

1장. 데이터 모델링의 이해

데이터 모델링의 중요성 및 유의점

- **중복** : 같은 시간 같은 데이터 제공
- **비유연성** : 사소한 업무변화에 데이터 모델이 수시로 변경되면 안됨. 데이터 정의를 사용 프로세스와 분리한다.
- **비일관성** : 데이터 간 상호 연관 관계에 대해 명확히 정의해야 한다.

데이터 모델링

개념적, 논리적, 물리적 데이터 모델링

데이터 독립성 요소

외부 스키마 : 개개 사용자가 보는 개인적 DB 스키마

개념 스키마 : 모든 사용자 관점을 통합한 **전체 DB**

내부 스키마 : **물리적** 장치에서 데이터가 실제적 **저장**

데이터 독립성

논리적 독립성 : 개념스키마 변경, 외부스키마에 영향X

물리적 독립성 : 내부스키마 변경, 외부/개념스키마에 영향X

Mapping(사상) : 상호 독립적인 개념을 연결시켜주는 다리

논리적 사상 : 외부 스키마 - 개념 스키마

물리적 사상 : 개념 스키마 - 내부 스키마

데이터 모델링의 3요소

어떤 것(Things)

성격(Attributes)

관계(Relationships)

모델링의 특징

(현실세계) -> 추상화, 단순화, 정확화 -> (모델)

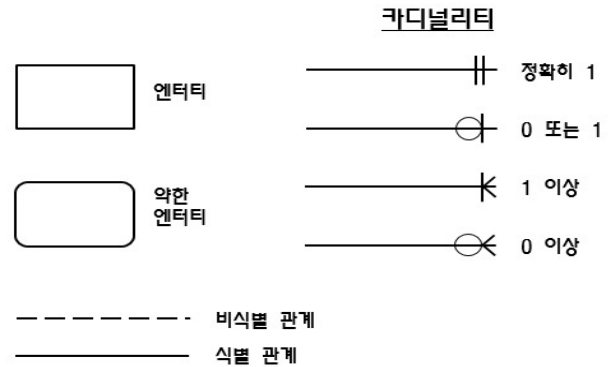
데이터 모델 표기법

1976년 피터첸이 Entity Relationship Model 개발

- **IE, Baker 기법**이 많이 쓰임
- 엔터티, 관계, 속성으로 이뤄짐

ERD 작업순서

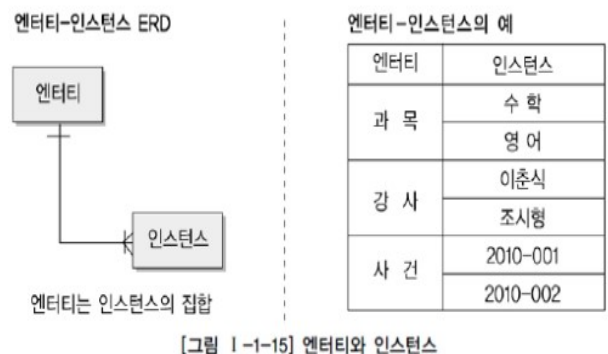
1. **엔터티 그림**
2. **엔터티 배치**
3. **엔터티 관계설정**
4. **관계명 기술**
5. **관계의 참여도 기술**
6. **관계필수여부**



좋은 데이터 모델의 요소

1. **완전성** : 업무에 필요한 모든 데이터가 모델에 정의
2. **중복배제** : 하나의 DB내에 동일한 사실은 한번만.
3. **업무규칙** : 많은 규칙을 사용자가 공유하도록 제공
4. **데이터 재사용** : 데이터가 독립적으로 설계돼야 함
5. **의사소통** : 업무규칙은 엔터티, 서브타입, 속성, 관계 등의 형태로 최대한 자세히 표현
6. **통합성** : 동일한 데이터는 한번만 정의, 참조 활용

엔터티 : 업무에 필요하고 유용한 정보를 저장하고 관리하기 위한 집합적인 것, 보이지 않는 개념 포함



엔터티의 특징

1. 반드시 해당 **업무에서 필요하고 관리**하고자 함

2. 유일한 식별자에 의해 식별 가능
3. 두 개 이상의 인스턴스의 집합
4. 업무 프로세스에 의해 이용되어야 함
5. 반드시 속성이 있어야 함
6. 다른 엔터티와 최소 1개 이상의 관계가 있어야 함
-> 통계성/코드성 엔터티는 관계 생략 가능

엔터티의 분류

유무형에 따른 분류 : 유형, 개념, 사건 엔터티

- **유형** : 물리적 형태 ex) 사원, 물품, 강사
- **개념** : 개념적 정보 ex) 조직, 보험상품
- **사건** : 업무수행시 발생 ex) 주문, 청구, 미납

발생시점에 따른 분류 : 기본/키, 중심, 행위 엔터티

- **기본** : 그 업무에 원래 존재하는 정보, 타 엔터티의 부모 역할, 자신의 고유한 주식별자 가짐 ex) 사원, 부서
- **중심** : 기본 엔터티로부터 발생, 다른 엔터티와의 관계로 많은 행위 엔터티 생성 ex) 계약, 사고, 주문
- **행위** : 2개 이상의 부모 엔터티로부터 발생, 자주 바뀌거나 양이 증가 ex) 주문목록, 사원변경이력

엔터티의 명명

현업업무에서 사용하는 용어 사용, 약어 사용금지, 단수명사 사용, 고유한 이름 사용, 생성의미대로 부여

속성 : 업무에서 필요로 하는 인스턴스로 관리하고자 하는 의미상 분리되지 않는 최소의 데이터 단위

- 한 개의 엔터티는 2개 이상의 인스턴스 집합
- 한 개의 엔터티는 2개 이상의 속성을 가짐
- 한 개의 속성은 1개의 속성값을 가짐

구성 방식의 분류 : PK, FK, 일반 속성

속성의 분류 : 기본, 설계, 파생 속성

- **기본** : 업무로부터 추출한 모든 일반적인 속성
- **설계** : 업무를 규칙화하기 위해 새로 만들거나 변형, 정의하는 속성 ex) 일련번호
- **파생** : 다른 속성에 영향을 받아 발생하는 속성, 빠른 성능을 낼 수 있도록 원래 속성의 값을 계산, 적을수록 좋음 ex) 합

도메인 : 속성에 대한 데이터 타입, 크기, 제약사항 지정

속성의 명명

1. 해당업무에서 사용하는 이름 부여
2. 서술식 속성명은 사용 금지
3. 약어 사용 금지
4. 구체적으로 명명하여 데이터 모델에서 유일성 확보

관계 : 엔터티의 인스턴스 사이의 논리적인 연관성으로서 존재의 형태로서나 행위로서 서로에게 연관성이 부여된 상태, 관계 페어링의 집합
ex) 강사 - 가르친다(관계) - 수강생

페어링 : 엔터티 안에 인스턴스가 개별적으로 관계를 가지는 것

UML(통합모델링언어)에서의 관계

연관관계(실선) : 항상 이용하는 관계 ex) 소속된다.

의존관계(점선) : 상대 행위에 의해 발생하는 관계 ex) 주문한다.

관계의 표기법

관계명 : 관계의 이름

관계차수 : 1:1, 1:M, M:N

관계선택성(관계선택사양) : 필수관계, 선택관계

관계 체크사항

1. 2개의 엔터티 사이에 관심있는 연관 규칙O?
2. 2개의 엔터티 사이에 정보의 조합 발생O?
3. 업무기술서, 장표에 관계연결에 대한 규칙 서술O?
4. 업무기술서, 장표에 관계연결을 가능케 하는 동사O?

식별자 : 엔터티 내에서 인스턴스를 구분하는 구분자
식별자는 논리적, Key는 물리적 데이터 모델링 단계에 사용

식별자의 특징 : 유일성, 최소성, 불변성, 존재성

- **유일성** : 주식별자에 의해 모든 인스턴스들이 유일하게 구분

- **최소성** : 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함
- **불변성** : 지정된 주식별자의 값은 자주 변하지 않아야 함. 변하면 이전 기록 말소됨
- **존재성** : 주식별자가 지정되면 반드시 값이 들어와야 함

식별자 분류

대표성여부 : 주식별자, 보조식별자

- **주** : 엔터티 내에서 각 어커런스를 구분할 수 있는 구분자, 타 엔터티와 참조관계를 연결O
- **보조** : 구분자이나 **대표성X**, **참조관계 연결X**

스스로생성여부 : 내부식별자, 외부식별자

내부 : 스스로 생성되는 식별자

외부 : 타 엔터티로부터 받아오는 식별자

속성의 수 : 단일식별자, 복합식별자

단일 : 하나의 속성으로 구성

복합 : 2개 이상의 속성으로 구성

대체 여부 : 본질식별자, 인조식별자

본질 : 업무에 의해 만들어지는 식별자

인조 : 인위적으로 만든 식별자

주식별자 도출기준

1. 해당 업무에서 자주 이용되는 속성임
 2. 명칭, 내역 등과 같이 이름으로 기술되는 것들은 X
 3. 복합으로 주식별자로 구성할 경우 너무 많은 속성 X
- > 너무 많으면 인조식별자 생성한다.

식별자 관계

주식별자 : 자식의 주식별자로 부모의 주식별자 상속

- 부모로부터 받은 식별자를 자식엔터티의 주식별자로 이용하는 경우

- **강한 연결관계** 표현, **실선** 표기

- 식별자 관계로만 설정 시 주식별자 증가로 오류 유발

비식별자 : 부모 속성을 자식의 일반 속성으로 사용

1. 부모 없는 자식이 생성될 수 있는 경우

2. 부모와 자식의 생명주기가 다른 경우 (별도로소멸)
 3. 여러개의 엔터티가 하나의 엔터티로 통합되어 표현되었는데 각각의 엔터티가 별도의 관계를 가진 경우
 4. 자식엔터티에 별도의 주식별자를 생성하는 것이 더 유리한 경우
 5. SQL 문장이 길어져 복잡성 증가되는 것 방지
- **약한 연결관계** 표현, **점선** 표기
 - 비식별자 관계로만 설정 시 부모 엔터티와 조인하여 성능 저하
- =====

2장. 데이터 모델과 성능

성능 데이터 모델링 : DB 성능향상을 목적으로 설계단계의 데이터 모델링 때부터 정규화, 반정규화, 테이블통합, 테이블분할, 조인구조, PK, FK 등 여러 가지 성능과 관련된 사항이 데이터 모델링에 반영될 수 있도록 하는 것

분석/설계 단계에서 데이터 모델에 성능을 고려한 데이터 모델링을 수행할 경우 성능저하에 따른 재업무 비용을 최소화 할 수 있음.
데이터의 증가가 빠를수록 성능저하에 따른 성능개선비용은 기하급수적으로 증가하게 된다.

성능 데이터 모델링 고려사항 순서

1. 데이터 모델링을 할 때 정규화를 정확하게 수행
2. DB 용량산정을 수행한다.
3. DB에 발생하는 트랜잭션의 유형을 파악한다.
4. 용량과 트랜잭션의 유형에 따라 반정규화를 수행
5. 이력모델의 조정, PK/FK조정, 슈퍼/서브타입 조정
6. 성능관점에서 데이터 모델을 검증한다.

함수적 종속성 : 데이터들이 어떤 기준 값에 의해 종속되는 현상

- (-) **정규화** : 반복적인 데이터를 분리하고 각 데이터가 종속된 테이블에 적절하게 배치되도록 하는 것
- **1차 정규화** : 같은 성격, 내용 컬럼이 연속될 때

컬럼 제거, 테이블 생성

- **2차 정규화** : PK 복합키 구성일 때 부분적 함수 종속 관계 테이블 분리
- **3차 정규화** : PK가 아닌 일반 컬럼에 의존하는 컬럼 분리

(+) **반정규화** : 정규화된 엔터티, 속성, 관계에 대해 시스템의 **성능향상**과 개발과 운영의 **단순화**를 위해 **중복, 통합, 분리** 등을 수행하는 데이터 모델링의 기법
조회 시 디스크 I/O가 많거나 경로가 멀어 조인에 의한 성능 저하를 막기 위해 수행

일반적으로 **정규화시 입력/수정/삭제 성능이 향상**되며 **반정규화시 조인 성능이 향상**된다.

반정규화 절차

1. 반정규화 대상조사 (범위처리빈도수, 범위, 통계성)

1. 자주 사용되는 테이블에 접근하는 프로세스의 수가 많고 항상 일정한 범위만을 조회하는 경우
2. 테이블에 대량의 데이터가 있고 대량의 데이터 범위를 자주 처리하는 경우에 처리범위를 일정하게 줄이지 않으면 성능을 보장할 수 없는 경우
3. 통계성 프로세스에 의해 통계 정보를 필요로 할 때 별도의 통계테이블을 생성한다.
4. 테이블에 지나치게 많은 조인이 걸려 데이터를 조회하는 작업이 기술적으로 어려울 경우

2. 다른 방법유도 검토 (뷰, 클러스터링, 인덱스 조정)

1. VIEW 사용 : 지나치게 많은 조인이 걸려 데이터를 조회하는 작업이 기술적으로 어려울 경우 VIEW를 사용한다. (VIEW가 성능향상X)
2. 클러스터링 : 대량의 데이터처리나 부분처리에 의해 성능이 저하되는 경우 클러스터링을 적용하거나 인덱스를 조정함 (조회가 대부분일 때 클러스터링 적용)
3. **파티셔닝** : 대량의 데이터는 PK의 성격에 따라 부분적인 테이블로 분리할 수 있다. 파티셔닝 키에 의해 물리적 저장공간 분리
4. 캐시 : 응용 애플리케이션에서 로직을 구사하는 방법을 변경함으로써 성능을 향상시킬 수 있다.

3. 반정규화 적용

3-1. 테이블 반정규화

테이블 병합(1:1관계, 1:M관계, 슈퍼/서브타입)

1. 1:1관계를 통합하여 성능향상
2. 1:M관계를 통합하여 성능향상
3. **슈퍼/서브** 관계를 통합하여 성능향상

테이블분할(수직분할, 수평분할)

1. **수직분할** : 칼럼단위 테이블을 디스크 I/O를 분산처리하기 위해 테이블을 1:1로 분리하여 성능향상
2. **수평분할** : 로우단위로 집중 발생하는 트랜잭션을 분석하여 디스크 I/O 및 데이터 접근의 효율성을 높여 성능을 향상하기 위해 **로우단위로 테이블을 쪼개**

테이블 추가

1. **중복** : 다른 업무이거나 서버가 다른 경우 동일한 테이블구조를 중복하여 **원격조인을 제거하여 성능 향상**
2. **통계** : SUM, AVG 등을 미리 수행하여 계산해 둬으로써 조회 시 성능을 향상
3. **이력** : 이력테이블 중에서 마스터 테이블에 존재하는 레코드를 중복하여 이력테이블에 존재시켜 성능 향상
4. **부분** : 하나의 테이블의 전체 칼럼 중 자주 이용하는 집중화된 칼럼들이 있을 때 디스크 I/O를 줄이기 위해 해당 칼럼들을 모아놓은 별도의 반정규화된 테이블을 생성

3-2. 칼럼 반정규화

1. **중복칼럼 추가** : 조인에 의해 처리할 때 성능저하를 예방하기 위해 중복된 칼럼을 위치시킴
2. **파생칼럼 추가** : 트랜잭션이 처리되는 시점에 계산에 의해 발생하는 성능저하를 예방하기 위해 미리 값을 계산하여 칼럼에 보관
3. **이력테이블 칼럼추가** : 대량의 이력데이터를 처리할 때 불특정 날 조회나 최근 값을 조회할 때 나타날 수 있는 성능저하를 예방하기 위해 이력테이블에 기능성 칼럼(최근값 여부, 시작과 종료일자 등)을 추가함

4. 응용시스템 오작동을 위한 컬럼 추가 :

업무적으로는 의미가 없지만 사용자의 실수로 원래 값으로 복구하기 원하는 경우 이전 데이터를 임시적으로 중복하여 보관하는 기법

5. PK에 의한 컬럼 추가 : 단일 PK 안에서 특정 값을 별도로 조회하는 경우 성능 저하 발생할 수 있어 일반속성으로 추가함

3-3. 관계 반정규화; 무결성 유지

중복관계 추가 : 데이터를 처리하기 위한 여러 경로를 거쳐 조인이 가능하지만 이 때 발생할 수 있는 성능저하를 예방하기 위해 추가적인 관계를 맺는 방법

로우 체이닝 : 로우의 길이가 너무 길어서 데이터 블록 하나에 데이터가 모두 저장되지 않고 두 개 이상의 블록에 걸쳐 하나의 로우가 저장되어 있는 형태

로우 마이그레이션 : 데이터블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식

로우 체이닝과 로우 마이그레이션이 발생하여 많은 블록에 데이터가 저장되면 DB 메모리에서 디스크 I/O가 발생할 때 많은 I/O가 발생하여 성능저하 발생 트랜잭션을 분석하여 적절하게 1:1관계로 분리함으로써 성능향상이 가능하도록 해야 한다.

PK에 의해 테이블을 분할하는 방법(파티셔닝)

1. RANGE PARTITION : 대상 테이블이 날짜 또는 숫자값으로 분리가 가능하고 각 영역별로 트랜잭션이 분리되는 경우 ex) 요금_0401
2. LIST PARTITION : 지점, 사업소 등 핵심적인 코드값으로 PK가 구성되어 있고 대량의 데이터가 있는 테이블의 경우 ex) 고객_서울
3. HASH PARTITION : 지정된 HASH 조건에 따라 해시 알고리즘이 적용되어 테이블이 분리

테이블에 대한 수평/수직분할의 절차

1. 데이터 모델링을 완성한다.

2. DB 용량산정을 한다.

3. 대량 데이터가 처리되는 테이블에 대해 트랜잭션 처리 패턴을 분석한다.

4. 컬럼 단위로 집중화된 처리가 발생하는지, 로우 단위로 집중화된 처리가 발생하는지 분석하여 집중화된 단위로 테이블을 분리하는 것을 검토한다.

- 컬럼 많음 -> 1:1 분리

- 데이터 많음 -> 파티셔닝

슈퍼/서브 타입 모델 : 업무를 구성하는 데이터를 공통과 차이점의 특징을 고려하여 효과적 표현.

논리적 모델

- 슈퍼 타입 : 공통 부분

- 서브 타입 : 공통으로부터 상속받아 다른 엔터티와 차이가 있는 속성

슈퍼/서브 타입 데이터 모델의 변환기술

1. 개별로 발생하는 트랜잭션에 대해서는 개별 테이블로 구성 (OneToOne Type)
2. 슈퍼타입+서브타입에 대해 발생하는 트랜잭션에 대해서는 슈퍼+서브타입 테이블로 구성 (Plus Type)
3. 전체를 하나로 묶어 트랜잭션이 발생할 때는 하나의 테이블로 구성 (Single Type, All in One Type)

인덱스 특성을 고려한 PK/FK DB 성능향상

인덱스의 특징은 여러 개의 속성이 하나의 인덱스로 구성되어 있을 때 앞쪽에 위치한 속성의 값이 비교자로 있어야 좋은 효율을 나타낸다. 앞쪽에 위치한 속성의 값이 가급적 '=' 아니면 최소한 범위 'BETWEEN' '<>' 가 들어와야 효율적이다.

분산 DB

1. 여러 곳으로 분산되어있는 DB를 하나의 가상 시스템으로 사용할 수 있도록 한 DB
2. 논리적으로 동일한 시스템에 속하지만, 컴퓨터 네트워크를 통해 물리적으로 분산되어 있는 데이터집합

분산 DB를 만족하기 위한 6가지 투명성

1. 분할 투명성(단편화) : 하나의 논리적 Relation이

여러 단편으로 분할되어 각 사본이 여러 site에 저장

2. **위치 투명성** : 사용하려는 데이터의 저장 장소 명시 불필요, 위치정보가 시스템 카탈로그에 유지

3. **지역사상 투명성** : 지역 DBMS와 물리적 DB 사이의 Mapping 보장

4. **중복 투명성** : DB 객체가 여러 stie에 중복 되어 있는지 알 필요가 없는 성질

5. **장애 투명성** : 구성요소의 장애에 무관한 트랜잭션의 원자성 유지

6. **병행 투명성** : 다수 트랜잭션 동시 수행시 결과의 일관성 유지, TimeStamp, 분산 2단계 Locking 이용

분산 DB 장-단점

장점 : 지역 자치성, 신뢰성 가용성, 효율성 융통성, 빠른 응답속도, 비용절감, 각 지역 사용자 요구 수용

단점 : 비용증가, 오류의 잠재성 증대, 설계 관리의 복잡성, 불규칙한 응답 속도, 통제의 어려움, 데이터 무결성 위협

분산 DB 적용 기법

1. **테이블 위치 분산** : 설계된 테이블을 본사와 지사단위로 분산, 위치별 DB문서 필요

2. **테이블 분할 분산** : 각각의 테이블을 쪼개어 분산
- 수평분할 : 로우 단위로 분리, 지사별로 다를때, 중복X
- 수직분할 : 칼럼 단위로 분리, 각 테이블에 동일 PK 있어야 함.

3. **테이블 복제 분산** : 동일한 테이블을 다른 지역이나 서버에서 동시에 생성하여 관리하는 유형
- **부분복제** : 마스터 DB에서 테이블의 일부의 내용만 다른 지역이나 서버에 위치
- **광역복제** : 마스터 DB 테이블의 내용을 각 지역이나 서버에 존재

4. **테이블 요약 분산** : 지역 간에 또는 서버 간에 데이터가 비슷하지만 서로 다른 유형으로 존재하는 경우

- **분석요약** : 동일한 테이블 구조를 가지고 있으면서

분산되어 있는 동일한 내용의 데이터를 이용하여 통합된 데이터를 산출하는 방식, ex) 판매실적 지사A, 지사B

- **통합요약** : 분산되어 있는 다른 내용의 데이터를 이용하여 통합된 데이터를 산출하는 방식 ex) 판매실적 지사A:C제품, 지사B:D제품

분산 DB 설계를 고려해야 하는 경우

1. 성능이 중요한 사이트
2. 공통코드, 기준정보, 마스터 데이터의 성능향상
3. 실시간 동기화가 요구되지 않는 경우, Near Real Time 특징을 가지고 있는 경우
4. 특정 서버에 부하가 집중되어 부하를 분산
5. 백업 사이트 구성하는 경우

=====2과목=====

1장. SQL 기본

DB : 특정 기업이나 조직 또는 개인이 필요에 의해 데이터를 일정한 형태로 저장해 놓은 것을 의미한다.

DBMS : 효율적인 데이터 관리 뿐만 아니라 예기치 못한 사건으로 인한 데이터의 손상을 피하고, 필요시 필요한 데이터를 복구하기 위한 강력한 기능의 SW

SQL : 관계형 DB에서 데이터 정의, 조작, 제어를 위해 사용하는 언어

- **DML** : SELECT, INSERT, UPDATE, DELETE
- **DDL** : CREATE, ALTER, DROP, RENAME
- **DCL** : GRANT, REVOKE
- **TCL** : COMMIT, ROLLBACK

테이블 : DB 기본 단위, 데이터를 저장하는 객체

- 가로 = 행 = 로우 = 튜플 = 인스턴스
- 세로 = 열 = 컬럼

정규화 : 데이터의 정합성 확보와 데이터 입력/수정/삭제시 발생할 수 있는 이상현상을 방지하기 위해 중복 제거

기본키 : 테이블에 존재하는 각 행을 한 가지 의미로

특정할 수 있는 한 개 이상의 칼럼

외부키 : 다른 테이블의 기본키로 사용되고 있는 관계를 연결하는 칼럼

[DDL]

데이터 유형

- **CHAR(s)** : 고정 길이 문자열 정보. 최대 길이 만큼 공간 채움 'AA' = 'AA '
- **VARCHAR2(s)** : 가변 길이 문자열 정보. 할당된 변수 값의 바이트만 적용 'AA' != 'AA '
- **NUMBER** : 정수, 실수 등 숫자 정보
- **DATE** : 날짜와 시각 정보

- 테이블명은 다른 테이블의 이름과 중복되면 안 된다.
- 테이블 내의 **칼럼명은 중복될 수 없다.**
- 각 칼럼들은 , 로 구분되고 ; 로 끝난다.
- 칼럼 뒤에 데이터 유형은 꼭 지정되어야 한다.
- 테이블명과 칼럼명은 반드시 문자로 시작해야 한다.
- A-Z,a-z,0-9,_,,\$,#만 사용 가능

제약조건: 데이터의 무결성 유지

1. PRIMARY KEY(기본키) : UNIQUE & NOT NULL
2. UNIQUE KEY(고유키) : 고유키 정의
3. NOT NULL : NULL 값 입력금지
4. CHECK : 입력 값 범위 제한
5. FOREIGN KEY(외래키) : NULL 가능, 여러속성가능

테이블 생성

```
CREATE TABLE PLAYER (  
  PLAYER_ID CHAR(7) NOT NULL,  
  PLAYER_NAME VARCHAR2(20) NOT NULL);
```

테이블 구조 변경

- 추가 : **ALTER TABLE PLAYER ADD**(ADDRESS VARCHAR2(80));
- 삭제 : **ALTER TABLE PLAYER DROP COLUMN** ADDRESS;
- 수정 : **ALTER TABLE TEAM_TEMP MODIFY**

```
(ORIG_YYYY VARCHAR2(8) DEFAULT '20020129'  
NOT NULL);
```

제약조건 삭제 : **DROP CONSTRAINT** 조건명;

제약조건 추가 : **ADD CONSTRAINT** 조건명 조건 (칼럼명);

테이블명 변경 : **RENAME PLAYER TO** PLAYER_BACKUP;

테이블 삭제 : **DROP TABLE** PLAYER;

테이블 데이터 삭제 : **TRUNCATE TABLE** PLAYER;

컬럼명 변경 : **RENAME COLUMN** TEAM_ID TO T_ID;

[DML]

DDL 명령어의 경우 실행시 AUTO COMMIT 하지만 DML의 경우 COMMIT을 입력해야 한다.

INSERT INTO PLAYER (PLAYER) **VALUES** ('PJS');

UPDATE PLAYER **SET** BACK_NO = 60;

DELETE FROM PLAYER;

SELECT PLAYER_ID **FROM** PLAYER;

SELECT PLAYER **AS** "선수명" **FROM** PLAYER;

DISTINCT : 중복 시 1회만 출력

와일드카드

* : 모든

% : 모든

- : 한 글자

합성 연산자

문자와 문자 연결 : ||

[TCL]

트랜잭션 : 밀접히 관련되어 분리될 수 없는 1개 이상의 DB 조작. 논리적 연산단위

COMMIT : 올바르게 반영된 데이터를 DB에 반영

ROLLBACK : 트랜잭션 시작 이전의 상태로 되돌림

COMMIT 되지 않은 모든 트랜잭션을 롤백함.

SAVEPOINT : 저장 지점

트랜잭션의 특성

1. **원자성** : 트랜잭션에서 정의된 연산들은 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않아야 함
2. **일관성** : 트랜잭션 실행 전 DB 내용이 잘못 되지 않으면 실행 후도 잘못 되지 않아야 함
3. **고립성** : 트랜잭션 실행 도중 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다.
4. **지속성** : 트랜잭션이 성공적으로 수행되면 DB의 내용은 영구적으로 저장된다.

```
SAVEPOINT SVPT1;
ROLLBACK TO SVPT1;
COMMIT;
```

연산자의 종류

- BETWEEN a AND b : a와 b 값 사이에 있으면 됨
- IN (list) : 리스트에 있는 값중 어느 하나라도 일치
- IS NULL : NULL 값인 경우 (Oracle은 VARCHAR2 빈 문자열을 NULL로 판단)
- IS NOT NULL : NULL 값이 아닌 경우
- NOT IN (list) : list의 값과 일치하지 않는다
- LIKE '비교문자열' : 비교문자열과 형태가 일치

```
SELECT PLAYER_NAME 선수명 FROM PLAYER
WHERE TEAM_ID = 'K2'; -> 팀ID가 K2인 사람
WHERE TEAM_ID IN ('K2','K7'); -> K2,K7인 사람
WHERE HEIGHT BETWEEN 170 AND 180;
-> 키가 170 ~ 180인 사람
WHERE POSITION IS NULL; -> 포지션 없는 사람
```

NULL 값과의 수치연산은 NULL 값을 리턴한다.
NULL 값과의 비교연산은 거짓(FALSE)를 리턴한다.

연산자 우선순위 : ()->NOT->비교연산자->AND->OR

ROWNUM : 원하는 만큼의 행을 가져올 때 사용

```
SELECT * PLAYER_NAME FROM PLAYER
WHERE ROWNUM = 1;
```

단일행 함수

1. SELECT, WHERE, ORDER BY 절에서 사용 가능
2. 행에 개별적 조작
3. 여러 인자가 있어도 결과는 1개만 출력
4. 함수 인자에 상수, 변수, 표현식 사용 가능
5. 함수 중첩 가능

문자형 함수

LOWER : 문자열을 소문자로
UPPER : 문자열을 대문자로
ASCII : 문자의 ASCII 값 반환
CHR : ASCII 값에 해당하는 문자 반환
CONCAT : 문자열1, 2를 연결
SUBSTR : 문자열 중 m위치에서 n개의 문자 반환
LENGTH : 문자열 길이를 숫자 값으로 반환

```
CONCAT('RDBMS',' SQL') -> 'RDBMS SQL'
SUBSTR('SQL Expert',5,3) -> 'Exp'
LTRIM('xxxYYZZxYZ','x') -> 'YYZZxYZ'
TRIM('x' FROM 'xxYYZZxYZxx') -> 'YYZZxYZ'
```

숫자형 함수

SIGN(숫자) : 숫자가 양수면 1 음수면 -1 0이면 0 반환
MOD(숫자1, 숫자2) : 숫자1을 숫자2로 나누어 나머지 반환
CEIL(숫자) : 크거나 같은 최소 정수 반환
FLOOR(숫자) : 작거나 같은 최대 정수 리턴

```
ROUND(38.5235,3) -> 38.524
TRUNC(38.5235,3) -> 38.523
```

날짜형 함수

SYSDATE : 현재날짜와 시각 출력
EXTRACT : 날짜에서 데이터 출력
TO_NUMBER(TO_CHAR(d,'YYYY')) : 연도를 숫자로 출력
1 = 하루, 1/24 = 1시간, 1/24/60 = 1분

NULL 관련 함수 (NULL은 0 또는 공백 아님)

- **NVL(식1,식2)** : 식1의 값이 NULL 이면 식2 출력.
공집합을 바꿔주진 않음
- **NULLIF(식1,식2)** : 식1이 식2와 같으면 NULL을
아니면 식1을 출력
- **COALESCE(식1,식2)** : NULL이 아닌 최초의
표현식, 모두 NULL이면 NULL 반환
e.g. COALESCE(NULL, NULL, 'abc') -> 'abc'

다중행 집계 함수

1. 여러 행들의 그룹이 모여서 그룹당 단 하나의
결과를 돌려주는 함수이다.
2. GROUP BY 절은 행들을 소그룹화 한다.
3. SELECT, HAVING, ORDER BY 절에 사용 가능
 - ALL : Default 옵션. 생략 가능
 - DISTINCT : 같은 값을 하나의 데이터로 간주 옵션

COUNT(*) : NULL 포함 행의 수

COUNT(표현식) : NULL 제외 행의 수

SUM, AVG : NULL 제외 합계, 평균 연산

STDDEV : 표준 편차

VARIAN : 분산

MAX, MIN : 최대값, 최소값

GROUP BY, HAVING 절의 특징

1. GROUP BY 절을 통해 소그룹별 기준을 정한 후,
SELECT 절에 집계 함수를 사용한다.
2. 집계 함수의 통계 정보는 NULL 값을 가진 행을
제외하고 수행한다.
3. GROUP BY 절에서는 ALIAS 사용 불가
4. 집계 함수는 WHERE 절에 올 수 없다.
5. HAVING 절에는 집계함수를 이용하여 조건 표시
6. HAVING 절은 일반적으로 GROUP BY 뒤에 위치

SEARCHED_CASE_EXPRESSION

CASE WHEN LOC = 'a' THEN 'b'

SIMPLE_CASE_EXPRESSION

CASE LOC WHEN 'a' THEN 'b'

= DECODE(LOC, 'a', 'b')

ELSE NULL이 생략되어 있음.

ORDER BY 특징

1. SQL 문장으로 조회된 데이터들을 다양한 목적에
맞게 특정한 칼럼을 기준으로 정렬하여 출력하는데
사용한다.
2. ORDER BY 절에 칼럼명 대신 ALIAS 명이나 칼럼
순서를 나타내는 정수도 사용 가능하다.
3. DEFAULT 값으로 오름차순(ASC)이 적용되며
DESC 옵션을 통해 내림차순으로 정렬이 가능하다.
4. SQL 문장의 제일 마지막에 위치한다.
5. SELECT 절에서 정의하지 않은 칼럼 사용 가능

Oracle에서는 NULL을 가장 큰 값으로 취급하며 SQL
Server에서는 NULL을 가장 작은 값으로 취급한다.

SELECT 문장 실행 순서

SELECT ALIAS -> **FROM** -> **WHERE** -> **GROUP**
BY -> **HAVING** -> **SELECT** -> **ORDER BY**

메모리에 모든 칼럼 올리므로 ORDER BY에서
SELECT에 정의 안된 칼럼 써도 됨.

SQL Server의 WITH TIES

SELECT TOP(2) WITH TIES ENAME, SAL
FROM EMP

ORDER BY SAL DESC;

급여가 높은 2명을 내림차순으로 출력하는데, 같은
급여를 받는 사원은 같이 출력한다.

JOIN : 두 개 이상의 테이블들을 연결 또는 결합하여
데이터를 출력하는 것

일반적으로 행들은 PK나 FK 값의 연관에 의해
JOIN이 성립된다. 어떤 경우에는 PK, FK 관계가
없어도 논리적인 값들의 연관만으로 JOIN이
성립가능하다.

5가지 테이블을 JOIN 하기 위해서는 최소 4번의 JOIN
과정이 필요하다. (N-1)

EQUI JOIN : 2개의 테이블 간에 칼럼 값들이 서로
정확하게 일치하는 경우에 사용, 대부분 PK, FK의
관계를 기반으로 한다.

```
SELECT PLAYER.PLAYER_NAME
FROM PLAYER
```

위 SQL처럼 컬럼명 앞에 테이블 명을 기술해줘야 함

NON EQUI JOIN : 2개의 테이블 간에 칼럼 값들이 서로 정확하게 일치하지 않는 경우에 사용
'=' 연산자가 아닌 BETWEEN, >, <= 등 연산자 사용

```
SELECT E.ENAME, E.JOB, E.SAL, S.GRADE
FROM EMP E, SALGRADE S
WHERE E.SAL BETWEEN S.LOSAL AND S.HSAL;
위는 E의 SAL의 값을 S의 LOSAL과 HSAL 범위에서 찾는 것이다.
```

=====

2장. SQL 활용

집합 연산자 : 두 개 이상의 테이블에서 조인을 사용하지 않고 연관된 데이터를 조회할 때 사용
SELECT 절의 칼럼 수가 동일하고 SELECT 절의 동일 위치에 존재하는 칼럼의 데이터 타입이 상호 호환할 때 사용 가능

일반 집합 연산자

1. UNION : 합집합(중복 행 1개로) 정렬
 2. UNION ALL : 합집합(중복 행도 표시) 정렬X
 3. INTERSECT : 교집합(중복 행 1개로)
 4. MINUS : 차집합(중복 행 1개로)
 5. CROSS JOIN : 곱집합(PRODUCT)
- ALIAS는 처음 테이블, 정렬은 마지막 테이블 기준

순수 관계 연산자 : 관계형 DB를 새롭게 구현

1. SELECT -> WHERE절로 구현
 2. PROJECT -> SELECT절로 구현
 3. NATURAL JOIN -> 다양한 JOIN으로 구현
 4. DIVIDE -> 사용x
- $\{a,x\}\{a,y\}\{a,z\} \text{ divdie } \{x,z\} = \{a\}$

FROM 절 JOIN 형태

1. INNER JOIN

2. NATURAL JOIN
3. USING 조건절
4. ON 조건절
5. CROSS JOIN
6. OUTER JOIN

INNER JOIN : JOIN 조건에서 동일한 값이 있는 행만 반환, USING이나 ON 절을 필수적으로 사용

NATURAL JOIN : 두 테이블 간의 동일한 이름을 갖는 모든 칼럼들에 대해 EQUI JOIN 수행,
NATURAL JOIN이 명시되면 추가로 USING, ON, WHERE 절에서 JOIN 조건을 정의할 수 없다, SQL Server는 지원X

USING 조건절

같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 EQUI JOIN을 할 수 있다, JOIN 칼럼에 대해서 ALIAS나 테이블 이름과 같은 접두사를 붙일 수 없다, SQL Server 지원x

ON 조건절

ON 조건절과 WHERE 조건절을 분리하여 이해가 쉬우며, 칼럼명이 다르더라도 JOIN 조건을 사용할 수 있는 장점이 있다, ALIAS나 테이블명 반드시 사용

CROSS JOIN = 카티시안 곱

양쪽 집합의 M*N건의 데이터 조합이 발생한다.

OUTER JOIN (LEFT, RIGHT, FULL)

JOIN 조건에서 동일한 값이 없는 행도 반환 가능하다, USING이나 ON 조건절 반드시 사용해야 함. SQL 식에서 (+) 안붙은 쪽으로 JOIN한다.

- **LEFT OUTER JOIN** : 먼저 표기된 좌측 테이블에 해당하는 데이터를 읽은 후, 나중 표기된 우측 테이블에서 JOIN 대상 데이터를 읽어 온다. 우측 값에서 같은 값이 없는 경우 NULL 값으로 채운다.
- **RIGHT OUTER JOIN** : LEFT OUTER JOIN의 반대
- **FULL OUTER JOIN** : 좌우측 테이블의 모든 데이터를 읽어 JOIN하여 결과를 생성한다. 중복 데이터는 삭제한다.

계층형 질의 : 테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해 사용

START WITH : 계층 구조 전개에의 시작 위치 지정
CONNECT BY : 다음에 전개될 자식 데이터 지정
PRIOR : **CONNECT BY** 절에 사용되며, 현재 읽은 칼럼을 지정한다. **PRIOR** 자식 = 부모 형태를 사용하면 계층구조에서 부모 데이터에서 자식 데이터(부모->자식) 방향으로 전개하는 순방향 전개를 한다. 반대는 역방향 전개
NOCYCLE : 동일한 데이터가 전개되지 않음
ORDER SIBLINGS BY : 형제 노드간의 정렬 수행
WHERE : 모든 전개를 수행한 후에 지정된 조건을 만족하는 데이터만 추출한다.(필터링)

LEVEL : 루트 데이터이면 1, 그 하위 데이터면 2, 리프 데이터까지 1씩 증가
CONNECT_BY_ISLEAF : 해당 데이터가 리프 데이터면 1, 그렇지 않으면 0
CONNECT_BY_ISCYCLE : 해당 데이터가 조상이면 1, 아니면 0 (CYCLE 옵션 사용했을 시만 사용 가능)
SYS_CONNECT_BY_PATH : 루트 데이터부터 현재 전개할 데이터까지의 경로를 표시한다.
CONNECT_BY_ROOT : 현재 전개할 데이터의 루트 데이터를 표시한다. 단항 연산자이다.

셀프 조인 : 한 테이블 내 두 칼럼이 연관 관계가 있을 때 동일 테이블 사이의 조인. **FROM** 절에 동일 테이블이 2번 이상 나타난다. 반드시 테이블 별칭을 사용해야 함.

서브 쿼리 : 하나의 SQL문안에 포함되어 있는 또 다른 SQL문, 알려지지 않은 기준을 이용한 검색에 사용.

- 서브쿼리를 괄호로 감싸서 사용한다.
- 서브쿼리는 단일 행 또는 복수 행 비교 연산자와 함께 사용 가능하다. 단일 행 비교 연산자는 서브쿼리의 결과가 반드시 1건 이하여야 하고 복수 행 비교 연산자는 결과 건수와 상관없다.

- 서브쿼리에서는 **ORDER BY**를 사용하지 못한다.
- **SELECT**, **FROM**, **WHERE**, **HAVING**, **ORDER BY**, **INSERT-VALUES**, **UPDATE-SET** 절에 사용 가능

단일 행 비교 연산자 : =, <, >, <> 등

다중 행 비교 연산자 : **IN**, **ALL**, **ANY**, **SOME** 등
스칼라 서브쿼리 : 한 행, 한 칼럼만을 반환하는 서브쿼리

동작 방식에 따른 서브쿼리 분류

- 비연관 서브쿼리 : 서브쿼리가 메인쿼리 칼럼 안가짐. 메인쿼리에 값 제공 목적
- 연관 서브쿼리 : 서브쿼리가 메인쿼리 칼럼 가짐

반환 데이터에 따른 서브쿼리 종류

- 단일행 서브쿼리 : 실행결과 1건 이하
- 다중행 서브쿼리 : 실행결과 여러 건
- 다중컬럼 서브쿼리 : 실행결과 컬럼 여러 개

인라인 뷰 : **FROM** 절에서 사용되는 서브쿼리. **ORDER BY**를 사용 가능

뷰 : 테이블은 실제로 데이터를 가지고 있는 반면, 뷰는 실제 데이터를 가지고 있지 않다. 가상 테이블이라고도 함.

실행 시점에 SQL 재작성하여 수행됨.

뷰 사용 장점

1. **독립성** : 테이블 구조가 변경되어도 뷰를 사용하는 응용 프로그램은 변경하지 않아도 된다.
2. **편리성** : 복잡한 질의를 뷰로 생성함으로써 관련 질의를 단순하게 작성할 수 있다.
3. **보안성** : 직원의 급여정보와 같이 숨기고 싶은 정보가 존재할 때 사용

```
CREATE VIEW V_PLAYER_TEAM AS SELECT ...  
;  
DROP VIEW V_PLAYER_TEAM;
```

ROLLUP : Subtotal을 생성하기 위해 사용.
Grouping Columns의 수를 N이라고 했을 때 N+1

Level의 Subtotal이 생성된다. 인수 순서에 주의

GROUPING : 집계 표시면1, 아니면0

CUBE : 결합 가능한 모든 값에 대하여 다차원 집계를 생성, ROLLUP에 비해 시스템에 부하 심함.
 2^N

$CUBE(A, B) = GROUPING SETS(A, B, (A, B), ())$

GROUPING SETS : 인수들에 대한 개별 집계를 구할 수 있다. 다양한 소계 집합 생성 가능

윈도우 함수 : 행과 행간의 관계를 정의하거나 행과 행간을 비교, 연산하는 함수

[순위 관련 함수]

- **RANK** : 동일한 값에 대해서는 동일한 순위를 부여 (1,2,2,4)
- **DENSE_RANK** : 동일한 순위를 하나의 등수로 간주(1,2,2,3)
- **ROW_NUMBER** : 동일한 값이라도 고유한 순위 부여 (1,2,3,4)

[집계 관련 수함]

- **SUM** : 파티션별 윈도우의 합 구할 수 있다.
e.g. 같은 매니저를 두고 있는 직원들의 월급 합
- **MAX, MIN** : 파티션별 윈도우의 최대,최소 값을 구할 수 있다. e.g. 같은 매니저를 두고 있는 직원들 중 최대 값
- **AVG** : 원하는 조건에 맞는 데이터에 대한 통계 값
e.g. 같은 매니저 내에서 앞의 사번과 뒤의 사번의 평균
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
(현재 행을 기준으로 파티션 내에서 앞의 1건, 현재행, 뒤의 1건을 범위로 지정)
- **COUNT** : 조건에 맞는 데이터에 대한 통계 값
ex)본인의 급여보다 50 이하가 적거나 150 이하로 많은 급여를 받는 인원수

[행 순서 관련 함수] - SQL Server 지원X

- **FIRST_VALUE** : 파티션별 윈도우의 처음 값
- **LAST_VALUE** : 파티션별 윈도우의 마지막 값
- **LAG** : 파티션별 윈도우에서 이전 몇 번째 행의 값
- **LEAD** : 파티션별 윈도우에서 이후 몇 번째 행의 값

[비율 관련 함수]

- **RATIO_TO_REPORT** : 파티션 내 전체 SUM에 대한 행별 칼럼 값의 백분율을 소수점으로 구할 수 있다. $>0, \leq 1$
- **PERCENT_RANK** : 파티션별 윈도우에서 처음 값을 0, 마지막 값을 1로 하여 행의 순서별 백분율을 구한다. $0 \leq \leq 1$
- **CUME_DIST** : 현재 행보다 작거나 같은 건수에 대한 누적백분율을 구한다. $>0, \leq 1$
- **NTILE** : 파티션별 전체 건수를 인수 값으로 N등분한 결과를 구할 수 있다.

[DCL] : 유저 생성하고 권한을 제어할 수 있는 명령어

Oracle과 SQL Server의 사용자 아키텍처 차이

Oracle : 유저를 통해 DB에 접속을 하는 형태, ID와 PW 방식으로 인스턴스에 접속을 하고 그에 해당하는 스키마에 오브젝트 생성 등의 권한을 부여받게 됨

SQL Server : 인스턴스에 접속하기 위해 로그인이라는 것을 생성하게 되며, 인스턴스 내에 존재하는 다수의 DB에 연결하여 작업하기 위해 유저를 생성한 후 로그인과 유저를 매핑해 주어야 한다. Windows 인증 방식과 혼합 모드 방식이 존재함

시스템 권한 : 사용자가 SQL 문을 실행하기 위해 필요한 적절한 권한

- **GRANT** : 권한 부여
- **REVOKE** : 권한 취소

```
GRANT CREATE USER TO SCOTT;  
CONN SCOTT/TIGER(ID/PW)  
CREATE USER PJS IDENTIFIED BY KOREA7;  
GRANT CREATE SESSION TO PJS;
```

GRANT CREATE TABLE TO PJS;
REVOKE CREATE TABLE FROM PJS;

모든 유저는 각각 자신이 생성한 테이블 외에 다른 유저의 테이블에 접근하려면 해당 테이블에 대한 오브젝트 권한을 소유자로부터 부여받아야 한다.

ROLE : 유저에게 알맞은 권한들을 한 번에 부여하기 위해 사용하는 것

CREATE ROLE LOGIN_TABLE;
GRANT CREATE TABLE TO LOGIN_TABLE;
DROP USER PJS CASCADE;
CASCADE : 하위 오브젝트까지 삭제

절차형 SQL : SQL문의 연속적인 실행이나 조건에 따른 분기처리를 이용하여 특정 기능을 수행하는 저장 모듈을 생성할 수 있다. Procedure, User Defined Function, Trigger 등이 있음

저장 모듈 : PL/SQL 문장을 DB 서버에 저장하여 사용자와 애플리케이션 사이에서 공유할 수 있도록 만든 일종의 SQL 컴포넌트 프로그램, 독립적으로 실행되거나 다른 프로그램으로부터 실행될 수 있는 완전한 실행 프로그램

PL/SQL 특징

1. Block 구조로 되어있어 각 기능별로 모듈화 가능
2. 변수, 상수 등을 선언하여 SQL 문장 간 값을 교환
3. IF, LOOP 등의 절차형 언어를 사용하여 절차적인 프로그램이 가능하도록 한다.
4. DBMS 정의 에러나 사용자 정의 에러를 정의하여 사용할 수 있다.
5. PL/SQL은 Oracle에 내장되어 있으므로 호환성 굳
6. 응용 프로그램의 성능을 향상시킨다.
7. Block 단위로 처리 -> 통신량을 줄일 수 있다.

DECLARE : BEGIN~END 절에서 사용될 변수와 인수에 대한 정의 및 데이터 타입 선언부

BEGIN~END : 개발자가 처리하고자 하는 SQL문과 여러 가지 비교문, 제어문을 이용 필요한 로직 처리

EXCEPTION : BEGIN~END 절에서 실행되는

SQL문이 실행될 때 에러가 발생하면 그 에러를 어떻게 처리할지 정의하는 예외 처리부

T-SQL : 근본적으로 SQL Server를 제어하는 언어
CREATE Procedure schema_NAME.Procedure_name

Trigger : 특정한 테이블에 INSERT, UPDATE, DELETE와 같은 DML문이 수행되었을 때, DB에서 자동으로 동작하도록 작성된 프로그램, 사용자 호출이 아닌 DB 자동 수행

프로시저와 트리거의 차이점

프로시저 : BEGIN~END 절 내에 COMMIT, ROLLBACK과 같은 트랜잭션 종료 명령어 사용가능. EXECUTE 명령어로 실행

트리거 : BEGIN~END 절 내에 사용 불가. 생성 후 자동 실행.

- 끝 -