스 조작을 가리킨다. 하나의 트랜잭션에는 하나 이상의 SQL 문장이 포함된다. 트랜잭션은 분할할 수 없는 최소의 단위이다. 그렇기 때문에 전부 적용하거나 전부 취소한다. 즉, TRANSACTION은 ALL OR NOTHING의 개념인 것이다.

계좌이체 사례 - STEP1. 100번 계좌의 잔액에서 10,000원을 뺀다. - STEP2. 200번 계좌의 잔액에 10,000원을 더한다.

계좌이체라는 작업 단위는 이런 두 개의 업데이트가 모두 성공적으로 완료되었을 때 종료된다. 둘 중 하나라도 실패할 경우 계좌이체는 원래의 금액을 유지하고 있어야만 한다. 만약 어떠한 장애에 의해 어느 쪽이든 한 쪽만 실행했을 경우, 이체한 금액은 어디로 증발해 버렸거나 마음대로 증가하게 된다. 당연히 그런 일이 있어서는 안 되므로 이러한 경우에는 수정을 취소하여 원 상태로 되돌려야 한다. 이런 계좌이체 같은 하나의 논리적인 작업 단위를 구성하는 세부적인 연산들의 집합을 트랜잭션이라 한다.

트랜잭션의 대상이 되는 SQL문은 UPDATE, INSERT, DELETE 등 데이터를 수정하는 DML 문

트랜잭션의 특성



COMMIT

입력한 자료나 수정한 자료에 대해서 또는 삭제한 자료에 대해서 전혀 문제가 없다고 판단되었을 경우 COMMIT 명령어를 통해서 트랜잭션을 완료할 수 있다. COMMIT이나 ROLLBACK 이전의 데이터 상태는 다음과 같다.

- 단지 메모리 BUFFER에만 영향을 받았기 때문에 데이터의 변경 이전 상태로 복구 가능하다.

- 현재 사용자는 SELECT 문장으로 결과를 확인 가능하다.

- 다른 사용자는 현재 사용자가 수행한 명령의 결과를 볼 수 없다.

- 변경된 행은 잠금(LOCKING)이 설정되어서 다른 사용자가 변경할 수 없다.

Oracle은 DML을 실행하는 경우 DBMS가 트랜잭션을 내부적으로 실행하며 DML 문장 수행 후 사용자가 임의로 COMMIT 혹은 ROLLBACK을 수행해 주어야 트랜잭션이 종료된다. (일부 툴에서는 AUTO COMMIT을 옵션으로 선택할 수 있다.)

SQL Server는 기본적으로 AUTO COMMIT 모드이기 때문에 DML 수행 후 사용자가 COMMIT이나 ROLLBACK을 처리할 필요가 없다. DML 구문이 성공이면 자동으로 COMMIT이 되고 오류가 발생할 경우 자동으로 ROLLBACK 처리된다.

ROLLBACK

테이블 내 입력한 데이터나, 수정한 데이터, 삭제한 데이터에 대하여 COMMIT 이전에는 변경 사항을 취소할 수 있는데 데이터베이스에서는 롤백(ROLLBACK) 기능을 사용한다. 롤백(ROLLBACK)은 데이터 변경 사항이 취소되어 데이터의 이전 상태로 복구되며, 관련된 행에 대한 잠금(LOCKING)이 풀리고 다른 사용자들이 데이터 변경을 할 수 있게 된다.

Oracle과 SQL Server의 차이는 COMMIT과 동일하지만, SQL Server에서는 ROLLBACK을 명시적으로 작성하여야 수행된다.

COMMIT과 ROLLBACK을 사용함으로써 다음과 같은 효과를 볼 수 있다.

- 데이터 무결성 보장 - 영구적인 변경을 하기 전에 데이터의 변경 사항 확인 가능 - 논리적으로 연관된 작업을 그룹핑하여 처리 가능

SAVEPOINT

저장점(SAVEPOINT)을 정의하면 롤백(ROLLBACK)할 때 트랜잭션에 포함된 전체 작업을 롤백하는 것이 아니라 현 시점에서 SAVEPOINT까지 트랜잭션의 일부만 롤백할 수 있다. 따라서 복잡한 대규모 트랜잭션에서 에러가 발생했을 때 SAVEPOINT까지의 트랜잭션만 롤백하고 실패한 부분에 대해서만 다시 실행할 수 있다.

[예제] Oracle SAVEPOINT SVPT1; 저장점이 생성되었다. INSERT INTO PLAYER (PLAYER\_ID, TEAM\_ID, PLAYER\_NAME, POSITION, HEIGHT, WEIGHT, BACK\_NO) VALUES ('1999035', 'K02', '이운재', 'GK', 182, 82, 1); 1개의 행이 만들어졌다. ROLLBACK TO SVPT1; 롤백이 완료되었다.

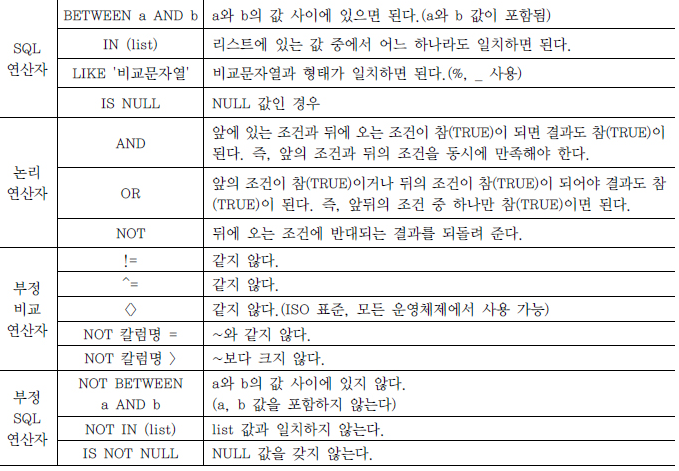
[예제] SQL Server SAVE TRAN SVTR1; 저장점이 생성되었다. INSERT INTO PLAYER (PLAYER\_ID, TEAM\_ID, PLAYER\_NAME, POSITION, HEIGHT, WEIGHT, BACK\_NO) VALUES ('1999035', 'K02', '이운재', 'GK', 182, 82, 1); 1개의 행이 만들어졌다. ROLLBACK TRAN SVTR1; 롤백이 완료되었다.

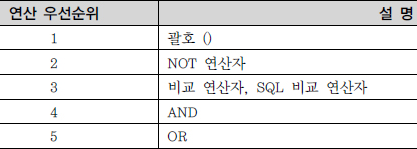
- CREATE, ALTER, DROP, RENAME, TRUNCATE TABLE 등 DDL 문장을 실행하면 그 전후 시점에 자동으로 커밋된다. - 부연하면, DML 문장 이후에 커밋 없이 DDL 문장이 실행되면 DDL 수행 전에 자동으로 커밋된다. - 데이터베이스를 정상적으로 접속을 종료하면 자동으로 트랜잭션이 커밋된다. - 애플리케이션의 이상 종료로 데이터베이스와의 접속이 단절되었을 때는 트랜잭션이 자동으로 롤백된다.

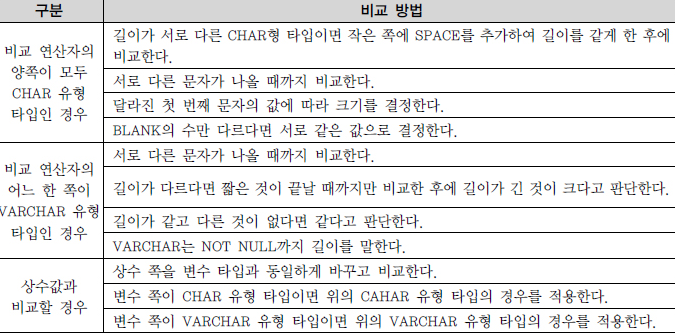
SQL Server의 트랜잭션은 DBMS가 트랜잭션을 컨트롤하는 방식인 AUTO COMMIT이 기본 방식이다. 다음의 경우는 Oracle과 같이 자동으로 트랜잭션이 종료된다.

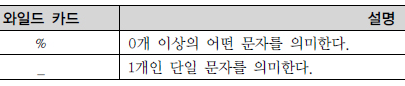
- 애플리케이션의 이상 종료로 데이터베이스(인스턴스)와의 접속이 단절되었을 때는 트랜잭션이 자동으로 롤백된다.

5. WHERE절









IS NULL 연산자

- NULL 값과의 수치연산은 NULL 값을 리턴한다. - NULL 값과의 비교연산은 거짓(FALSE)을 리턴한다. - 어떤 값과 비교할 수도 없으며, 특정 값보다 크다, 적다라고 표현할 수 없다.

따라서 NULL 값의 비교는 비교 연산자인 “=”, “>”, “>=”, “<”, “=”를 통해서 비교할 수도 없고, 만일 비교 연산을 하게 되면 결과는 거짓(FALSE)을 리턴하고, 수치 연산자(+,-,\*,/ 등)를 통해서 NULL 값과 연산을 하게 되면 NULL 값을 리턴한다. NULL 값의 비교 연산은 IS NULL, IS NOT NULL 이라는 정해진 문구를 사용해야 제대로 된 결과를 얻을 수 있다.

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.

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

TOP (Expression) [PERCENT] [WITH TIES]

- 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.

6. 내장 함수

단일행 함수, 다중행 함수(집계 함수, 그룹 함수, 윈도우 함수)

단일행 함수의 특징

- SELECT, WHERE, ORDER BY 절에 사용 가능하다.

- 각 행(Row)들에 대해 개별적으로 작용하여 데이터 값들을 조작하고, 각각의 행에 대한 조작 결과를 리턴한다.

- 여러 인자(Argument)를 입력해도 단 하나의 결과만 리턴한다.

- 함수의 인자(Arguments)로 상수, 변수, 표현식이 사용 가능하고, 하나의 인수를 가지는 경우도 있지만 여러 개의 인수를 가질 수도 있다.

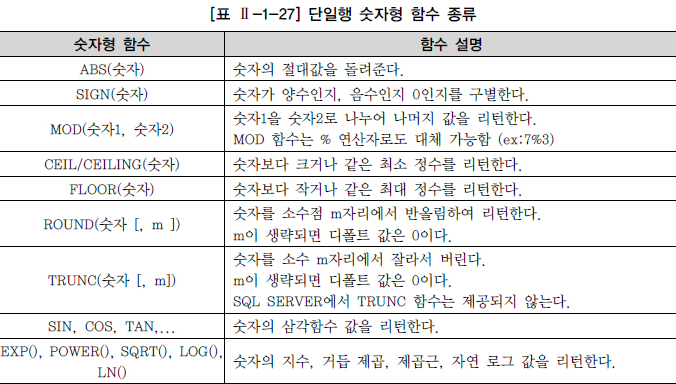
- 특별한 경우가 아니면 함수의 인자(Arguments)로 함수를 사용하는 함수의 중첩이 가능하다.

문자형 함수

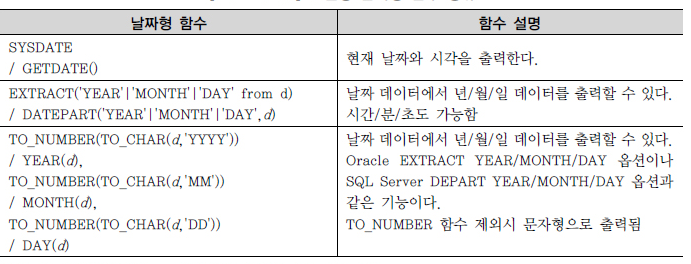


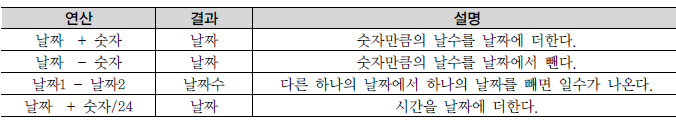
LRTIM(문자열,지정문자), TRIM(both, 지정문자 from 문자열), SUBSTR(문자열,m,n)

숫자형 함수



날짜형 함수





변환형 함수

Oracle TO\_NUMBER(문자열) 문자열을 숫자로

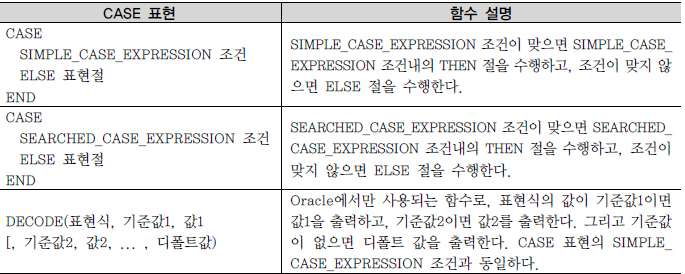
TO\_CHAR(숫자|날짜, FORMAT) 숫자나 날짜를 주어진 FORMAT 형태로 문자열 변환

TO\_DATE(문자열, FORMAT) 문자열을 주어진 FORMAT 형태로 날짜 타입 변환

SQL Server CAST (expression AS data\_type(length)) expression을 목표 데이터 유형으로

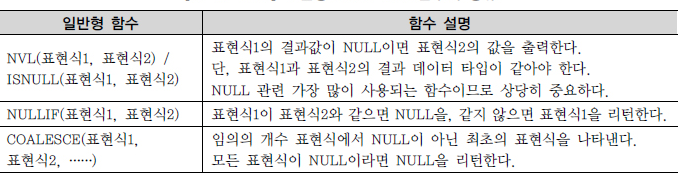
CONVERT (data\_type(length),expression,style) expression을 목표 데이터 유형으로

CASE 함수 => IF-THEN-ELSE 논리



NULL 관련 함수

NULL 사칙연산 숫자 = NULL



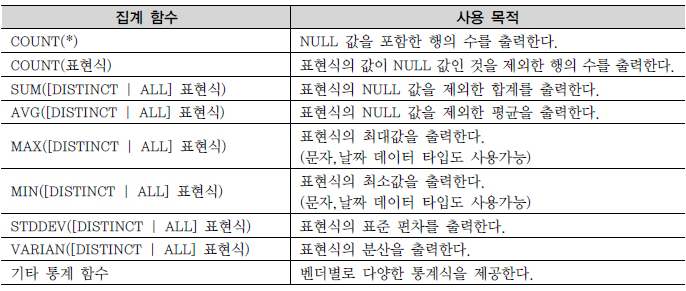
7. GROUP BY, HAVING절 ( 다중행 함수)

- 여러 행들의 그룹이 모여서 그룹당 단 하나의 결과를 돌려주는 함수이다.

- GROUP BY 절은 행들을 소그룹화 한다.

- SELECT 절, HAVING 절, ORDER BY 절에 사용할 수 있다. WHERE절은 안된다.

집계함수



일반적으로 집계 함수는 GROUP BY 절과 같이 사용되지만 아래와 같이 테이블 전체가 하나의 그룹이 되는 경우에는 GROUP BY 절 없이 단독으로도 사용 가능하다.

GROUP BY절, HAVING절

GROUP BY 절과 HAVING 절은 다음과 같은 특성을 가진다.

- GROUP BY 절을 통해 소그룹별 기준을 정한 후, SELECT 절에 집계 함수를 사용한다.

- 집계 함수의 통계 정보는 NULL 값을 가진 행을 제외하고 수행한다.

- GROUP BY 절에서는 SELECT 절과는 달리 ALIAS 명을 사용할 수 없다.

- 집계 함수는 WHERE 절에는 올 수 없다. (집계 함수를 사용할 수 있는 GROUP BY 절보다 WHERE 절이 먼저 수행된다)

- WHERE 절은 전체 데이터를 GROUP으로 나누기 전에 행들을 미리 제거시킨다.

- HAVING 절은 GROUP BY 절의 기준 항목이나 소그룹의 집계 함수를 이용한 조건을 표시할 수 있다.

- GROUP BY 절에 의한 소그룹별로 만들어진 집계 데이터 중, HAVING 절에서 제한 조건을 두어 조건을 만족하는 내용만 출력한다.

- HAVING 절은 일반적으로 GROUP BY 절 뒤에 위치한다.

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

8. ORDER BY절

ORDER BY 절은 SQL 문장으로 조회된 데이터들을 다양한 목적에 맞게 특정 칼럼을 기준으로 정렬하여 출력하는데 사용한다. ORDER BY 절에 칼럼(Column)명 대신에 SELECT 절에서 사용한 ALIAS 명이나 칼럼 순서를 나타내는 정수도 사용 가능하다. 그리고 별도로 정렬 방식을 지정하지 않으면 기본적으로 오름차순이 적용되며, SQL 문장의 제일 마지막에 위치한다.

GROUP BY 절과 ORDER BY가 같이 사용될 때 SELECT 문장은 6개의 절로 구성이 되고, SELECT 문장의 수행 단계는 아래와 같다.

5. SELECT 칼럼명 [ALIAS명] 1. FROM 테이블명 2. WHERE 조건식 3. GROUP BY 칼럼(Column)이나 표현식 4. HAVING 그룹조건식 6. ORDER BY 칼럼(Column)이나 표현식;

TOP N

Oracle에서 순위가 높은 N개의 로우를 추출하기 위해 ORDER BY 절과 WHERE 절의 ROWNUM 조건을 같이 사용

[예제] 사원 테이블에서 급여가 높은 3명만 내림차순으로 출력

[예제] SELECT ENAME, SAL FROM EMP WHERE ROWNUM < 4 ORDER BY SAL DESC;

* 잘못된 예시 : 정렬보다 where이 먼저

[예제] SELECT ENAME, SAL FROM (SELECT ENAME, SAL FROM EMP ORDER BY SAL DESC) WHERE ROWNUM < 4 ; => 올바른 예시

반면 SQL Server는 TOP 조건을 사용하게 되면 별도 처리 없이 관련 Order By 절의 데이터 정렬 후 원하는 일부 데이터만 쉽게 출력

[예제] SELECT TOP(2) ENAME, SAL FROM EMP ORDER BY SAL DESC;

9. JOIN

EQUI JOIN

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

같은 내용을 ANSI/ISO SQL 표준 방식으로 표현

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

Non EQUI JOIN

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

3개 이상

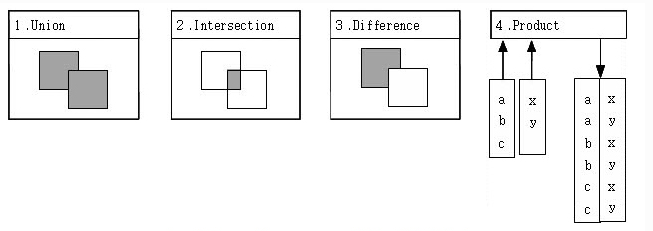
[예제] SELECT P.PLAYER\_NAME 선수명, P.POSITION 포지션, T.REGION\_NAME 연고지, T.TEAM\_NAME 팀명, S.STADIUM\_NAME 구장명 FROM PLAYER P, TEAM T, STADIUM S WHERE P.TEAM\_ID = T.TEAM\_ID AND T.STADIUM\_ID = S.STADIUM\_ID ORDER BY 선수명;

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;

2-2. SQL 활용

1. 표준 조인

일반 집합 연산자



1. UNION 연산은 UNION 기능으로, 중복을 없앤다. => 시스템 부하를 주는 정렬 작업

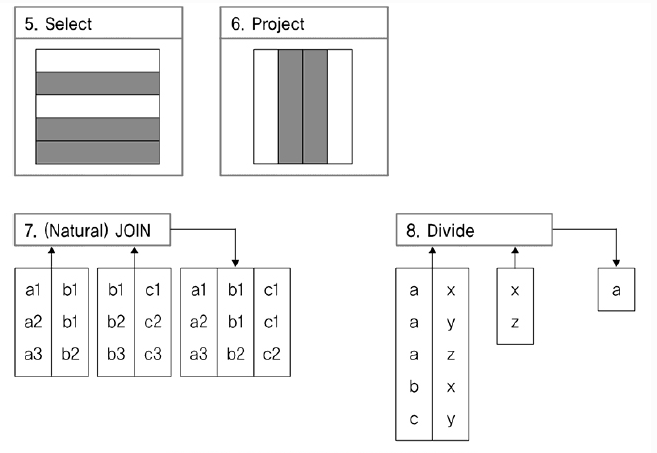
2. INTERSECTION 연산은 INTERSECT 기능으로,

3. DIFFERENCE 연산은 EXCEPT(Oracle은 MINUS) 기능으로,

4. PRODUCT 연산은 CROSS JOIN 기능으로 구현되었다.

UNION ALL : 공통집합을 중복 없이 보여준다. => 시스템 부하가 없다.

순수 관계 연산자



5. SELECT 연산은 WHERE 절로 구현되었다.

6. PROJECT 연산은 SELECT 절로 구현되었다.

7. (NATURAL) JOIN 연산은 다양한 JOIN 기능으로 구현되었다.

8. DIVIDE 연산은 현재 사용되지 않는다.

INNER JOIN JOIN 조건에서 동일한 값이 있는 행만 반환

USING 조건절이나 ON 조건절을 필수적으로 사용

NATURAL JOIN

NATURAL JOIN은 두 테이블 간의 동일한 이름을 갖는 모든 칼럼들에 대해 EQUI(=) JOIN을 수행한다. NATURAL JOIN이 명시되면, 추가로 USING 조건절, ON 조건절, WHERE 절에서 JOIN 조건을 정의할 수 없다. 그리고, SQL Server에서는 지원하지 않는 기능이다.

별도의 칼럼 순서를 지정하지 않으면 NATURAL JOIN의 기준이 되는 칼럼 들이 다른 칼럼보다 먼저 출력된다. (ex: DEPTNO가 첫 번째 칼럼이 된다.) 이때 NATURAL JOIN은 JOIN에 사용된 같은 이름의 칼럼을 하나로 처리한다.

DEPTNO EMPNO ENAME JOB MGR HIREDATE SAL COMM DNAME LOC

반면, INNER JOIN의 경우 첫 번째 테이블, 두 번째 테이블의 칼럼 순서대로 데이터가 출력된다. 이때 NATURAL JOIN은 JOIN에 사용된 같은 이름의 칼럼을 하나로 처리하지만, INNER JOIN은 별개의 칼럼으로 표시한다.

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO DEPTNO DNAME LOC

USING 조건절

NATURAL JOIN에서는 모든 일치되는 칼럼들에 대해 JOIN이 이루어지지만, FROM 절의 USING 조건절을 이용하면 같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI JOIN을 할 수가 있다. 다만, 이 기능은 SQL Server에서는 지원하지 않는다.

ON 조건절

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

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

CROSS JOIN => PRODUCT

CROSS JOIN은 E.F.CODD 박사가 언급한 일반 집합 연산자의 PRODUCT의 개념으로 테이블 간 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말한다.

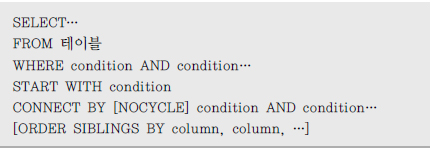
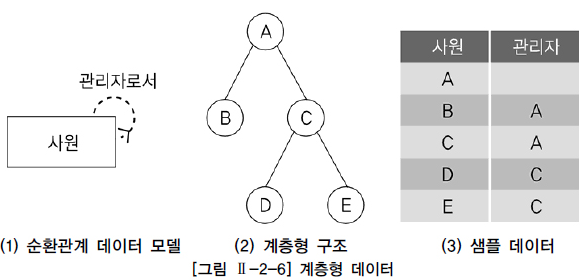
OUTER JOIN : FULL OUTER JOIN => UNION

INNER(내부) JOIN과 대비하여 OUTER(외부) JOIN이라고 불리며, JOIN 조건에서 동일한 값이 없는 행도 반환할 때 사용할 수 있다.

2.계층형 질의

테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해서 계층형 질의(Hierarchical Query)를 사용한다. 계층형 데이터란 동일 테이블에 계층적으로 상위와 하위 데이터가 포함된 데이터를 말한다.

Oracle



- START WITH절은 계층 구조 전개의 시작 위치를 지정하는 구문이다. 즉, 루트 데이터를 지정한다.(액세스)

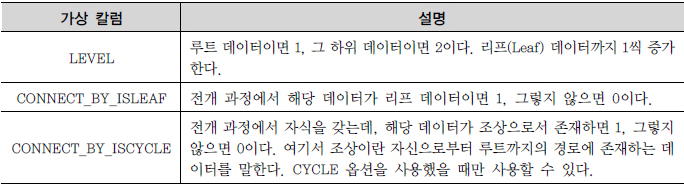
- CONNECT BY절은 다음에 전개될 자식 데이터를 지정하는 구문이다. 자식 데이터는 CONNECT BY절에 주어진 조건을 만족해야 한다.(조인)

- PRIOR : CONNECT BY절에 사용되며, 현재 읽은 칼럼을 지정한다. PRIOR 자식 = 부모 형태를 사용하면 계층구조에서 자식 데이터에서 부모 데이터(자식 → 부모) 방향으로 전개하는 순방향 전개를 한다. 그리고 PRIOR 부모 = 자식 형태를 사용하면 반대로 부모 데이터에서 자식 데이터(부모 → 자식) 방향으로 전개하는 역방향 전개를 한다.

- NOCYCLE : 데이터를 전개하면서 이미 나타났던 동일한 데이터가 전개 중에 다시 나타난다면 이것을 가리켜 사이클(Cycle)이 형성되었다라고 말한다. 사이클이 발생한 데이터는 런타임 오류가 발생한다. 그렇지만 NOCYCLE를 추가하면 사이클이 발생한 이후의 데이터는 전개하지 않는다.

- ORDER SIBLINGS BY : 형제 노드(동일 LEVEL) 사이에서 정렬을 수행한다.

- WHERE : 모든 전개를 수행한 후에 지정된 조건을 만족하는 데이터만 추출한다.(필터링)

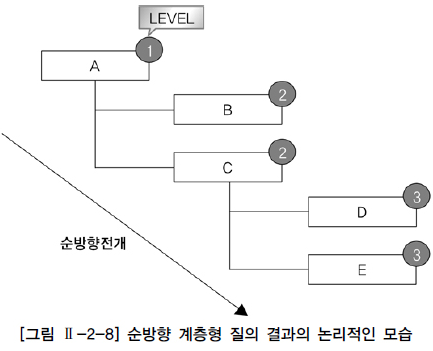


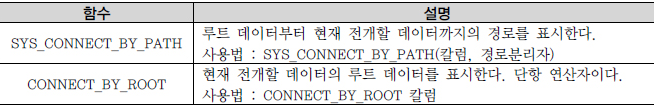
LPAD 함수는 지정한 길이 만큼 왼쪽부터 특정문자로 채워준다.

- 함수  :  **LPAD(**"값", "총 문자길이", "채움문자"**)**

[예제] SELECT LEVEL, LPAD(' ', 4 \* (LEVEL-1)) || 사원 사원, 관리자, CONNECT\_BY\_ISLEAF ISLEAF FROM 사원 START WITH 관리자 IS NULL CONNECT BY PRIOR 사원 = 관리자;

[실행 결과] LEVEL 사원 관리자 ISLEAF ----- -------- ----- ------ 1 A 0 2 B A 1 2 C A 0 3 D C 1 3 E C 1





[예제] SELECT CONNECT\_BY\_ROOT 사원 루트사원, SYS\_CONNECT\_BY\_PATH(사원, '/') 경로, 사원, 관리자 FROM 사원 START WITH 관리자 IS NULL CONNECT BY PRIOR 사원 = 관리자

[실행 결과] 루트사원 경로 사원 관리자 ------- ------- ---- ----- A /A A A /A/B B A A /A/C C A A /A/C/D D C A /A/C/E E C

SQL Server

WITH EMPLOYEES\_ANCHOR AS ( SELECT EMPLOYEEID, LASTNAME, FIRSTNAME, REPORTSTO, 0 AS LEVEL FROM EMPLOYEES WHERE REPORTSTO IS NULL /\* 재귀 호출의 시작점 \*/ UNION ALL SELECT R.EMPLOYEEID, R.LASTNAME, R.FIRSTNAME, R.REPORTSTO, A.LEVEL + 1 FROM EMPLOYEES\_ANCHOR A, EMPLOYEES R WHERE A.EMPLOYEEID = R.REPORTSTO ) SELECT LEVEL, EMPLOYEEID, LASTNAME, FIRSTNAME, REPORTSTO FROM EMPLOYEES\_ANCHOR GO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Level EmployeeID LastName FirstName ReportsTo ---- -------- ------- ----- -------- 0 2 Fuller Andrew NULL 1 1 Davolio Nancy 2 1 3 Leverling Janet 2 1 4 Peacock Margaret 2 1 5 Buchanan Steven 2 1 8 Callahan Laura 2 2 6 Suyama Michael 5 2 7 King Robert 5 2 9 Dodsworth Anne 5 (9개 행 적용됨)

WITH 절의 CTE 쿼리를 보면, UNION ALL 연산자로 쿼리 두 개를 결합했다. 둘 중 위에 있는 쿼리를 ‘앵커 멤버’(Anchor Member)라고 하고, 아래에 있는 쿼리를 ‘재귀 멤버’(Recursive Member)라고 한다. 아래는 재귀적 쿼리의 처리 과정이다.

1. CTE 식을 앵커 멤버와 재귀 멤버로 분할한다. 2. 앵커 멤버를 실행하여 첫 번째 호출 또는 기본 결과 집합(T0)을 만든다. 3. Ti는 입력으로 사용하고 Ti+1은 출력으로 사용하여 재귀 멤버를 실행한다. 4. 빈 집합이 반환될 때까지 3단계를 반복한다. 5. 결과 집합을 반환한다. 이것은 T0에서 Tn까지의 UNION ALL이다.

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.

셀프 조인

셀프 조인(Self Join)이란 동일 테이블 사이의 조인을 말한다. 따라서 FROM 절에 동일 테이블이 두 번 이상 나타난다. 동일 테이블 사이의 조인을 수행하면 테이블과 칼럼 이름이 모두 동일하기 때문에 식별을 위해 반드시 테이블 별칭(Alias)를 사용해야 한다. 그리고 칼럼에도 모두 테이블 별칭을 사용해서 어느 테이블의 칼럼인지 식별해줘야 한다.

SELECT ALIAS명1.칼럼명, ALIAS명2.칼럼명, ... FROM 테이블1 ALIAS명1, 테이블2 ALIAS명2 WHERE ALIAS명1.칼럼명2 = ALIAS명2.칼럼명1;

 “자신과 상위, 차상위 관리자를 같은 줄에 표시하라.” 이 문제를 해결하기 위해서는 FROM 절에 사원 테이블을 두 번 사용해야 한다.

3. 서브 쿼리

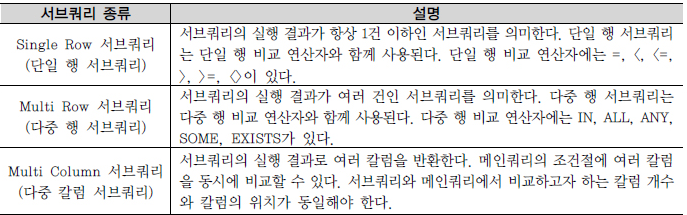
서브쿼리를 사용할 때 다음 사항에 주의해야 한다.

* 1. 서브쿼리를 괄호로 감싸서 사용한다.
  2. 서브쿼리는 단일 행(Single Row) 또는 복수 행(Multiple Row) 비교 연산자와 함께 사용 가능하다. 단일 행 비교 연산자는 서브쿼리의 결과가 반드시 1건 이하이어야 하고 복수 행 비교 연산자는 서브쿼리의 결과 건수와 상관 없다.
  3. 서브쿼리에서는 ORDER BY를 사용하지 못한다. ORDER BY절은 SELECT절에서 오직 한 개만 올 수 있기 때문에 ORDER BY절은 메인쿼리의 마지막 문장에 위치해야 한다.

서브쿼리가 SQL문에서 사용이 가능한 곳은 다음과 같다.

- SELECT 절 - FROM 절 - WHERE 절 - HAVING 절 - ORDER BY 절 - INSERT문의 VALUES 절 - UPDATE문의 SET 절



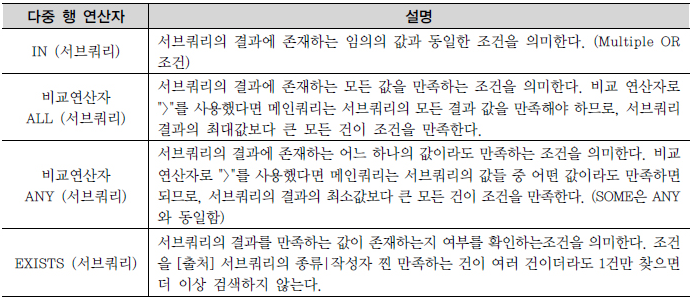


단일행 서브쿼리

[예제] SELECT PLAYER\_NAME 선수명, POSITION 포지션, BACK\_NO 백넘버 FROM PLAYER WHERE TEAM\_ID = (SELECT TEAM\_ID FROM PLAYER WHERE PLAYER\_NAME = '정남일') ORDER BY PLAYER\_NAME;

동명이인이 존재하지 않아 하나의 행만 반환

다중행 서브쿼리



[예제] 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;

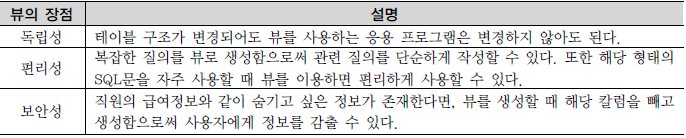
서브쿼리의 결과로 2개 이상의 행이 반환되어(동명이인이 존재) 단일 행 비교 연산자인 '='로는 처리가 불가능하기 때문

다중 칼럼 서브쿼리

[예제] SELECT TEAM\_ID 팀코드, PLAYER\_NAME 선수명, POSITION 포지션, BACK\_NO 백넘버, HEIGHT 키 FROM PLAYER WHERE (TEAM\_ID, HEIGHT) IN (SELECT TEAM\_ID, MIN(HEIGHT) FROM PLAYER GROUP BY TEAM\_ID) ORDER BY TEAM\_ID, PLAYER\_NAME;

뷰(View)

테이블은 실제로 데이터를 가지고 있는 반면, 뷰(View)는 실제 데이터를 가지고 있지 않다. 뷰는 단지 뷰 정의(View Definition)만을 가지고 있다. 질의에서 뷰가 사용되면 뷰 정의를 참조해서 DBMS 내부적으로 질의를 재작성(Rewrite)하여 질의를 수행한다. 뷰는 실제 데이터를 가지고 있지 않지만 테이블이 수행하는 역할을 수행하기 때문에 가상 테이블(Virtual Table)이라고도 한다.



CREATE VIEW V\_PLAYER\_TEAM AS SELECT P.PLAYER\_NAME, P.POSITION, P.BACK\_NO, P.TEAM\_ID, T.TEAM\_NAME FROM PLAYER P, TEAM T WHERE P.TEAM\_ID = T.TEAM\_ID;

DROP VIEW V\_PLAYER\_TEAM;

4. 그룹 함수

ANSI/ISO SQL 표준은 데이터 분석을 위해서 다음 세 가지 함수를 정의하고 있다.

- AGGREGATE FUNCTION - GROUP FUNCTION - WINDOW FUNCTION

ROLLUP 함수

GROUP BY 절 + ORDER BY 절 사용

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

[실행 결과] DNAME JOB Total Empl Total Sal ---------- -------- -------- ------- ACCOUNTING CLERK 1 1300 ACCOUNTING MANAGER 1 2450 ACCOUNTING PRESIDENT 1 5000 RESEARCH ANALYST 2 6000 RESEARCH CLERK 2 1900 RESEARCH MANAGER 1 2975 SALES CLERK 1 950 SALES MANAGER 1 2850 SALES SALESMAN 4 5600 9개의 행이 선택되었다.

ROLLUP 함수 + ORDER BY 절 사용

[예제] SELECT DNAME, JOB, COUNT(\*) "Total Empl", SUM(SAL) "Total Sal" FROM EMP, DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO GROUP BY ROLLUP (DNAME, JOB) ORDER BY DNAME, JOB ;

[실행 결과] DNAME JOB Total Empl Total Sal ------------ ------- -------- -------- ACCOUNTING CLERK 1 1300 ACCOUNTING MANAGER 1 2450 ACCOUNTING PRESIDENT 1 5000 ACCOUNTING 3 8750 RESEARCH ANALYST 2 6000 RESEARCH CLERK 2 1900 RESEARCH MANAGER 1 2975 RESEARCH 5 10875 SALES CLERK 1 950 SALES MANAGER 1 2850 SALES SALESMAN 4 5600 SALES 6 9400 14 29025 13개의 행이 선택되었다.

L1 - GROUP BY 수행시 생성되는 표준 집계 (9건) L2 - DNAME 별 모든 JOB의 SUBTOTAL (3건) L3 - GRAND TOTAL (마지막 행, 1건)

GROUPING 함수 사용

ROLLUP, CUBE, GROUPING SETS 등 새로운 그룹 함수를 지원하기 위해 GROUPING 함수 추가

[예제] SELECT DNAME, GROUPING(DNAME), JOB, GROUPING(JOB), COUNT(\*) "Total Empl", SUM(SAL) "Total Sal" FROM EMP, DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO GROUP BY ROLLUP (DNAME, JOB);

[실행 결과] DNAME ------ GROUPING(DNAME) -------------- JOB --- GROUPING(JOB) ----------- Total Empl -------- Total Sal ------ SALES 0 CLERK 0 1 950 SALES 0 MANAGER 0 1 2850 SALES 0 SALESMAN 0 4 5600 SALES 0 1 6 9400 RESEARCH 0 CLERK 0 2 1900 RESEARCH 0 ANALYST 0 2 6000 RESEARCH 0 MANAGER 0 1 2975 RESEARCH 0 1 5 10875 ACCOUNTING 0 CLERK 0 1 1300 ACCOUNTING 0 MANAGER 0 1 2450 ACCOUNTING 0 PRESIDENT 0 1 5000 ACCOUNTING 0 1 3 8750 1 1 14 29025 13개의 행이 선택되었다.

CUBE 함수

ROLLUP에서는 단지 가능한 Subtotal만을 생성하였지만, CUBE는 결합 가능한 모든 값에 대하여 다차원 집계를 생성한다. CUBE를 사용할 경우에는 내부적으로는 Grouping Columns의 순서를 바꾸어서 또 한 번의 Query를 추가 수행해야 한다. 뿐만 아니라 Grand Total은 양쪽의 Query 에서 모두 생성이 되므로 한 번의 Query에서는 제거되어야만 하므로 ROLLUP에 비해 시스템의 연산 대상이 많다. 이처럼 Grouping Columns이 가질 수 있는 모든 경우에 대하여 Subtotal을 생성해야 하는 경우에는 CUBE를 사용하는 것이 바람직하나, ROLLUP에 비해 시스템에 많은 부담을 주므로 사용에 주의해야 한다. CUBE 함수의 경우 표시된 인수들에 대한 계층별 집계를 구할 수 있으며, 이때 표시된 인수들 간에는 계층 구조인 ROLLUP과는 달리 평등한 관계이므로 인수의 순서가 바뀌는 경우 행간에 정렬 순서는 바뀔 수 있어도 데이터 결과는 같다. 그리고 CUBE도 결과에 대한 정렬이 필요한 경우는 ORDER BY 절에 명시적으로 정렬 칼럼이 표시가 되어야 한다.

[예제] 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) ;

[실행 결과] DNAME JOB Total Empl Total Sal ------------- --------- --------- -------- All Departments All Jobs 14 29025 All Departments CLERK 4 4150 All Departments ANALYST 2 6000 All Departments MANAGER 3 8275 All Departments SALESMAN 4 5600 All Departments PRESIDENT 1 5000 SALES All Jobs 6 9400 SALES CLERK 1 950 SALES MANAGER 1 2850 SALES SALESMAN 4 5600 RESEARCH All Jobs 5 10875 RESEARCH CLERK 2 1900 RESEARCH ANALYST 2 6000 RESEARCH MANAGER 1 2975 ACCOUNTING All Jobs 3 8750 ACCOUNTING CLERK 1 1300 ACCOUNTING MANAGER 1 2450 ACCOUNTING PRESIDENT 1 5000 18개의 행이 선택되었다.

(All Departments - CLERK, ANALYST, MANAGER, SALESMAN, PRESIDENT 별 집계가 5건 추가되었다.)

GROUPING SETS 함수

GROUPING SETS를 이용해 더욱 다양한 소계 집합을 만들 수 있는데, GROUP BY SQL 문장을 여러 번 반복하지 않아도 원하는 결과를 쉽게 얻을 수 있게 되었다. GROUPING SETS에 표시된 인수들에 대한 개별 집계를 구할 수 있으며, 이때 표시된 인수들 간에는 계층 구조인 ROLLUP과는 달리 평등한 관계이므로 인수의 순서가 바뀌어도 결과는 같다. 그리고 GROUPING SETS 함수도 결과에 대한 정렬이 필요한 경우는 ORDER BY 절에 명시적으로 정렬 칼럼이 표시가 되어야 한다.

[예제] SELECT DECODE(GROUPING(DNAME), 1, 'All Departments', DNAME) AS DNAME, DECODE(GROUPING(JOB), 1, 'All Jobs', JOB) AS JOB, COUNT(\*) "Total Empl", SUM(SAL) "Total Sal" FROM EMP, DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO GROUP BY GROUPING SETS (JOB, DNAME);

[실행 결과] DNAME JOB Total Empl Total Sal -------------- --------- ---------- --------- All Departments CLERK 4 4150 All Departments SALESMAN 4 5600 All Departments PRESIDENT 1 5000 All Departments MANAGER 3 8275 All Departments ANALYST 2 6000 ACCOUNTING All Jobs 3 8750 RESEARCH All Jobs 5 10875 SALES All Jobs 6 9400 8개의 행이 선택되었다.

Department 별 집계 5개 + Job 별 집계 3개 = 8개

5. 윈도우 함수

행과 행간의 관계를 쉽게 정의하기 위해 만든 함수가 바로 WINDOW FUNCTION

WINDOW 함수는 다른 함수와는 달리 중첩(NEST)해서 사용하지는 못하지만, 서브쿼리에서는 사용할 수 있다.

(1) 그룹 내 순위 함수

RANK 함수

RANK 함수는 ORDER BY를 포함한 QUERY 문에서 특정 항목(칼럼)에 대한 순위를 구하는 함수이다. 이때 특정 범위(PARTITION) 내에서 순위를 구할 수도 있고 전체 데이터에 대한 순위를 구할 수도 있다. 또한 동일한 값에 대해서는 동일한 순위를 부여하게 된다.

[예제] 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;

업무 구분이 없는 ALL\_RANK 칼럼에서 FORD와 SCOTT, WARD와 MARTIN은 동일한 SALARY이므로 같은 순위를 부여한다. 그리고 업무를 PARTITION으로 구분한 JOB\_RANK의 경우 같은 업무 내 범위에서만 순위를 부여한다. 하나의 SQL 문장에 ORDER BY SAL DESC 조건과 PARTITION BY JOB 조건이 충돌이 났기 때문에 JOB 별로는 정렬이 되지 않고, ORDER BY SAL DESC 조건으로 정렬이 되었다.

DENSE\_RANK 함수

RANK 함수와 흡사하나, 동일한 순위를 하나의 건수로 취급하는 것이 틀린 점이다.

RANK 공동 2등 -> 다음 4등, DENSE\_RANK 공동 2등 -> 다음 3등

ROW\_NUMBER 함수

RANK나 DENSE\_RANK 함수가 동일한 값에 대해서는 동일한 순위를 부여하는데 반해, 동일한 값이라도 고유한 순위를 부여한다.

공동 2등 x -> 2, 3등으로 나눈다.(Oracle의 경우 rowid가 적은 행이 우선)

(2) 일반 집계 함수

SUM, MAX, MIN. AVG, COUNT ex) MIN(SAL) OVER(PARTITION BY MGR ORDER BY HIREDATE)

(3) 그룹 내 행 순서 함수

FIRST\_VALUE 함수

파티션별 윈도우에서 가장 먼저 나온 값을 구한다. SQL Server에서는 지원하지 않는 함수이다.

LAST\_VALUE 함수

파티션별 윈도우에서 가장 나중에 나온 값을 구한다. SQL Server에서는 지원하지 않는 함수이다.

LAG 함수 LAG(SAL, 2, 0)

파티션별 윈도우에서 이전 몇 번째 행의 값을 가져올 수 있다. SQL Server에서는 지원하지 않는 함수이다.

두 번째 인자는 몇 번째 앞의 행을 가져올지 결정하는 것이고 (DEFAULT 1), 세 번째 인자는 예를 들어 파티션의 첫 번째 행의 경우 가져올 데이터가 없어 NULL 값이 들어오는데 이 경우 다른 값으로 바꾸어 줄 수 있다. 결과적으로 NVL이나 ISNULL 기능과 같다.

LEAD 함수

파티션별 윈도우에서 이후 몇 번째 행의 값을 가져올 수 있다. 참고로 SQL Server에서는 지원하지 않는 함수이다. 인자는 LAG와 비슷한 맥락

(4) 그룹 내 비율 함수

RATIO\_TO\_REPORT 함수

파티션 내 전체 SUM(칼럼)값에 대한 행별 칼럼 값의 백분율을 소수점으로 구할 수 있다. 결과 값은 > 0 & <= 1 의 범위를 가진다. 그리고 개별 RATIO의 합을 구하면 1이 된다. SQL Server에서는 지원하지 않는 함수이다.

PERCENT\_RANK 함수

파티션별 윈도우에서 제일 먼저 나오는 것을 0으로, 제일 늦게 나오는 것을 1로 하여, 값이 아닌 행의 순서별 백분율을 구한다. 결과 값은 >= 0 & <= 1 의 범위를 가진다. 참고로 SQL Server에서는 지원하지 않는 함수이다.

CUME\_DIST 함수

파티션별 윈도우의 전체건수에서 현재 행보다 작거나 같은 건수에 대한 누적백분율을 구한다. 결과 값은 > 0 & <= 1 의 범위를 가진다. 참고로 SQL Server에서는 지원하지 않는 함수이다.

다른 WINDOW 함수의 경우 동일 순서면 앞 행의 함수 결과 값을 따르는데, CUME\_DIST의 경우는 동일 순서면 뒤 행의 함수 결과값을 기준으로 한다.

NTILE 함수

파티션별 전체 건수를 ARGUMENT 값으로 N 등분한 결과를 구할 수 있다.

위 예제에서 NTILE(4)의 의미는 14명의 팀원을 4개 조로 나눈다는 의미이다. 전체 14명을 4개의 집합으로 나누면 몫이 3명, 나머지가 2명이 된다. 나머지 두 명은 앞의 조부터 할당한다. 즉, 4명 + 4명 + 3명 + 3명으로 조를 나누게 된다.

(5) 선형 분석을 포함한 통계 분석 관련 함수=> 넘어간드아

6. DCL(데이터 통제어)

유저 생성 권한 =>  GRANT CREATE USER TO SCOTT;

유저가 로그인을 하려면 CREATE SESSION 권한을 부여 => GRANT CREATE SESSION TO PJS;

테이블 생성 권한 => GRANT CREATE TABLE TO PJS;

SCOTT 유저에게 MENU 테이블을 SELECT 할 수 있는 권한을 부여

* GRANT SELECT ON MENU TO SCOTT;

ROLE

ROLE을 생성하고, ROLE에 각종 권한들을 부여한 후 ROLE을 다른 ROLE이나 유저에게 부여할 수 있다. 또한 ROLE에 포함되어 있는 권한들이 필요한 유저에게는 해당 ROLE만을 부여함으로써 빠르고 정확하게 필요한 권한을 부여할 수 있게 된다.

ROLE에는 시스템 권한과 오브젝트 권한을 모두 부여할 수 있으며, ROLE은 유저에게 직접 부여될 수도 있고, 다른 ROLE에 포함하여 유저에게 부여될 수도 있다.

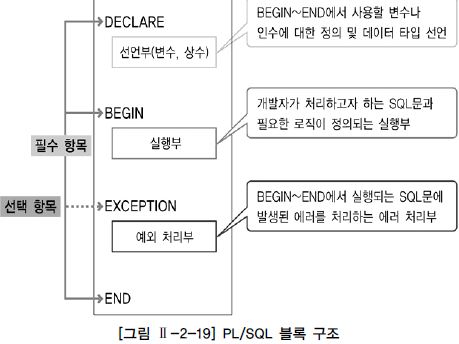
[예제] JISUNG 유저에게 CREATE SESSION과 CREATE TABLE 권한을 가진 ROLE을 생성한 후 ROLE을 이용하여 다시 권한을 할당한다. 권한을 취소할 때는 REVOKE를 사용한다.

Oracle에서는 기본적으로 몇 가지 ROLE을 제공하고 있다. 그 중 가장 많이 사용하는 ROLE은 CONNECT와 RESOURCE이다.

7. 절차형 SQL

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.

PL/SQL(Oracle)



PL/SQL의 특징

- PL/SQL은 Block 구조로 되어있어 각 기능별로 모듈화가 가능하다.

- 변수, 상수 등을 선언하여 SQL 문장 간 값을 교환한다.

- IF, LOOP 등의 절차형 언어를 사용하여 절차적인 프로그램이 가능하도록 한다.

- DBMS 정의 에러나 사용자 정의 에러를 정의하여 사용할 수 있다.

- PL/SQL은 Oracle에 내장되어 있으므로 Oracle과 PL/SQL을 지원하는 어떤 서버로도 프로그램을 옮길 수 있다.

- PL/SQL은 응용 프로그램의 성능을 향상시킨다.

- PL/SQL은 여러 SQL 문장을 Block으로 묶고 한 번에 Block 전부를 서버로 보내기 때문에 통신량을 줄일 수 있다.

CREATE [OR REPLACE] Procedure [Procedure\_name] ( argument1 [mode] data\_type1, argument2 [mode] date\_type2, ... ... ) IS [AS] ... ... BEGIN ... ... EXCEPTION ... ... END; /

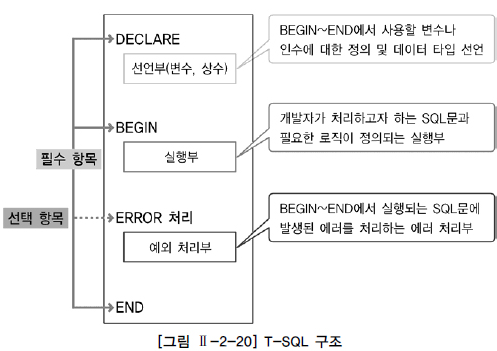
DROP Procedure [Procedure\_name];

마지막에 있는 슬래쉬(“/”)는 데이터베이스에게 프로시저를 컴파일하라는 명령어

Argument => 매개변수 mode => OUT(OS->프로시저), IN(반대), INOUT(쌍방향)

프로시저의 변경이 필요할 경우 Oracle은 [CREATE OR REPLACE]

T-SQL (SQL Server)



- 변수 선언 기능 @@이라는 전역변수(시스템 함수)와 @이라는 지역변수가 있다.

- 지역변수는 사용자가 자신의 연결 시간 동안만 사용하기 위해 만들어지는 변수이며 전역변수는 이미 SQL서버에 내장된 값이다.

- 데이터 유형(Data Type)을 제공한다. 즉 int, float, varchar 등의 자료형을 의미한다.

- 연산자(Operator) 산술연산자( +, -, \*, /)와 비교연산자(=, <, >, <>) 논리연산자(and, or, not) 사용이 가능하다.

- 흐름 제어 기능 IF-ELSE와 WHILE, CASE-THEN 사용이 가능하다.

- 주석 기능한줄 주석 : -- 뒤의 내용은 주석범위 주석 : /\* 내용 \*/ 형태를 사용하며, 여러 줄도 가능함

CREATE Procedure [schema\_name.]Procedure\_name @parameter1 data\_type1 [mode], @parameter2 date\_type2 [mode], ... ... WITH AS ... ... BEGIN ... ... ERROR 처리 ... ... END;

DROP Procedure [schema\_name.]Procedure\_name;

프로시저의 변경이 필요할 경우 SQL Server는 CREATE 구문을 ALTER 구문으로 변경

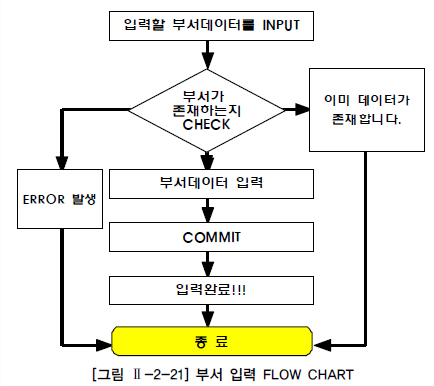
Argument => 매개변수 mode =>

1. VARYING 결과 집합이 출력 매개 변수로 사용되도록 지정합니다. CURSOR 매개변수에만 적용된다.
2. DEFAULT 지정된 매개변수가 프로시저를 호출할 당시 지정되지 않을 경우 지정된 기본값으로 처리한다. 즉, 기본 값이 지정되어 있으면 해당 매개 변수를 지정하지 않아도 프로시저가 지정된 기본 값으로 정상적으로 수행이 된다.
3. OUT, OUTPUT 프로시저에서 처리된 결과 값을 EXECUTE 문 호출 시 반환한다.
4. READONLY 자주 사용되지는 않는다. 프로시저 본문 내에서 매개 변수를 업데이트하거나 수정할 수 없음을 나타낸다. 매개 변수 유형이 사용자 정의 테이블 형식인 경우 READONLY를 지정해야 한다.

WITH =>

1. RECOMPILE 데이터베이스 엔진에서 현재 프로시저의 계획을 캐시하지 않고 프로시저가 런타임에 컴파일 된다. 데이터베이스 엔진에서 저장 프로시저 안에 있는 개별 쿼리에 대한 계획을 삭제하려 할 때 RECOMPILE 쿼리 힌트를 사용한다.
2. ENCRYPTIONCREATE PROCEDURE 문의 원본 텍스트가 알아보기 어려운 형식으로 변환된다. 변조된 출력은 SQL Server의 카탈로그 뷰 어디에서도 직접 표시되지 않는다. 원본을 볼 수 있는 방법이 없기 때문에 반드시 원본은 백업을 해두어야 한다.
3. EXECUTE AS 해당 저장 프로시저를 실행할 보안 컨텍스트를 지정한다.

[예제] SCOTT 유저가 소유하고 있는 DEPT 테이블에 새로운 부서를 등록하는 Procedure를 작성



[예제] Oracle CREATE OR REPLACE Procedure p\_DEPT\_insert -------------① ( v\_DEPTNO in number, v\_dname in varchar2, v\_loc in varchar2, v\_result out varchar2) IS cnt number := 0; BEGIN SELECT COUNT(\*) INTO CNT -------------② FROM DEPT WHERE DEPTNO = v\_DEPTNO AND ROWNUM = 1; if cnt > 0 then -------------③ v\_result := '이미 등록된 부서번호이다'; else INSERT INTO DEPT (DEPTNO, DNAME, LOC) -------------④ VALUES (v\_DEPTNO, v\_dname, v\_loc); COMMIT; -------------⑤ v\_result := '입력 완료!!'; end if; EXCEPTION -------------⑥ WHEN OTHERS THEN ROLLBACK; v\_result := 'ERROR 발생'; END; /

Trigger

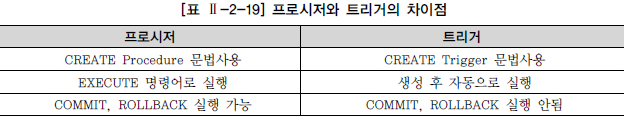
Trigger란 특정한 테이블에 INSERT, UPDATE, DELETE와 같은 DML문이 수행되었을 때, 데이터베이스에서 자동으로 동작하도록 작성된 프로그램이다.

Trigger는 테이블과 뷰, 데이터베이스 작업을 대상으로 정의할 수 있으며, 전체 트랜잭션 작업에 대해 발생되는 Trigger와 각 행에 대해서 발생되는 Trigger가 있다.

[예제] Oracle CREATE OR REPLACE Trigger SUMMARY\_SALES ---------------- ① AFTER INSERT ON ORDER\_LIST FOR EACH ROW DECLARE ---------------- ② o\_date ORDER\_LIST.order\_date%TYPE; o\_prod ORDER\_LIST.product%TYPE; BEGIN o\_date := :NEW.order\_date; o\_prod := :NEW.product; UPDATE SALES\_PER\_DATE ---------------- ③ SET qty = qty + :NEW.qty, amount = amount + :NEW.amount WHERE sale\_date = o\_date AND product = o\_prod; if SQL%NOTFOUND then ---------------- ④ INSERT INTO SALES\_PER\_DATE VALUES(o\_date, o\_prod, :NEW.qty, :NEW.amount); end if; END; /

1. Trigger를 선언한다.CREATE OR REPLACE Trigger SUMMARY\_SALES : Trigger 선언문AFTER INSERT : 레코드가 입력이 된 후 Trigger 발생 ON ORDER\_LIST : ORDER\_LIST 테이블에 Trigger 설정FOR EACH ROW : 각 ROW마다 Trigger 적용
2. o\_date(주문일자), o\_prod(주문상품) 값을 저장할 변수를 선언하고, 신규로 입력된 데이터를 저장한다.
3. 먼저 입력된 주문 내역의 주문 일자와 주문 상품을 기준으로 SALES\_PER\_DATE 테이블에 업데이트한다.
4. 처리 결과가 SQL%NOTFOUND이면 해당 주문 일자의 주문 상품 실적이 존재하지 않으며, SALES\_ PER\_DATE 테이블에 새로운 집계 데이터를 입력한다.

프로시저와 트리거의 차이점



2-3. SQL 최적화와 기본 원리

1. 옵티마이저와 실행 계획

옵티마이저(Optimizer)는 사용자가 질의한 SQL문에 대해 최적의 실행 방법을 결정하는 역할을 수행한다. 이러한 최적의 실행 방법을 실행계획(Execution Plan)이라고 한다.

규칙기반 옵티마이저

규칙기반 옵티마이저는 규칙(우선 순위)을 가지고 실행계획을 생성한다. SQL문을 실행하기 위해서 이용 가능한 인덱스 유무와 (유일, 비유일, 단일, 복합 인덱스)종류, SQL문에서 사용하는 연산자(=, <, <>, LIKE, BETWEEN 등)의 종류 그리고 SQL문에서 참조하는 객체(힙 테이블, 클러스터 테이블 등)의 종류 등 이러한 정보에 따라 우선 순위(규칙)가 정해져 있고, 이 우선 순위를 기반으로 실행계획을 생성한다.

비용기반 옵티마이저

비용이란 SQL문을 처리하기 위해 예상되는 소요시간 또는 자원 사용량을 의미한다. 비용기반 옵티마이저는 비용을 예측하기 위해서 규칙기반 옵티마이저가 사용하지 않는 테이블, 인덱스, 칼럼 등의 다양한 객체 통계정보와 시스템 통계정보 등을 이용한다. 통계정보가 없는 경우 비용기반 옵티마이저는 정확한 비용 예측이 불가능해져서 비효율적인 실행계획을 생성할 수 있다. 그렇기 때문에 정확한 통계정보를 유지하는 것은 비용기반 최적화에서 중요한 요소이다.

실행 계획

Oracle의 실행계획 형태는 [그림 Ⅱ-3-4]와 같다. 실행계획을 구성하는 요소에는 조인 순서(Join Order), 조인 기법(Join Method), 액세스 기법(Access Method), 최적화 정보(Optimization Information), 연산(Operation) 등이 있다

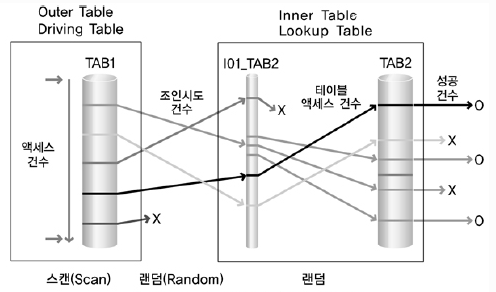
조인 순서는 조인작업을 수행할 때 참조하는 테이블의 순서, 조인 기법은 두 개의 테이블을 조인할 때 사용할 수 있는 방법으로서 여기에는 NL Join, Hash Join, Sort Merge Join 등이 있다.

액세스 기법은 하나의 테이블을 액세스할 때 사용할 수 있는 방법이다. 여기에는 인덱스를 이용하여 테이블을 액세스하는 인덱스 스캔(Index Scan)과 테이블 전체를 모두 읽으면서 조건을 만족하는 행을 찾는 전체 테이블 스캔(Full Table Scan) 등이 있다.

최적화 정보에는 Cost, Card, Bytes가 있다. Cost는 상대적인 비용 정보이고 Card는 Cardinality의 약자로서 주어진 조건을 만족한 결과 집합 혹은 조인 조건을 만족한 결과 집합의 건수를 의미한다. Bytes는 결과 집합이 차지하는 메모리 양을 바이트로 표시한 것이다.

연산(Operation)은 여러 가지 조작을 통해서 원하는 결과를 얻어내는 일련의 작업이다. 연산에는 조인 기법(NL Join, Hash Join, Sort Merge Join 등), 액세스 기법(인덱스 스캔, 전체 테이블 스캔 등), 필터, 정렬, 집계, 뷰 등 다양한 종류가 존재한다.

SQL 처리 흐름도 => 실행계획을 시각화한 것

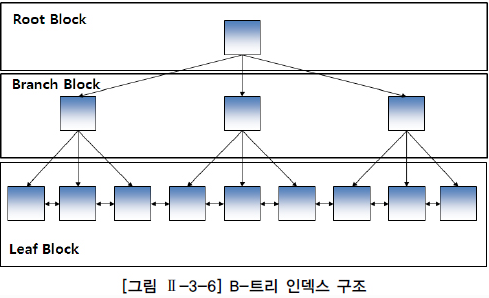


TAB1을 Outer Table 또는 Driving Table이라고 하고, TAB2를 Inner Table 또는 Lookup Table이라고 한다. 테이블의 액세스 방법은 TAB1은 테이블 전체 스캔을 의미하고 TAB2는 I01\_TAB2 이라는 인덱스를 통한 인덱스 스캔을 했음을 표시한 것이다. 조인 방법은 NL Join을 수행했음을 표시한 것이다. TAB1에 대한 액세스는 스캔(Scan) 방식이고 조인시도 및 I01\_TAB2 인덱스를 통한 TAB2 액세스는 랜덤(Random) 방식이다. 대량의 데이터를 랜덤 방식으로 액세스하게 되면 많은 I/O가 발생하여 성능상 좋지 않다.

2. 인덱스 기본

인덱스는 원하는 데이터를 쉽게 찾을 수 있도록 돕는 책의 찾아보기와 유사한 개념이다. 인덱스는 테이블을 기반으로 선택적으로 생성할 수 있는 구조이다. 테이블에 인덱스를 생성하지 않아도 되고 여러 개를 생성해도 된다. 인덱스의 기본적인 목적은 검색 성능의 최적화이다.

트리 기반 인덱스



SQL Server의 클러스터형 인덱스

첫째, 인덱스의 리프 페이지가 곧 데이터 페이지다. 따라서 테이블 탐색에 필요한 레코드 식별자가 리프 페이지에 없다(인덱스 키 칼럼과 나머지 칼럼을 리프 페이지에 같이 저장하기 때문에 테이블을 랜덤 액세스할 필요가 없다). 클러스터형 인덱스의 리프 페이지를 탐색하면 해당 테이블의 모든 칼럼 값을 곧바로 얻을 수 있다.

둘째, 리프 페이지의 모든 로우(=데이터)는 인덱스 키 칼럼 순으로 물리적으로 정렬되어 저장된다. 테이블 로우는 물리적으로 한 가지 순서로만 정렬될 수 있다. 그러므로 클러스터형 인덱스는 테이블당 한 개만 생성할 수 있다.

전체 테이블 스캔

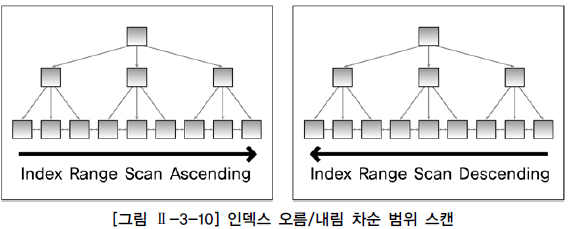
옵티마이저가 연산으로서 전체 테이블 스캔 방식을 선택하는 이유는 일반적으로 다음과 같다.

1) SQL문에 조건이 존재하지 않는 경우

2) SQL문의 주어진 조건에 사용 가능한 인덱스가 존재하는 않는 경우

3) 옵티마이저의 취사 선택

인덱스 스캔



3. 조인 수행 원리

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.

FOR 선행 테이블 읽음 → 외부 테이블(Outer Table) FOR 후행 테이블 읽음 → 내부 테이블(Inner Table) (선행 테이블과 후행 테이블 조인)

추출버퍼는 SQL문의 실행결과를 보관하는 버퍼로서 일정 크기를 설정하여 추출버퍼에 결과가 모두 차거나 더 이상 결과가 없어서 추출버퍼를 채울 것이 없으면 결과를 사용자에게 반환한다. 추출버퍼는 운반단위, Array Size, Prefetch Size라고도 한다.

NL Join은 주로 랜덤 액세스 방식으로 데이터를 읽는다.

Sort Merge Join

Sort Merge Join은 조인 칼럼을 기준으로 데이터를 정렬하여 조인을 수행한다. 주로 스캔 방식으로 데이터를 읽는다.

Sort Merge Join은 정렬할 데이터가 많아 메모리에서 모든 정렬 작업을 수행하기 어려운 경우에는 임시 영역(디스크)을 사용하기 때문에 성능이 떨어질 수 있다.

그러나 Sort Merge Join은 Hash Join과는 달리 동등 조인 뿐만 아니라 비동등 조인에 대해서도 조인 작업이 가능하다는 장점이 있다.

Hash Join

Hash Join은 해슁 기법을 이용하여 조인을 수행한다. 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.

Hash Join은 조인 칼럼의 인덱스를 사용하지 않기 때문에 조인 칼럼의 인덱스가 존재하지 않을 경우에도 사용할 수 있는 조인 기법이다.

동등 조인에서만 사용할 수 있다.

Hash Join을 할 때는 결과 행의 수가 적은 테이블을 선행 테이블로 사용하는 것이 좋다.

선행 테이블을 Build Input이라고도 하며, 후행 테이블은 만들어진 해쉬 테이블에 대해 해쉬 값의 존재여부를 검사한다고 해서 Prove Input이라고도 한다.