데이터 모델링 이해

데이터 모델링은 기업 업무에서 발생하는 데이터에 대해 물리적으로 데이터베이스화 하기 위한 과정으로 개념 데이터 모델링, 논리 데이터 모델링, 물리 데이터 모델링 그리고 객체지향 모델링으로 나눌 수 있다.

객체지향 모델링은 논리 데이터 모델링과 매우 유사하다. 그럼에도 차이점이 존재하는데 객체지향 모델링에서는 데이터와 프로세스를 같은 엔티티로 결합시켰다는 것이다.

개념 데이터 모델링은 주제별로 분류 가능한 업무를 분석한 후 핵심 엔터티(Entity)를 추출 하고 그들 간의 관계를 정의하여 전체 데이터 모델의 골격을 생성한다.

논리 데이터 모델링은 개념 데이터 모델링 단계에서 정의한 핵심 엔터티와 관계를 바탕으로 상세 속성을 정의하고 식별자를 확정하며 정규화와 같은 상세화 과정을 수행한다. 마지막으로 물리 데이터 모델링 단계에서는 논리 데이터 모델을 기반으로 목표하는 DBMS의 특성 및 구현 환경 등을 감안한 스키마(데이터 구조)를 일정한 기준과 규칙에 의해 도출하고 칼럼(Column)의 데이터 타입과 크기를 정의한다. 또한 데이터 사용량을 분석 예측하는 과정을 통해 효율적인 데이터베이스가 될 수 있도록 인덱스의 정의 및 역정규화 작업을 수행한다.

이러한 단계별 내용은 프로젝트 진행과도 일치한다. 물리 데이터 모델링 단계 후에 얻어진 스키마를 실제 데이터베이스로 생성하면 본격적인 애플리케이션 개발 단계로 넘어가게 된다.

데이터 모델의 목적

데이터 모델은 데이터베이스 설계에 대한 계획 또는 청사진이다. 설계자와 개발자, 사용자 등 모든 관련자들은 데이터 모델을 통해 구축될 시스템의 데이터 구조에 대해 형상을 이해하고, 요구 사항의 구현과 변경 등에 대해 원활한 의사소통을 도모하게 된다.

데이터 모델링 단계에서 업무를 잘못 이해했거나 관계를 잘못 정의한 것이 발견되었다면 해당하는 다이어그램과 일부 관련된 문서만 변경하면 된다. 그러나 데이터베이스와 응용 프로그램 개발이 완료된 후 이러한 오류를 발견하여 수정하려면 이와 관련된 많은 프로그램과 SQL문이 변경되어야 할 뿐만 아니라 데이터가 새로운 구조로 옮겨져야 하는 등 이러한 변경을 반영하는 데 많은 비용과 시간이 필요하게 되기 때문에 가능하면 설계 단계에서 조기에 오류들이 발견되고 정정되도록 원활한 의사소통이 필요하게 되고, 이러한 목적을 달성하는 데 데이터 모델은 최적의 수단으로 활용될 수 있다.

좋은 데이터 모델의 요소

|  |
| --- |
| **완전성(Completeness)** |
| 업무에서 필요로 하는 모든 데이터가 데이터 모델에 정의되어 있어야 한다. 이 기준이 충족되지 못하면 다른 어떤 평가 기준도 의미가 없어진다 |
| **중복 배제(Non-Redundancy)** |
| 하나의 테이블 내에 동일한 사실은 반드시 한 번만 기록하여야 한다. 예를 들면, 하나의 테이블에서 ‘나이’ 칼럼과 ‘생년월일’ 칼럼이 동시에 존재한다면 이것은 데이터 중복이라고 볼 수 있다. |
| **비즈니스 룰(Business Rules)** |
| 데이터 모델링 과정에서 도출되어 지고 규명되어지는 수많은 업무 규칙(Business Rules)을 데이터 모델에 표현하고 이를 해당 데이터 모델을 활용 하는 모든 사용자가 그 규칙에 대해 동일한 판단을 하고 데이터를 조작할 수 있게 되는 것이다 |
| **데이터 재사용(Data Reusability)** |
| 회사 전체 관점에서 공통 데이터를 도출하고 이를 전 영역에서 사용하기에 적절한 형태로 설계하여 시스템을 구축해야 한다. 부서 단위로 정보 관리를 하면 어플리케이션에 의존적으로 데이터가 생성되고 부서간 중복 데이터와 데이터 일관성 문제도 발생한다. 따라서 데이터 재사용을 향상시키기 위해 데이터 중심의 통합 모델로 설계되어야 한다. |
| **간결성(Elegance)** |
| 아무리 효율적으로 데이터를 잘 관리할 수 있더라도 그것의 사용, 관리 측면에서 복잡하다면 잘 만들어진 데이터 모델이라고 할 수 없다. 결국 간결한 모델의 기본적인 전제는 통합이다. 합리적으로 잘 정돈된 방법으로 데이터를 통합하여 데이터의 집합을 정의하고, 이를 데이터 모델로 잘 표현하여 활용한다면 웬만한 업무 변화에도 데이터 모델이 영향을 받지 않고 운용될 수 있게 된다 |

엔티티(Entity, 개체, 집합)

엔터티는 업무 활동상 지속적인 관심을 가지고 있어야 하는 대상으로서, 그 대상들 간에 동질성을 지닌 개체 집합이나 그들이 행하는 행위의 집합으로 정의할 수 있다. 집합에 들어갈 개체들의 동일한 성질을 어디까지 한정할 것인지를 결정하는 것이 중요하다.

예를 들면, ‘고객’이라는 집합을 ‘우리 상품을 구매한 사람이나 법인’으로 정의했다면 아직 구매를 한 적이 없는 잠재 고객이나 구매 상담자, 또 법인번호가 없는 단체나 개인 사업자 등은 이들과 동질성을 갖지 못한다. 그러나 이들이 현재 어떠한 방법으로든 관리가 되고 있거나 앞으로 관심을 갖고자 하는 범주에 해당한다면 집합의 정의를 더 확장해야만 이들을 고객 집합 내에 끌어들일 수 있다.

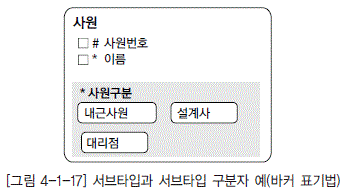
집합에 대한 정의 문제는 엔터티의 속성과 관계에도 영향을 미치고, 이것은 결과적으로 데이터 모델 전체의 구성에 영향을 미치게 되므로 집합에 대한 명확한 정의는 데이터 모델링에서 가장 핵심적인 사안이라 할 수 있다. 그러므로 엔터티를 정의할 때는 어떤 대상이 속하는지 여부를 명확하게 구분할 수 있도록 정의해야 한다.

존재 종속(Dependency)

만약 한 엔터티의 존재가 다른 엔터티(들)의 존재에 영향을 받는다면 이를 존재 종속(Existence-Dependent)이라 한다. 예를 들어, 어느 보험회사가 직원들에게는 그의 부양 가족들에게 보험 혜택을 준다고 하자. 부양가족은 해당 직원이 계속 근무하게 되는 경우에만 보험 혜택을 받으며, 그가 퇴사한 경우에는 혜택을 받을 수가 없다. 따라서 부양가족이라는 엔티티는 직원 엔티티에 존재 종속되어 있다.

서브타입(Sub/Supertype)

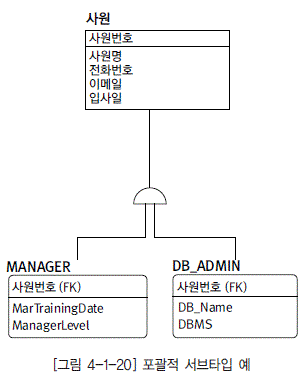
서브타입은 해당 엔티티의 부분 집합이다. 반드시 구분자를 가지고 있는 것은 아니며, 만약 구분자가 없다면 응용 프로그램에서 적절한 코드가 작성되어 구분해야 한다.



**배타적 vs. 포괄적 서브타입**

서브타입은 서브타입끼리 상호 배타적(Exclusive)이거나 포괄적(Inclusive)일 수 있다. 만약 배타적이라면 서

브타입은 반드시 어느 하나의 서브타입이어야 하지만, 포괄적이라면 어느 하나 또는 모두가 될 수 있다.

식별자(Unique Identifier)

엔터티의 각 개체들은 인스턴스라고 하는데, 인스턴스는 그들을 지칭하거나 식별해 주는 속성인 식별자(Unique Identifier)를 가지고 있다. 식별자는 하나 또는 그 이상의 속성으로 구성된다. 특히 두 개나 그 이상의 속성으로 이루어진 식별자를 복합 식별자(Composite Identifier)라 부른다.

속성(Attribute)

속성은 개체 집합의 특성을 설명하는 항목이라고 할 수 있다. 속성 구성을 통해 개체를 분명하게 특정 짓고 변별하게 한다.

각 속성은 가질 수 있는 값의 범위가 있다. 이를 그 속성의 도메인이라고 한다. 예를 들면 학점은 0.0에서 4.5 사이의 실수값이며, 주소는 100자 이내의 문자열로 지정할 수 있다. 서로 다른 개체 집합에 정의된 속성은 같은 도메인을 공유할 수 있다. 예를 들면, 학생 개체 집합에서 주소 속성에 해당하는 도메인과 교수 개체 집합에서의 주소 속성 도메인은 같은 값들의 범위를 가질 수 있다.

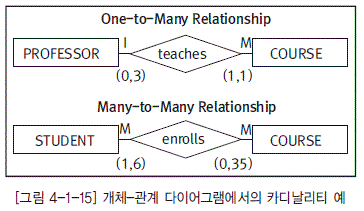
개체 집합 내에서 각각의 개체를 식별할 수 있도록 하나 (또는 그 이상의 속성들)로 이루어진 속성 집합을 식별자(Unique Identifier)라고 한다. 예를 들면 학번, 성명, 주소, 생년월일, 학과 등의 속성들로 이루어진 학생 엔터티에서 학번 속성은 식별자가 될 수 있다.

관계(Relationship)

관계는 엔터티와 엔터티 간 연관성을 표현하는 것으로 엔터티 간에 논리적으로 존재할 수 있는 수많은 관계 중에서 정말로 의미가 있고 관리할 가치가 있는 관계를 식별해 낸다는 것이 쉬운 일은 아니다.

카디날리티(Cardinality, 연결도)

카디날리티(Cardinality)란 관계에 참여하는 하나의 개체에 대해 다른 엔터티에서 몇 개의 개체가 참여하는지를 나타낸다. 예를 들면, 한 명의 학생이 1개 이상 6개 이하의 과목에 등록할 수 있다면 카디날리티는 (1, 6)이다. 한 명의 교수가 최대 3개의 과목을 가르칠 수 있다면 카디날리티는 (0, 3)이다. 카디날리티는 (Min, Max)의 값 한 쌍으로 표현하는데, 여기서 Min은 관계에 참여하는 개체의 최소 개수, Max는 관계에 참여하는 최대 개수를 각각 의미한다.

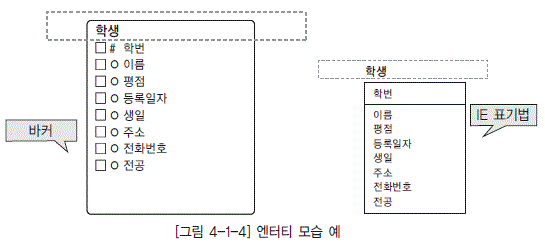


|  |  |
| --- | --- |
| 일대일(One To One, 1:1) | X에 속하는 한 개체는 Y에 속하는 한 개체에만 연결되며, Y에 속하는 한 개체도 X 에 속하는 한 개체에만 연결된다 |
| 일대다(One To Many, 1:M) | X에 속하는 한 개체는 Y에 속하는 한 개체에만 연결되며, Y에 속하는 한 개체는 X에 속하는 여러 개체와 연결된다 |
| 다대다(Many To Many, M:M) | X에 속하는 한 개체는 Y에 속하는 여러 개체와 연결될 수 있으며, Y에 속하는 한 개체도 X에 속하는 여러 개체와 연결될 수 있다 |

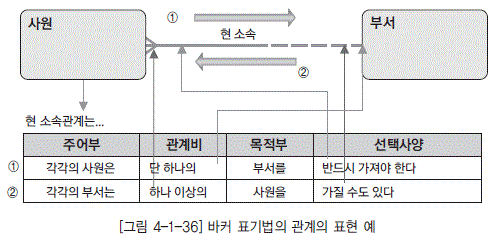
개체-관계 모델 기법(Entity-Relationship Modeling)

ERD란 최종 사용자의 관점에서 데이터 구조를 그림 형태로 묘사한 다이어그램이다. 개체, 관계, 속성이라는 3개의 요소로 다이어그램을 표현한다.

다이어그램을 표기하는 방법은 **바커 표기법**(Barker Notation)과 **정보공학 표기법**(IE Notation)이 대표적이다.



**바커 표기법 이해**



각 사원은 단 하나의 부서를 반드시 가져야 한다.

각 부서는 여러명의 사원을 가질 수도 있다.

논리 데이터 모델링

요구사항 분석서

정보 요구사항 개요

정보 요구사항이란 일상적으로 수행하는 업무의 개선 사항이나 신규 개발 사항으로 시스템을 통해 기능상의 목적을 달성하기 위해 요청하는 내용이다. 현업 사용자들이 이야기하는 정보 요구 사항을 IT 업무 담당자들은 처음부터 철저하게 이해하고, 무슨 내용이며 어떤 기능들을 요구하는지 정확하게 분석하기 위해 많은 시간과 노력을 집중해야 한다.

**요구사항의 수집**

관련 문서, 사용자 면담, 설문서, 워크숍, 현행 업무 처리 매뉴얼, 현행 시스템 관련 산출물을 통해 수집한다.

**요구사항의 분류**

요구사항은 기능 개선, 성능 개선, 외부 인터페이스 , 보안 개선 등 유형으로 분류될 수 있다.

**요구사항 정의서 작성**

요구사항 정의에는 기능, 성능, 인터페이스, 보안 등의 개선 요청사항 외에도 비기능적 정보 요구 사항을 포함하여 작성해야 한다. 비기능적 정보 요구 사항은 다음과 같다.

1. 시스템이 만족시켜야 하는 제약 조건(기술적 제약 조건, H/W, S/W와 관련된 제약 조건)

2. 시스템이 반드시 만족시켜야 하는 주요 성능 척도(반응 시간, 저장 능력, 동시 처리 능력)

3. 신뢰성, 확장성, 이식성, 보안

정보 요건을 상세화하여 요구사항을 정의하는 기법에는 프로세스 관점과 객체 관점으로 나눌 수 있다.

**프로세스 관점의 요구사항 상세화**

프로세스는 시작 시점과 종료 시점이 명확하고 실행 횟수를 셀 수 있는 업무 활동을 의미한다. 프로세스는 업무를 어떻게 수행하는가 보다는 어떤 업무가 수행되는지를 나타낸다. 따라서 입력(Input)과 출력(Output)이 있으며 입력을 출력으로 바꾸는 변환과정을 포함한다

프로세스 중심의 프로세스 목록과 프로세스의 업무 흐름도를 바탕으로 하여 프로세스 계층도(3-3-5)와 프로세스 정의서(3-3-6)를 작성한다.

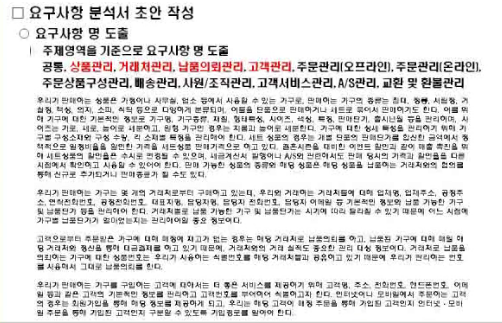
**객체지향 관점의 요구 사항 상세화**

유스케이스를 중심으로 각 사용자간의 기능적인 요구사항들을 정의한다.

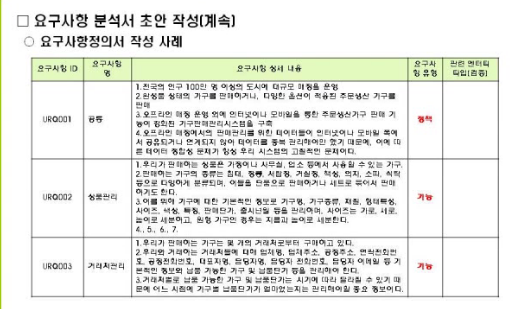
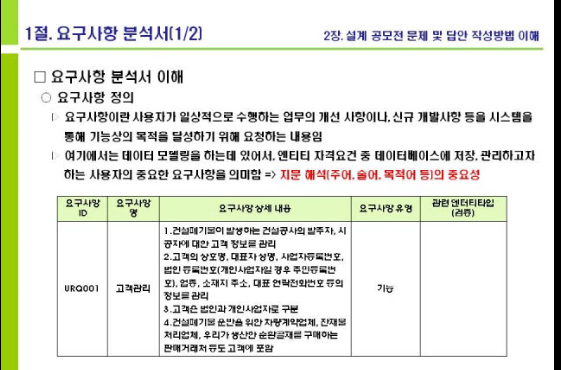
Use-case diagram

Class diagram

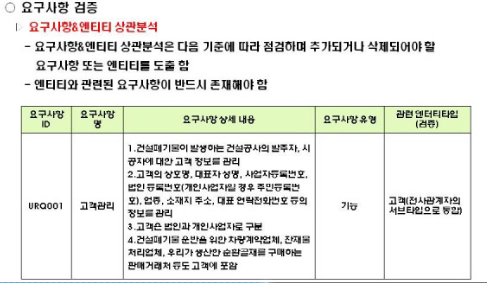
1. 요구사항 명 도출

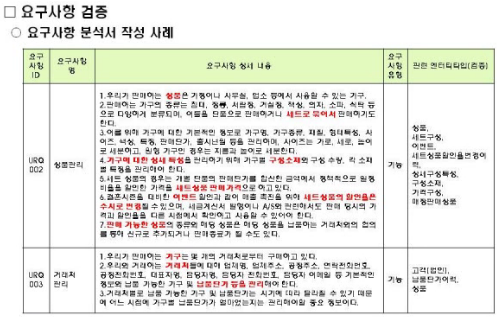


~관리한다. ~나눠야한다. 등을 **개조식**으로 요구사항을 정리한다.



1. 관련 엔티티 타입(검증)



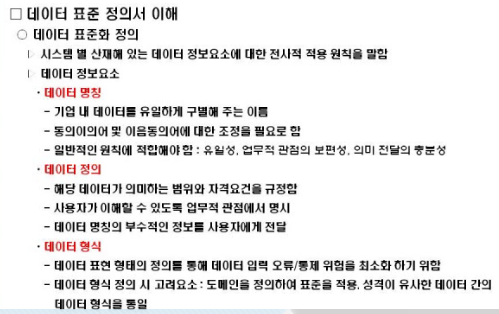


데이터 표준 정의서

데이터 표준에 대한 이해

논리 모델을 **만들어 가는 중에서** 계속 표준 정의서를 작성해 나가야 한다.

<http://www.dbguide.net/db.db?cmd=view&boardUid=12731&boardConfigUid=9&boardIdx=30&boardStep=1>

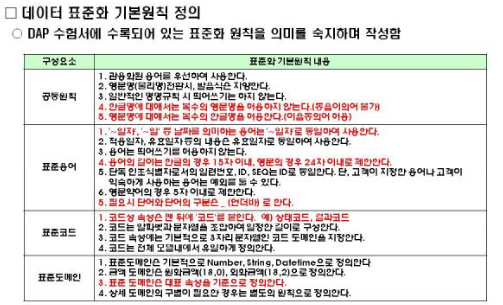


데이터 표준화 구성요소 (표준용어, 표준단어, 표준도메인)

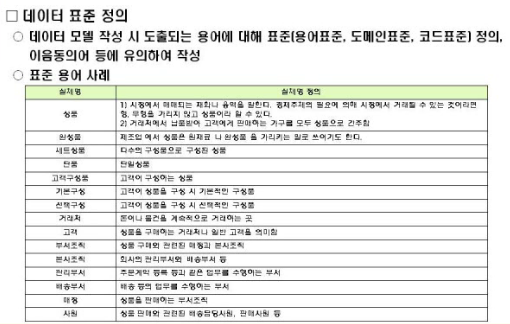
공모전에는 **‘표준용어**’, ‘**표준단어**’, ‘**표준도메인**’을 반드시 기술해야 한다.

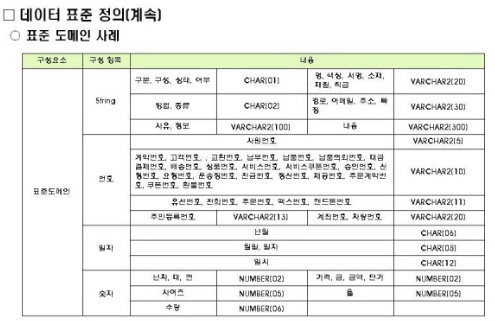


데이터 표준화 기본 원칙



데이터 표준 정의



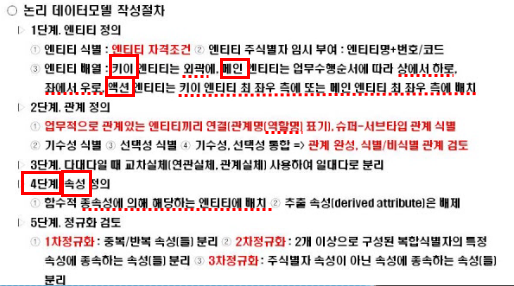




데이터 모델 작성

**논리 데이터모델 작성 절차**

**1.엔티티 정의 >> 2.관계 정의 >> 3.다대다 교차 해소 >> 4.속성 정의 >> 5.정규화**



**속성은 맨 마지막에 해줘야 한다. 잘못된 사례는 엔티티 정의를 하면서 속성을 기입하는 것이다.**

**엔티티 정의가 모두 끝나고, 관계를 도출이 끝난 이후에 속성을 정의하는 것이다. 식별자, FK 등만 관계에서 속성이 만들어질 뿐이다.**

**그리고 엔티티 정의도 엔티티가 모두 식별된 후에 해야 하는데 우리는 너무 섣불리 엔티티를 정의하고 있다. 이런 순서를 무시하고 작업을 하고 있다.**

**함수적 종속성이 있는 것은 해당 엔티티에 배치한다. 예를 들면 강사 이름 🡪 강사 엔티티에 속성으로 배치한다.**

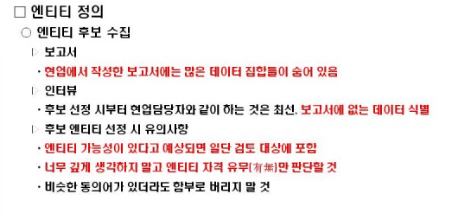
**추출 속성은 논리 모델에서 배제한다.**

1단계: 엔티티 정의

엔티티 후보 수집

너무 깊게 생각하지 말고, 후보 유무를 판단하여 결정하라.

이음동의어라고 버리지 말고 후보로 도출하라.

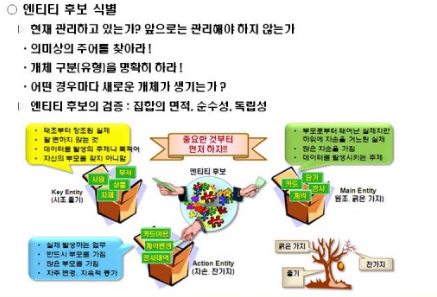


엔티티 후보 식별

의미상 주어, 개체 유형(Key, Main, Action )

* Key : 사원, 부서, 상품, 자재 등 잘 변화지 않는 것. 자신의 부모를 갖지 않음
* Main: 카드, 계약, 단가, 공사. 부모로부터 태어난 실체
* Action: 실제 발생하는 업무로 구체적인 값이 있다. 반드시 부모를 가짐.

후보의 검증: 집합의 면적, 순수성, 독립성

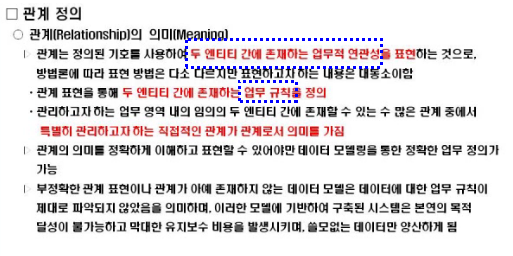


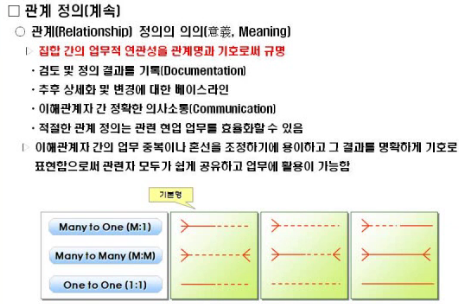


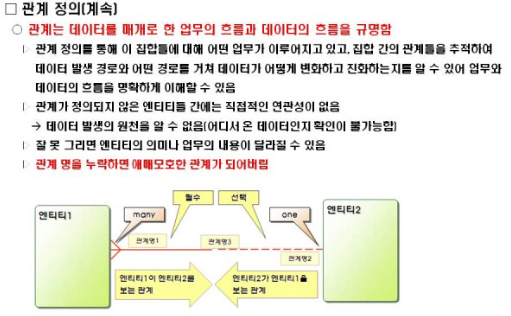
2단계: 관계 정의

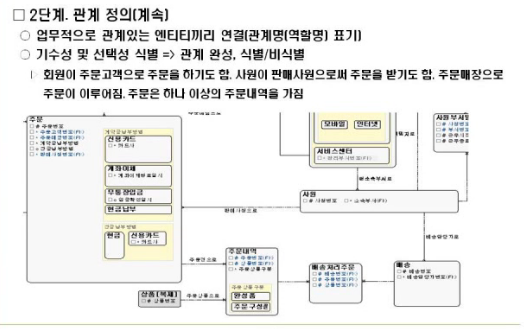
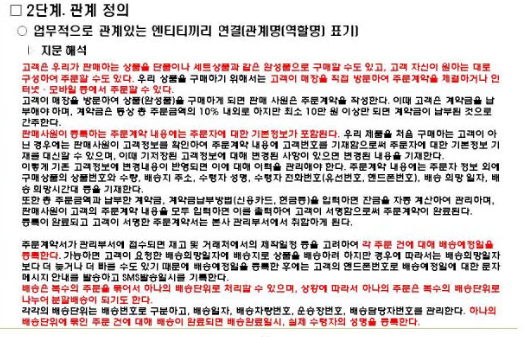
관계명, 기수성, 식별/비식별

관계명을 정확하게 표현되야 한다. 관계가 정의되지 않은 엔티티들 간에는 직접적인 연관성이 없다고 본다.

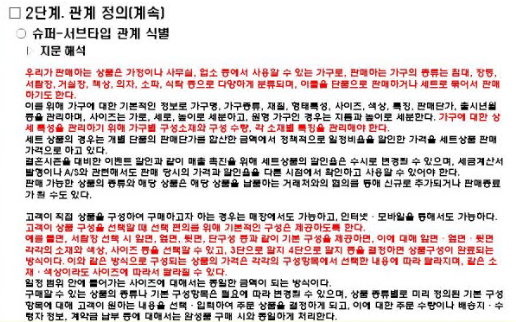




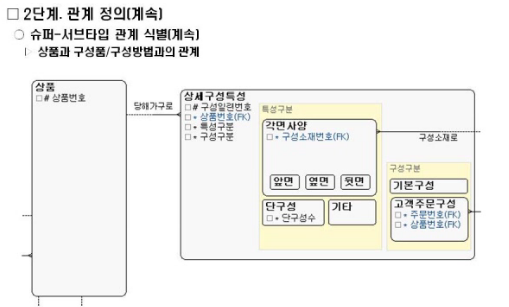




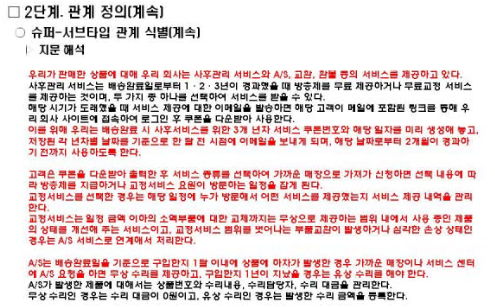
서브 타입





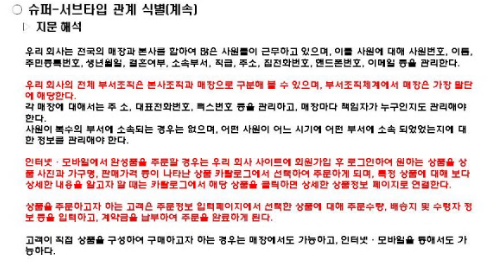


슈퍼 타입



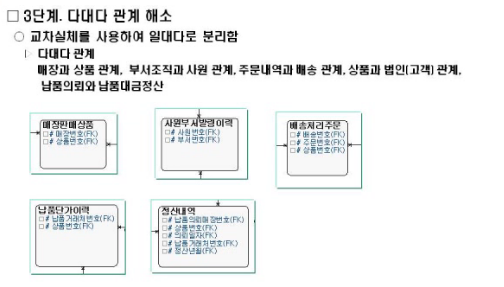




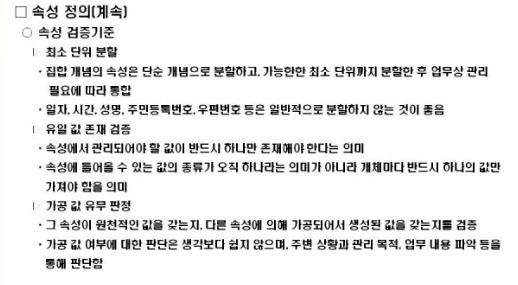
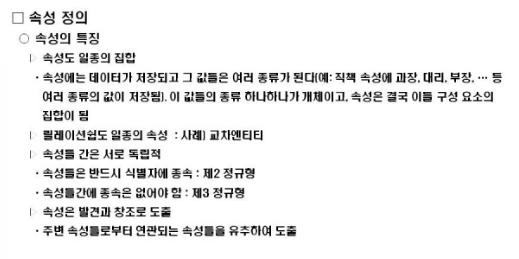


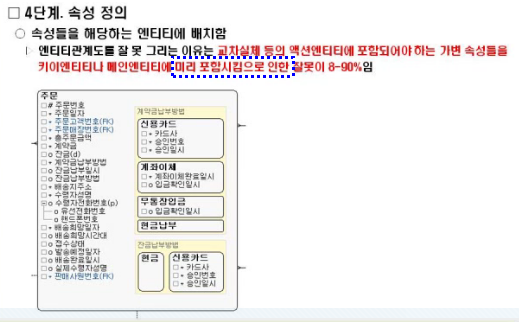


3단계: 다대다 해소



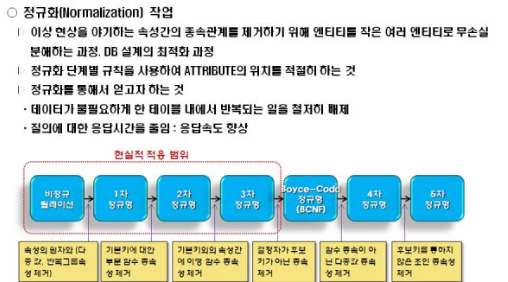
4단계: 속성 정의





5단계: 정규화 검증

**이상 현상**을 야기하는 속성들간의 종속관계를 해소하여 불필요한 데이터 방지 및 속도 개선을 목적으로 한다. 이상현상은 두개가 아닌 3개의 엔티티가 연관할 때 발생한다.

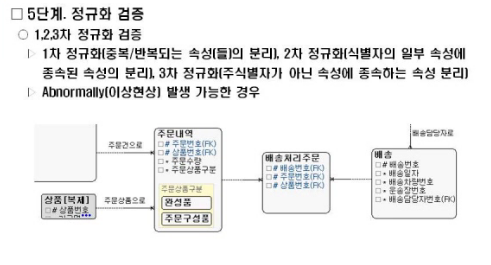




1차: 중복/반복되는 속성의 분리

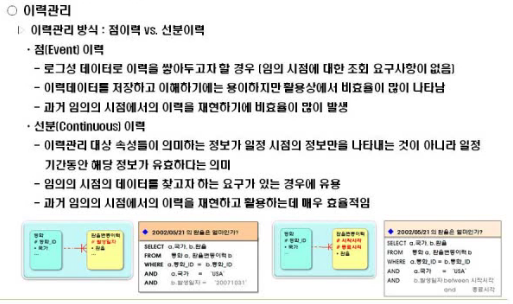
2차: 식별자 일부 속성에 종속된 속성 분리

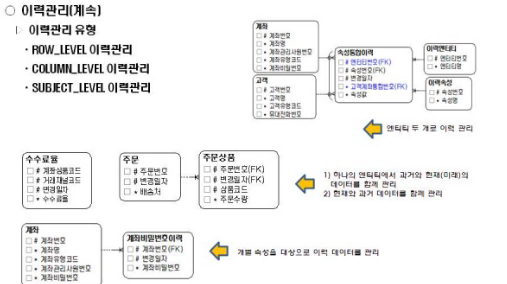
3차: 식별자 아닌 속성에 종속된 속성 분리



이력 관리

점(Event) 이력과 선분(Continuous) 이력





엔티티 상세화

식별자(UID, Unique Identifier) 확정

엔터티 내의 모든 인스턴스는 유일하게 구분되어야 한다. 이러한 유일성을 보장하기 위해서 필요한 것이 식별자이다.

핵심(Key) 엔티티에서 본질 식별자 정의하기

키 엔터티는 사원, 고객, 상품과 같이 부모를 가지지 않는 최상위 엔터티이다.

메인 엔티티는 해당 업무의 근본이 되는 엔티티이다. 하위에 수많은 엔티티를 거르리고 있는 최상위 조상이기도 하다.

행위(Actitivity) 엔티티에서 본질 식별자 정의하기

정규화(Normalization)

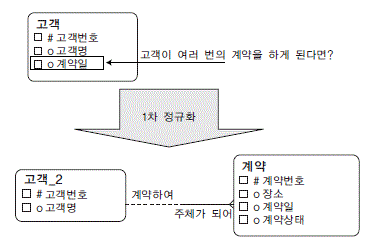
정규화는 논리적 데이터 모델을 일관성이 있고 중복을 제거하여 보다 안정성을 갖는 바람직한 자료구조로 만들기 위해 여러 단계를 거친다. 그 단계는 제 1차 정규형에서부터 제 5차 정규형과 BCNF(Boyce-Codd Normal Form)까지로 구성되어 있다. 대체로 적절하고 일관성을 유지하면서 중복이 없는 논리적 데이터 모델을 구축하는 데에는 흔히 3차 정규형이 사용된다.

1. 궁극적으로 칼럼 간, 레코드 간, 테이블 간에 중복되는 데이터들을 최소화할 수 있다. 이것이 정규 화의 최대 성과라고 할 수 있다
2. 전체적으로 NULL 값의 사용이 줄어들게 된다
3. 데이터베이스 코드를 작성하는데 복잡한 변환 과정과 조인 질의, NULL 값 처리들이 필요하다는 것은 그 데이터베이스 설계가 문제가 있다는 말이 기도 하다. 데이터에 중복된 값이 적고, 그 부모가 누구인지가 항상 명시되어 있는 상황에서 무결성을 지키기 위한 복잡한 코드를 사용할 필요가 없어진다
4. 중복된 값이 최소화되고 모든 정보들이 자기가 있어야 할 자리에 존재하게 되기 때문에 향후 발생 하게 될 모델 변화에도 유연하게 대처할 수 있다.

**1차 정규화(1NF, first normal form)**

하나의 인스턴스에 속성은 반드시 하나의 값을 가져야 한다.

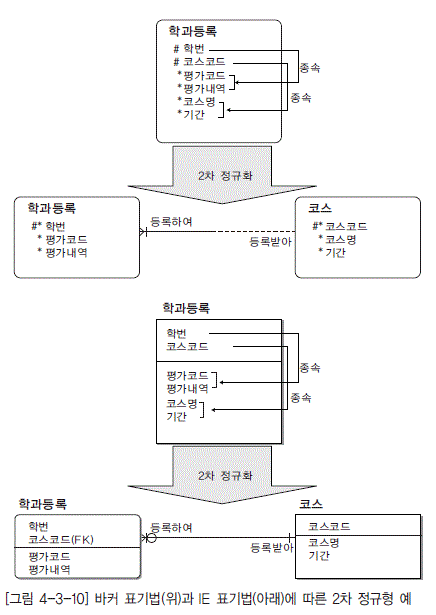
각 속성의 모든 값은 동일한 형식이어야 한다.



* 관계형 모델에서는 관계(Relation) 정의상 한 속성이 하나의 값만을 가져야 한다.
* 비정규형 관계가 관계로서의 모습을 갖추기 위해선 여러 개의 복합적인 의미를 가지고 있는 속 성이 분해되어 하나의 의미만을 표현하는 속성들로 분해되어야 한다

**2차 정규화(2NF, second normal form)**

식별자가 아닌 모든 속성들은 식별자에 완전 종속되어야 한다. 의미상의 주어 즉, 본질식별자를 알아야 식별자 부분 종속인지를 구분할 수 있다

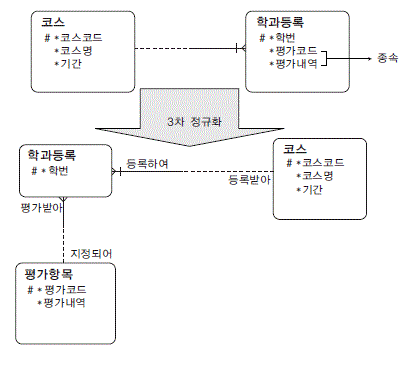


(2차 정규화 위반) 식별자가 ‘학번 + 코스코드’로 이루어진 학과등록 엔터티에서 학번 속성에 평가코드, 평가내역 속성들이 종속적이다. 코스코드 속성에 코스명, 기간 속성들이 종속적이다.

식별자에 종속되어 있지 않으면 잘못된 위치이며, 새로운 엔터티 즉, 상위 부모 엔터티를 생성하고 UID BAR를 상속받게 된다.

**3차 정규화 (3NF, Third Normal Form)**

2차 정규형을 만족하고 식별자를 제외한 나머지 속성들 간의 종속이 존재하면 안된다.

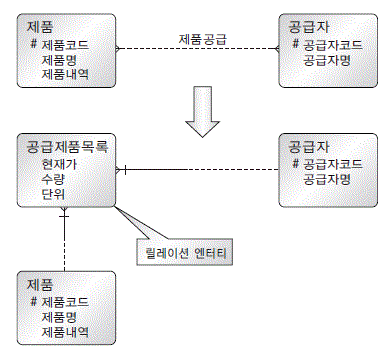


학과등록 엔터티에서 평가코드, 평가내역 속성들이 서로 간에 종속적이다. 즉, 평가내역 속성은 평가코드 속성에 종속적이다. 그렇기 때문에 이것은 3차 정규형을 위반하고 있는 것이다

**BCNF 정규화(이해안됨)**

M:M 관계 해소

이러한 관계는 데이터 모델이 아직 덜 완성된 모습이라고 할 수 있다. 대부분 기업의 업무 내용에 해당하는 데이터가 이러한 M:M 관계로 표현되고 있다.



제품 공급이라는 업무 내용에 의해서 ‘제품’과 ‘공급자’ 두 엔터티 간에 M:M 관계가 생겨났다. 이 관계를 해소하기 위해 ‘공급제품목록’이라는 새로운 엔터티가 만들어지고 또한 각각의 제품, 공급자와 1:M 의 관계를 부모로서 가지게 된다.

참조무결성(Reference Integrity) 규칙 정의

관계 테이블의 모든 외부 식별자 값은 관련 있는 관계 테이블의 모든 주 식별자 값이 존재해야 한다.

외부 식별자(FK)도 데이터 무결성에 관한 업무 규칙을 내포하고 있다.

사용자의 업무 규칙에 따라 적절한 규칙을 선택한다.

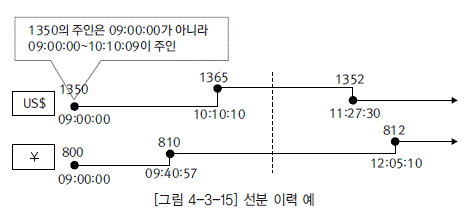
입력과 삭제 RI 규칙

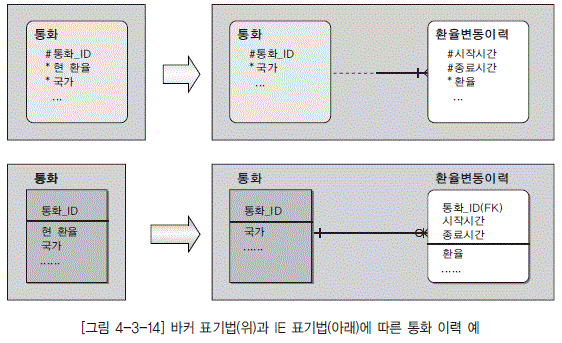
|  |  |  |  |
| --- | --- | --- | --- |
| **자식 인스턴스(Instance) 입력** | | **부모 인스턴스 삭제** | |
| **Dependent** | 대응되는 부모 실체에 인스턴스가 있는 경우에만 자식 실체에 입력을 허용 | **Restrict** | 대응되는 자식 실체의 인스턴스가 없는 경우에만 부모 실체 인스턴스 삭제를 허용 |
| Automatic | 자식 실체 인스턴스의 입력을 항상 허용하고, 대응되는 부모 건이 없는 경우 이를 자동 생성 | Cascade | 부모 실체 인스턴스의 삭제를 항상 허용하고, 대응되는 자식 실체의 인스턴스를 자동 삭제 |
| Nullify | 자식 실체 인스턴스의 입력을 항상 허용하고, 대응되는 부모 건이 없는 경우 자식 실체의 참조키 (FK)를 Null 값으로 처리 | Nullify | 부모 실체 인스턴스의 삭제를 항상 허용하고, 대응되는 자식 실체의 인스턴스가 존재하면, 그것의 참 조키(FK)를 Null 값으로 수정 |
| Default | 자식 실체 인스턴스의 입력을 항상 허용하고, 대응되는 부모 건이 없는 경우 참조키(FK)를 지정된 기본 값으로 처리 | Default | 부모 실체 인스턴스의 삭제를 항상 허용하고, 대응되는 자식 실체의 인스턴스가 존재하면, 그것의 참 조키(FK)를 기본 값으로 수정 |
| Customized | 특정한 검증 조건이 만족되는 경우에만 자식 실체 인스턴스의 입력을 허용 | Customized | 특정한 검증 조건이 만족되는 경우에만 부모 실체 인스턴스의 삭제를 허용 |
| No Effect | 자식 실체 인스턴스의 입력을 조건 없이 허용 | No Effect | 부모 실체 인스턴스 삭제를 조건 없이 허용 |

이력관리

이력은 선분이고 현재의 순간은 점이므로 선분을 관리해야 한다는 것은 결코 함부로 결정 해서는 안된다.

예를 들면 통화 데이터에서 관리되고 있던 환율에 대한 다음과 같은 업무 요구 사항이 발생하면 환율에 대한 이력을 관리하게 된다. 통화 기간과 관계없이 고정된 환율을 적용했던 요금 정책을 변동된 환율을 적용하여 실제적인 요금을 부과해야 한다.





발생 이력(Occurrence History)

어떤 이벤트가 발생할 때마다 이력 정보를 남기는 경우를 말한다. 예를 들면 접속기록이나 새로운 데이터데 등록할 때마다 그 시점을 기록한다.

변경 이력(Modification History)

데이터가 변경될 때마다 변경 전과 후의 차이를 확인해야 한다면 변경 이력을 남길 수 있다. 예를 들어, 고객이 주문을 하고서 주문 정보를 변경하였을 때 이전 주문과 변경된 새로운 주문 정보를 관리하기 위해서 변경된 새로운 주문 정보를 이력 정보로 남겨야 한다.

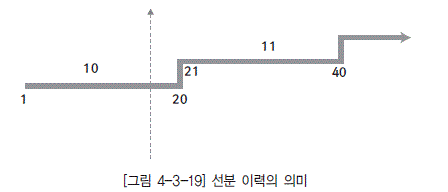
진행 이력(Progress History)

업무의 진행에 따라 관련 데이터를 남겨야만 하는 경우가 있다. 가장 대표적인 것이 주문 업무이다. 주문 업무 처리는 구매 신청 -> 입금 완료-> 배송 준비 중 -> 배송 중 -> 배송 완료 혹은 주문 취소 등과 같은 진행 상황이 있다. 대부분의 주문 업무 처리의 경우 각 단계가 언제 누구에 의해서 처리되고, 현재 단계는 무엇인지에 관한 정보가 필요한 경우가 많다. 이러한 경우 업무를 처리하기 위해서는 진행 이력이 중요하다. 물론 이런 형태는 변경 이력과 같은 형태로 관리 된다.

시점 vs. 선분 개념의 이력관리

시점 이력관리는 데이터 변경이 발생한 시각만을 관리하는 반면, 선분 이력관리는 변경 시작시점부터 그 상태의 종료 시점까지를 관리한다.

시점 관리를 하게 되면 ‘=’ 검색이 가능하지만, 선분 관리를 하면 ‘=’검색이 불가능하다. 따라서 ‘between’ 검색으로만 이력을 찾을 수 있다.



17이라는 지점을 선분 이력에서 찾는 방법은 1) 시작점<= 17 <= 종료점 또는 2) 17 Between 시작점 and 종료점을 사용해야 한다.

|  |
| --- |
| SELECT nm\_ip\_address\_single, nm\_ip\_address\_startrange, nm\_ip\_address\_finishrange, nm\_notes  FROM NETWORK\_ALLOW\_IP  WHERE nm\_ip\_address\_single BETWEEN '61.83.152.160' AND '61.83.152.165' |

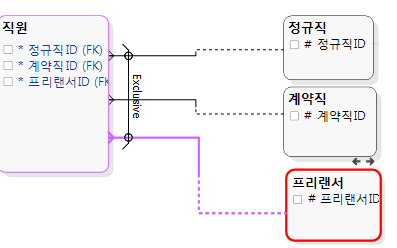
이력으로 남길 대상

인스턴스 레벨로 관리하는 경우 어떤 변경이 발생하면 행(row) 전체를 남기는 것이다.(ROW LEVEL)

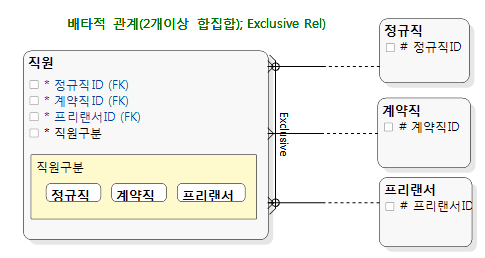
속성 레벨로 관리하는 경우, 변경된 속성만을 남기는 것이다.(COLUMN LEVEL)

배타적 관계

하나의 테이블에 다수의 테이블이 관계를 맺습니다. 이때 각 유사 테이블이 동시에 자식 테이블과 관계를 맺지 못하고 상호배타적으로 존재해야 하는 경우가 있습니다. 이 관계를 배타적 관계라고 합니다.



배타적 관계를 맺을 때, 해당 관계를 구분하는 ‘구분속성’을 정의할 수 있습니다.



속성 정의

속성의 기본 구성요소

1) 속성명

속성의 내용이나 목적이 무엇인지 알려주는 명사 또는 명사구이다.

해당 업무에서 일반적으로 사용하는 용어를 사용한다.

실체명은 속성 명으로 사용하지 말아야 한다.

필요시 표준 약어를 제정하여 속성 명을 생성하고 그 속성 명을 단 하나의 실체에만 속하도록 하는 것이 바람직하다.

2) 도메인

속성이 지닐 수 있는 값에 대한 업무적인 제약 조건으로 파악된 일련의 특성이다. 모든 영역에서 같은 도메인을 사용하는 것이 좋다. 속성이 기존 도메인 집합에 속해 있지 않는 경우에는 새로운 도메인을 추가한다. 도메인은 다음과 같은 속성들을 가진다.

데이터 타입

길이

허용 값(Permitted Value): 속성에 지정할 수 있는 모든 값들의 집합

디폴트 값 및 디폴트 알고리즘

3) 선택성

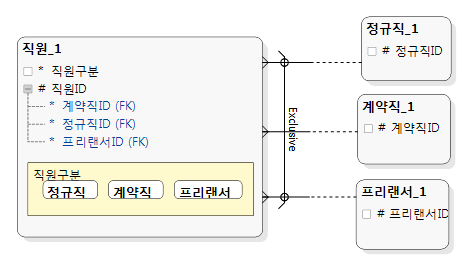
모든 건의 해당 속성이 반드시 값을 가져야 하는지 여부를 나타낸다.

선택성 조건 => 선택성이 다른 속성 값에 의해 영향을 받는 경우

필요조건 / 금지조건 / 무관계조건

배타적 속성

직원 테이블에서 직원 ID라는 컬럼을 가지고 있다. 이때 직원ID는 정규직ID, 계약직ID, 프리랜서ID 속성 컬럼 중에서 오직 하나를 취하게 됩니다. 직원ID 속성은 배타적 속성을 갖고 있다고 한다. 직원ID의 속성을 ‘Exclusive’로 속성범위를 변경하고, 관련된 속성들을 하위구조로 가져오면 배타적 속성관계를 표현합니다.



여기서 문제점은 물리설계로 넘어가면 직원ID만 남고, 3개의 FK ID들은 사라져 버럽니다. 이때 구분속성(직원구분) 정의해 두지 않으면 직원ID가 계약직인지, 정규직인지, 프리랜서인지를 확인할 수 없습니다. 따라서 이 경우는 ‘직원구분’이라는 서브타입을 만들어야 합니다.

**Cf. 병렬 관계**

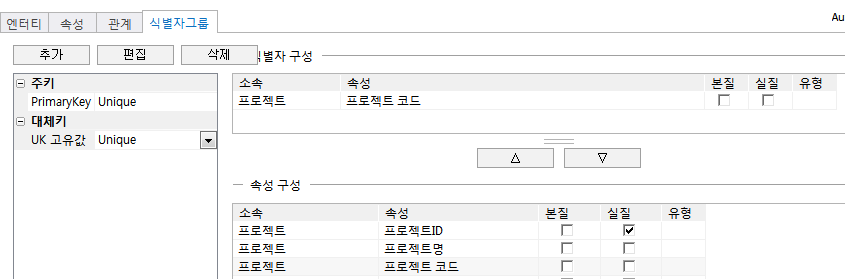
하나의 테이블에서 하나의 자식 테이블로 관계가 여러 개 가질 때를 병렬관계라고 한다.



Unique 속성

DA# **논리모델** 화면에서 ‘식별자 그룹’ 탭으로 이동한다.

UK식별자 그룹명을 추가한다. 해당 식별자에 해당하는 속성을 선택하여 등록처리 한다.



물리 데이터 모델링

타겟으로 하는 DB를 지정합니다.

ORACLE, SQLSERVER, SYSBASEASE 등등 실제 DBMS를 선택합니다.

물리 데이터 모델링 전환할 때 고려사항

* Super/Sub 관계의 엔터티를 몇 개의 테이블로 만들 것인가
* 배타적(Arc) 관계 엔터티의 외부키(Foreign Key)를 몇 개로 할 것인가
* 배타적 속성의 경우, 속성을 구분하게 하는 서브타입을 만들어야 한다
* 성능 향상을 위해 테이블을 추가해야 할 것인가 혹은 통합해야 할 것인가
* 통계 작업을 위해 합계(Summary) 테이블 같은 임시성 테이블을 몇 개로 할 것이며, 유일키를 무 엇으로 할 것인가
* 테이블의 칼럼을 다른 테이블에 중복할 것인가, 중복하면 어떤 애플리케이션이 관련되어 있는 가,
* 인덱스의 설정, 스냅샷(Snapshot) 또는 뷰(View) 등의 객체가 필요한가
* 분산 환경에서 테이블을 중복할 것인가, 중앙에 필요한 테이블을 따로 가져갈 것인가
* 데이터가 분산 환경에서 이동 시 문제를 어떻게 해결할 것인가

데이터 타입 DATA TYPE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **데이터 유형** | **범위** | **값** | **크기(bytes)** |
| 정수 | numeric/decimal | 10의 38승 |  |  |
| **bigint(8)** | 2의 63승 |  | 8 |
| **int(4)** | 2의 31승 | 21억4천7백 | 4 |
| smallint(2) | -32768~32767 |  | 2 |
| tinyint(1) | 0~255 |  | 1 |
| bit | 0 또는 1 |  |  |
| Numeric | Numeric(5,2) |  | 전체 자릿수와 소수 자릿수 |
| **Decimal** | decimal(5,2) |  | 전체 자릿수와 소수 자릿수 |
| 실수 | **float** | 1.79의 308승 |  | n값에 따라 다름 |
| **real** | **3.4의 38승** |  | 4 |
| 날짜 | date |  |  |  |
| datetime(8) | 밀리초(1/1000) |  |  |
| datetime2 | 마이크로초(1/백만) |  |  |
| Smalldatetime |  |  |  |
| smalldatetime | 1초단위 |  |  |
| 문자열 | char, varchar, varchar2 | 8000자 이하 |  |  |
| nchar, **nvarchar**, nvarchar2 | 4000자 이하, 유니코드 |  |  |
| text | 8000자 이상 |  | Varchar(**max**)로 대체  관련 함수 및 작업문:  **DATALENGTH**,  **PATINDEX**,  SUBSTRING,  TEXTPTR/TEXTVALID/WRITETEXT/UPDATETEXT/SET TEXTSIZE/READTEXT/TEXTVALID |
| ntext | 4000자 이상, 유니코드 |  | Nvarchar(**max**)로 대체 |
| 바이너리 | binary, varbinary | 8kb이하 |  |  |
| image | 8kb 초과 |  | **Varbinary**(**max**)로 대체,  2^31-1(2,147,483,647)바이트의 가변 길이 이진 데이터 |
| 화폐 | money | 2의 63승 |  |  |
| smallmoney | 214,748***.3647*** 까지 |  | 4 |
|  | Cursor |  |  | DECLARE, SET  OPEN, FETCH, CLOSE, DEALLOCATE  SP Output parameters  CUSRSOR\_STATUS 함수  SP함수들 (sp\_cursor\_list, sp\_describe\_cursor, sp\_describe\_cursor\_tables, sp\_describe\_cursor\_columns) |
|  | Sql\_variant | sql서버에서 지원하는 다양한 데이터 유형의 값을 저장할 수 있다. |  |  |

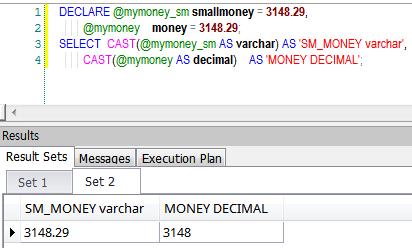
Real의 문제점은 SUM을 하면 float로 변환되면서 가비지 데이터가 들어간다. 따라서 float을 권장

Money 데이터 변환

**CAST(** @변수명 **AS** 변환할데이타유형**)**

Smallmoney를 varchar로, Money를 Decimal로 cast변환

SELECT CAST(@mymoney\_sm AS **Varchar**) , CAST(@mymoney AS **Decimal**);





데이터 표준화 명명🡪갱신필요

반드시 객체(명사)으로 짓는다. 동사형이나 액션, 기능으로 짓지 말자

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 테이블명 | 메인 엔티티 | TBL\_\*\*\*\*\*\_\*\*\*\*\* | |  | 파생 엔티티 | \*\*\*\*\_\*\*\*\*\* | |  | 교차 엔티티 | 부모명\_자식명\_\*\*\* | | 컬럼명 | 소문자를 사용하며 복합명사는 밑줄로 연결 |  | |  |  | 데이타유형\_\*\*\*\*\*\*\*\_\*\*\*\*\* | |  | 순수 식별키 | id\_테이블명 | |  | 코드키 | cd\_\*\*\*\* | |  | 명칭/이름 | nm\_\*\*\*\*\*\*\_name | |  | 구분(subtype) | sub\_\*\*\*\*\_identity | |  | 비고,메모, 대량 | tx\_\*\*\*\*\* | |  | 수/볼륨/양/순서 | no\_\*\*\*\* | |  | 날자, 시간, 요일 | dt\_\*\*\*\* | |  | 여부, 체크, 유형 | yn\_\*\*\*\* | |  | 이메일 | at\_contact\_email | |  | 비밀번호 | enc\_password | |  | 전화번호 | tel\_contact\_phone | |  | URL 링크, 참조 | url\_\*\*\*\*\*\_link | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DATA** | **pre-fix** | **TYPE** | **SIZE** | **IDENTITY** | **REMARK** |
| ID(pk) | id\_ | varchar | 12 |  | 공간차지 2KB |
| **int** | 4 | (1,1) | 2의 31승 |
| smallint |  |  | -32767 ~ 32,767 |
| tinyint |  |  | 0~255 |
| **bigint** | 8 | (1,1) | 2의 63승 |
| 코드명 | cd\_ | varchar2 | 5 | UNIQUE 속성 | 9만개 이하 |
| 표시순서 | no\_ | varchar | 12 |  |  |
| smallint |  |  |  |
| 명칭 | nm\_ | nvarchar |  |  | 2000자 |
| varchar2 | 80 |  | 2000자 |
| 아이디 |  | vachar | 100 |  |  |
| 파일경로 | url\_ | varchar2 | 500 |  | 가변 4KB |
| 메모, 비고 | tx\_ | nvarchar | MAX |  |  |
| nvarchar2 |  |  | 4000자 |
| ntext |  |  | 4000자 이상 |
| 수량 | no\_ | varchar2 | 10 |  | 10억 |
| 일자 | yy\_ | char | 8 |  | YYYYMMDD |
| char | 10 |  | YYYY-MM-DD |
| char | 12 |  | YYYYMMDDHHMM |
| char | 14 |  | YYYYMMDDHHMMSS |
| dt\_ | date |  | 일단위 | yyyymmdd |
| smalldatetime |  | 1초단위 | yyyymmdd hhmmss |
| datetime | 8 | 밀리세컨드 | yyyymmdd hhmmss[.nnn] |
| datetime2 |  | 마이크로세컨드 | yyyymmdd hhmmss[.nnnnnn] |
| tm\_ | time |  | 1/백만초 | hhmmss[.nnnnnn] |
| 연락처/이메일/비밀번호 | at\_ tel\_ pw\_ | varchar2 | 100 |  |  |
| nvarchar | 100 |  |  |
| varchar2 | 15 |  | XXXX-XXXX-XXXX |
| 여부 | yn\_ | char | 1 |  | 0~9, A~Z |
| bit |  |  | 0 or 1 |
| 서브타입 | sub\_ | char | 3 |  |  |
| 숫자 | int\_ | **int** | 4 | (1,1) | 2의 31승 |
| smallint |  |  | -32767 ~ 32,767 |
| tinyint |  |  | 0~255 |
| **bigint** | 8 | (1,1) | 2의 63승 |
| real\_ | float |  |  | 1.79의 308승 |
| real |  |  | 3.4의 38승 |
| 화폐 | money\_ | money |  |  | 2의 63승 |
| smallmoney |  |  | 214,748.3647까지 |

데이터타입 REAL 의 문제점 for SQL

1. **정수**를 다루는 경우, SUM에서도 근사치 적용의 이슈는 발생하지 않음

하지만, **SUM** 내장함수로 집계하면 real을 float로 형 변환하면서 소수점 아래가 이상한 값으로 채워진다. 하지만, ROUND(값, 2)로 하면 소숫자리 3자리에서 반올림하게 되어 엑셀값과 동일하다.

1. **7자리 이상의 수나 실수**를 다룰 때는 REAL타입 데이터는 값의 정확성이 제거되어 오류가 발생한다

결국 REAL 타입 데이터는 큰 숫자의 경우 깨지게 된다.

1. Real이라도 SUM에서 Round 3자리에서 반올림하면 올바른 값이 나옴

Real, Foat, Money vs. Decimal 테스트

13자리에 소수점 2자리를 가진 데이터(조)

데이터는 소수점 2자리를 가진 9조원에 가까운 화폐를 대상으로 테스트하였다.

엑셀 데이터(이상없음)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **REAL** | **FLOAT** | **MONEY** | **DECIMAL** |
| 1 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 2 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 3 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 4 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 5 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| 6 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| 7 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| 8 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| Total | 9876543120987.64 | 9876543120987.64 | 9876543120987.64 | 9876543120987.64 |

**DB Import**했을 경우, SQL로 보여지는 값**(표시가 이상함)**

|  |  |  |  |
| --- | --- | --- | --- |
| **REAL** | **FLOAT** | **MONEY** | **DECIMAL** |
| 1.234568E+12 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 1.234568E+12 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 1.234568E+12 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 1.234568E+12 | 1234567890123.45 | 1234567890123.45 | 1234567890123.45 |
| 1.234568E+12 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| 1.234568E+12 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| 1.234568E+12 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |
| 1.234568E+12 | 1234567890123.46 | 1234567890123.46 | 1234567890123.46 |

**1.234568E+12**

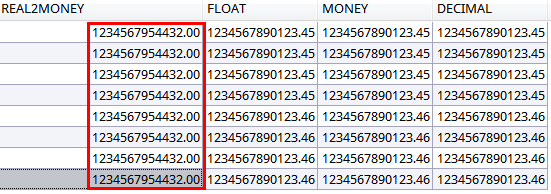
SUM을 한 경우.(오류 발생)



결국, **Float**, **Money**, **Decimal**인 경우만 정확한 값을 보여주고 있다. REAL의 경우 데이터 오류가 발생했다.

그러면 기존에 등록된 real 데이터 유형을 money로 컬럼 속송을 바꾸면 어떤 일이 생기는가?

1.234568E+12 값이 1234567954432.00로 변환되어 **잘못된 값으로 치환되었다**.



그렇다면 REAL 컬럼의 데이터를 MONEY 컬럼으로 복제하면 어떨까?



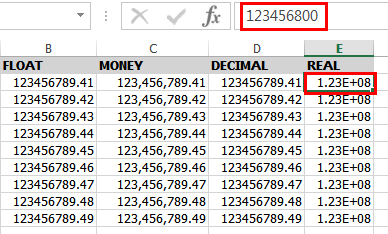
속성을 바꾼 경우와 같이 동일한 값이 들어간다.

9자리에 소수점 2자리를 가진 데이터(억)

엑셀 데이터

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **REAL** | **FLOAT** | **MONEY** | **DECIMAL** |
| 1 | 123456789.41 | 123456789.41 | 123456789.41 | 123456789.41 |
| 2 | 123456789.42 | 123456789.42 | 123456789.42 | 123456789.42 |
| 3 | 123456789.43 | 123456789.43 | 123456789.43 | 123456789.43 |
| 4 | 123456789.44 | 123456789.44 | 123456789.44 | 123456789.44 |
| 5 | 123456789.45 | 123456789.45 | 123456789.45 | 123456789.45 |
| 6 | 123456789.46 | 123456789.46 | 123456789.46 | 123456789.46 |
| 7 | 123456789.47 | 123456789.47 | 123456789.47 | 123456789.47 |
| 8 | 123456789.48 | 123456789.48 | 123456789.48 | 123456789.48 |
| 9 | 123456789.49 | 123456789.49 | 123456789.49 | 123456789.49 |
|  | 1111111105.05 | 1111111105.05 | 1111111105.05 | 1111111105.05 |

DB Import 데이터**(이상하게 표시됨)**



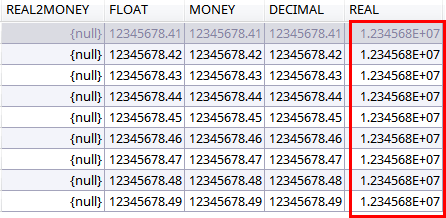
SUM하면 **값 오류 발생**



8자리에 소수점 2자리를 가진 데이터(천만)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **REAL** | **FLOAT** | **MONEY** | **DECIMAL** |
| 1 | 12345678.41 | 12345678.41 | 12345678.41 | 12345678.41 |
| 2 | 12345678.42 | 12345678.42 | 12345678.42 | 12345678.42 |
| 3 | 12345678.43 | 12345678.43 | 12345678.43 | 12345678.43 |
| 4 | 12345678.44 | 12345678.44 | 12345678.44 | 12345678.44 |
| 5 | 12345678.45 | 12345678.45 | 12345678.45 | 12345678.45 |
| 6 | 12345678.46 | 12345678.46 | 12345678.46 | 12345678.46 |
| 7 | 12345678.47 | 12345678.47 | 12345678.47 | 12345678.47 |
| 8 | 12345678.48 | 12345678.48 | 12345678.48 | 12345678.48 |
| 9 | 12345678.49 | 12345678.49 | 12345678.49 | 12345678.49 |
|  |  | 111111106.05 | 111111106.05 | 111111106.05 |

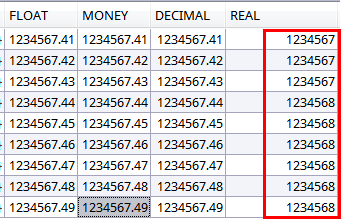
db임포트 (이상하게 표시)





7자리, 2자리 소수점(백만)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **REAL** | **FLOAT** | **MONEY** | **DECIMAL** |
| 1 | 1234567.41 | 1234567.41 | 1234567.41 | 1234567.41 |
| 2 | 1234567.42 | 1234567.42 | 1234567.42 | 1234567.42 |
| 3 | 1234567.43 | 1234567.43 | 1234567.43 | 1234567.43 |
| 4 | 1234567.44 | 1234567.44 | 1234567.44 | 1234567.44 |
| 5 | 1234567.45 | 1234567.45 | 1234567.45 | 1234567.45 |
| 6 | 1234567.46 | 1234567.46 | 1234567.46 | 1234567.46 |
| 7 | 1234567.47 | 1234567.47 | 1234567.47 | 1234567.47 |
| 8 | 1234567.48 | 1234567.48 | 1234567.48 | 1234567.48 |
| 9 | 1234567.49 | 1234567.49 | 1234567.49 | 1234567.49 |
|  |  | 11111107.05 | 11111107.05 | 11111107.05 |





8자리 이상 정수 사용하는 경우

엑셀 컬럼을 줄이면 지수형태로 표시되나, 넓히면 원래 값을 유지한다.

출처: BMS사이트

|  |
| --- |
|  |

DB에서는 지수 형태로 표시된다.(PROJECT\_LIST)

|  |
| --- |
|  |

396,000,000 (3.96E+08)

208,900,000 (2.089E+08)

SUM을 하면?

|  |
| --- |
| --SUM을 하면?  SELECT --id\_projects\_list, nm\_projects\_name, nm\_projects\_code,  SUM(real\_contract\_price) AS 'TOTAL'  FROM PROJECTS\_LIST  WHERE nm\_projects\_code = 'OR16-150221' OR nm\_projects\_code='O-16230' |

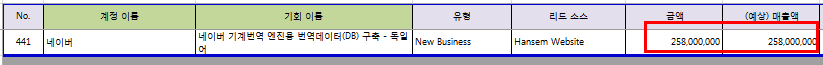
396000000+215464 = 396215464 🡺 OK

396000000+208,900,000 = 604900000 🡺 OK



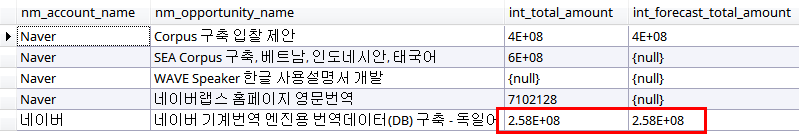
세일즈포스 사이트

엑셀에 표현된 값(정수값)

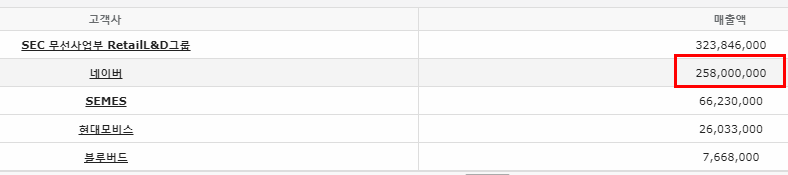


SQL서버에서 보이는 모습

정수 9자리 수의 경우, 2.xxE+08로 표기됨



웹(Sales.hansemeug.com)



초기 대응안: SUM할 때 Round 3자리에서 반올림하면 올바른 값이 나옴

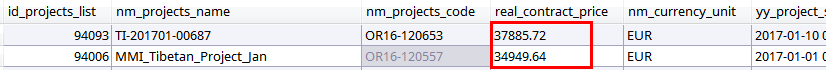
USD 달러의 경우 최고값

|  |
| --- |
|  |

EUR 달러의 경우 최고값

|  |
| --- |
|  |

SUM을 하면?



* (엑셀값) : 37885.72 + 34949.64= 7285**.36**
* (DB값) : 

시스템에서 보여지는 값은 72835.359375 이므로 따라서 소수점 3자리에서 반올림하면 됨

ROUND(값, 반올림할소수자리)

ROUND(val, 2)

|  |
| --- |
| --SUM을 하면?  SELECT --id\_projects\_list, nm\_projects\_name, nm\_projects\_code,  ROUND(SUM(real\_contract\_price),**2**) AS 'TOTAL'  FROM PROJECTS\_LIST  WHERE nm\_projects\_code = 'OR16-120653' OR nm\_projects\_code='OR16-120557' |



강제 형변환에 따른 오류 사례(BMS 운영계)

데이터 있는 상태에서 강제 형변환에 따른 값 유실 또는 변경사례

* 1. 청구서 (엑셀)



* 1. 청구서 테이블(BM\_INVOICE)

**Float** 타입

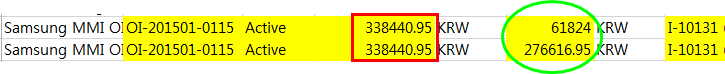


* 1. 고객 인보이스 테이블(CUSTOMER\_INVOICES\_LIST)

**Money** 타입

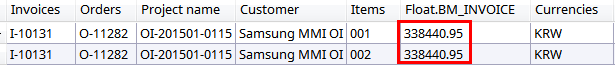


2-1. 청구 상세 내역(엑셀)



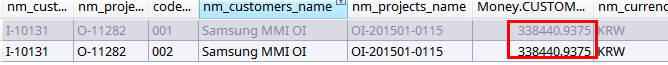
2-2. 청구서 테이블(BM\_INVOICE\_DETAIL)

**Real or Float** 타입



2-3. 고객 인보이스 상세 테이블(CUSTOMER\_INVOICES\_DETAIL

Money 타입



참고.float 및 real 데이터 사용

**float** 및 **real** 데이터 형식은 근사 데이터 형식입니다. **float** 및 **real**의 동작은 근사 숫자 데이터 형식에 대한 IEEE 754 사양을 따릅니다.

근사 숫자 데이터 형식은 여러 숫자에 지정된 정확한 값을 저장하지 않고 해당 값에 가장 가까운 근사값을 저장합니다. 대부분의 응용 프로그램에서 지정된 값과 저장된 근사값 간의 사소한 차이는 무시되지만 그렇지 않은 경우도 있습니다. **float** 및 **real** 데이터 형식에는 근사 속성이 있으므로 금융 관련 응용 프로그램과 같이 반올림 또는 등가 확인 등이 이루어지는 작업에서 정확한 숫자가 필요한 경우에는 사용하지 마십시오.

대신 integer, **decimal**, **money** 또는 **smallmoney** 데이터 형식을 사용하십시오.

특히 = 및 <> 연산자가 있는 WHERE 절 검색 조건에서는 **float** 또는 **real** 열을 사용하지 마십시오. **float** 및 **real** 열을 > 또는 < 비교로 제한하는 것이 가장 좋습니다.

IEEE 754 사양은 반올림, 올림, 내림 및 반내림의 네 가지 반올림 모드를 제공합니다. Microsoft SQL Server에서는 올림을 사용합니다. 네 가지 모두 보장된 전체 자릿수에 정확하게 맞지만 부동 소수점 값에 있어서는 결과가 약간 다를 수 있습니다. 부동 소수점의 이진 표시에는 여러 가지 합법적인 반올림 방법 중 하나가 사용될 수 있으므로 부동 소수점 값을 확실하게 수량화할 수는 없습니다.

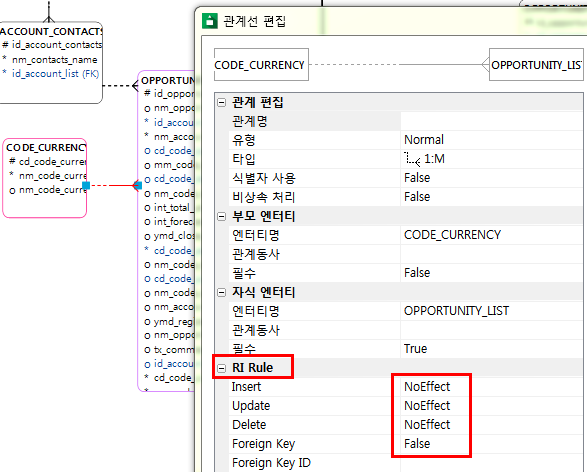
The behavior of float and real follows the IEEE 754 specification on approximate numeric data types. Because of the approximate nature of the float and real data types, do not use these data types when exact numeric behavior is required, such as in financial applications, in operations involving rounding, or in equality checks. Instead, use the integer, decimal, money, or smallmoney data types. Avoid using float or real columns in WHERE clause search conditions, especially the = and <> operators. **It is best to limit float and real columns to > or < comparisons.**

RI Rule: 무결성 규칙

무결성은 데이터의 정확성, 유효성, 일관성, 신뢰성을 위해 무효 갱신으로부터 데이터를 보호하기 위함입니다.

즉, 데이터가 테이블에 입력, 삭제, 수정될 때 자식 또는 부모 테이블의 처리에 대한 규칙을 정의하는 것입니다. 물리적 스키마의 생성은 삭제 규칙에만 스키마를 지원합니다.

**물리 모델에서** 테이블의 **관계선을 클릭하여** RI 룰을 지정한다.



|  |  |  |  |
| --- | --- | --- | --- |
| 등록(INSERT) | | 수정/삭제(Update/Delte) | |
| **Dependent** | 대응되는 부모 엔터티에 레코드가 있는 경우에만 자식 엔터티에 입력을 허용 | **Restrict** | 대응되는 자식 엔터티에 레코드가 없는 경우에만 부모 엔터티에 수정 및 삭제를 허용 |
| **Automatic** | 대응되는 부모 건이 없는 경우 이를 자동 생성 | **Cascade** | 부모 엔터티 레코드의 삭제/수정을 항상 허용, 대응되는 자식 엔터티의 레코드를 자동 수정 및 삭제 |
|  |  | **Null** | 대응되는 부모 엔터티의 레코드가 존재하면, 그것의 참조키(FK)를 NULL로 수정 및 삭제 |
| **Default** | 대응되는 부모 건이 없는 경우 참조키(FK)를 지정된 기본 값으로 처리 | **Default** | 대응되는 자식 엔터티의 레코드가 존재하면, 그것의 참조키(FK)를 지정된 기본 값으로 수정 및 삭제 |
| **No Effect** | 자식 엔터티 레코드의 입력을 조건 없이 허용 | **No Effect** | 부모 엔터티 레코드의 입력을 조건 없이 허용 |
| **Customized** | 특정한 검증 조건이 만족되는 경우에만 자식 엔터티 레코드의 입력을 허용 | **Customized** | 특정한 검증 조건이 만족되는 경우에만 부모 엔터티 레코드의 삭제/수정을 허용 |

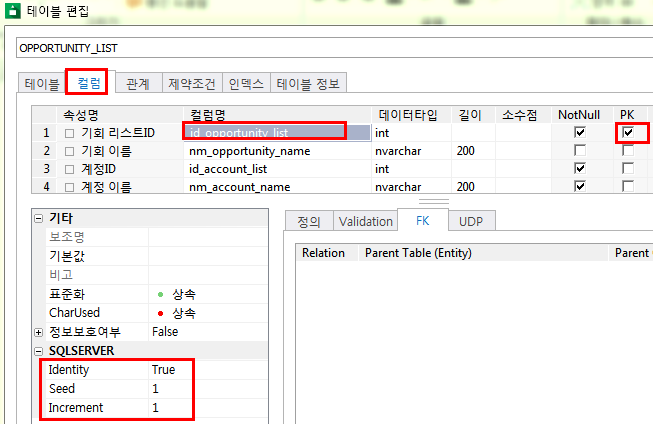
CONSTRAINTS: 제약조건

컬럼 데이터 값의 제약조건을 지정할 있습니다. PK, FK, CEHCK, INDENTITY 등의 제약조건이 있습니다.

반드시 테이블에서 제약조건을 확인합니다.

Identity, 식별키, 외래키, 체크 등의 제한조건은 자동으로 스크립트 생성이 되지 않을 수 있다. 해당 **물리 모델링에서** 체크해줘야 한다. 경우에 따라서 종종 테이블에 구현이 안되어 있을 수 있으므로 명령어로 만들어 주어야 한다.

PK & IDENTITY(자동 증가)



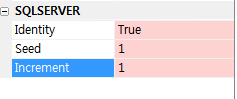
**물리모델** 화면에서 컬럼 속성에서 Indentity, Seed, Increment 지정한다.

**DA#5**

Identity = true,

Seed = 1 (초기값),

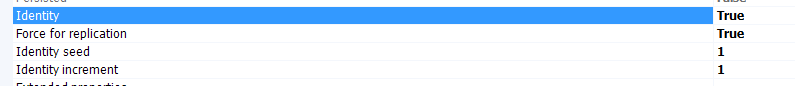
Increment = 1 (증가값)



**Toad for SQL Server Tool에서 설정하기**

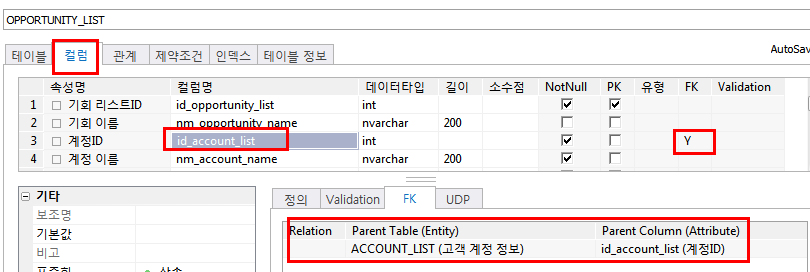
Alter Table > 해당 컬럼 선택 > Identity = True > Seed와 Increment 지정

Identity (1,1) 1부터 시작하여 1씩 증가

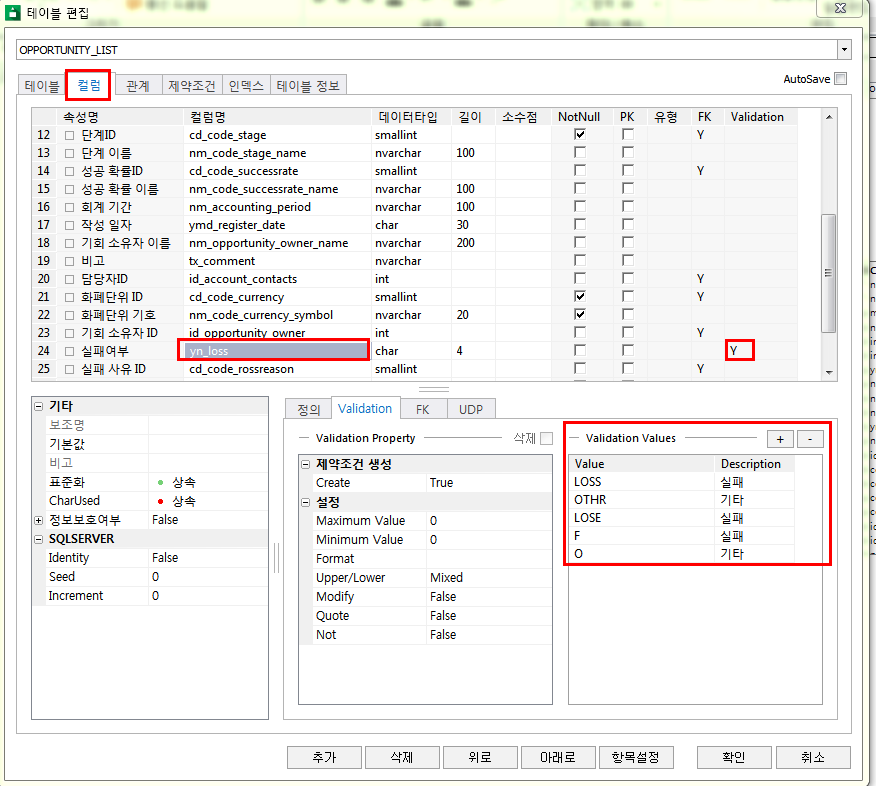


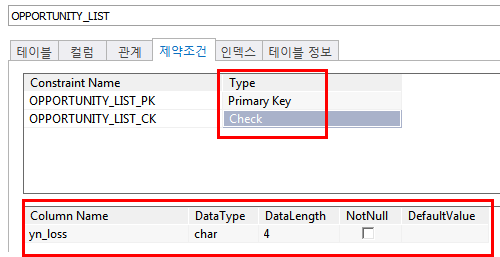


FK



CHECK





해당 컬럼(yn\_loss) 편집화면에서 값의 유효성을 지정할 수 있습니다,

INDEX 생성

|  |
| --- |
| CREATE UNIQUE **INDEX** [CUSTOMER\_INVOICES\_LIST\_PK] ON [CUSTOMER\_INVOICES\_LIST]  ( [pk\_customer\_invoices\_list] )  GO  DROP **INDEX** CUSTOMER\_INVOICES\_DETAIL\_PK  GO |

SP: 저장 프로시져

뷰는 정적인 결과를 처리하지만, SP는 동적으로 값을 전달받아 처리한다.

프로시져를 작성하려면 DECLARE 변수 선언을 DECLARE없이 파라미터에 기술하면 된다.

에 회계기간 정보를 **파라미터로 전달하여 결과를 뽑으려면** Stored Procedure를 생성해야 한다.

|  |
| --- |
| CREATE **VIEW** v\_totalAmount  AS  SELECT SUM( int\_total\_amount) Total\_QuatorAmount FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period = **'Q4-2017'** AND yn\_loss = 'OTHR';  SELECT \* FROM v\_totalAmount |

SP 함수로 만들어 사용하기

|  |
| --- |
| -- 회계분기별 매출 총계  --@nmAccountingQuater = 'Q4-2017'  CREATE PROCEDURE sp\_totalAmount(@nmAccountingQuater **nvarchar**(**20**))  AS  SELECT SUM( int\_total\_amount) Total\_QuatorAmount FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period = @nmAccountingQuater AND yn\_loss = 'OTHR';  GO |

|  |
| --- |
| -- SP실행  [sp명] [괄호없이파라미터1] , [괄호업는파라미터2]  sp\_totalAmount 'Q4-2017' |

연간 누적 매출 총계 SQL문

|  |
| --- |
| -- 연간 누적 매출 총계  -- TotalAmountClosedCurrent  --@nmAccountYear = '%2017'  DECLARE @nmAccountYear **nvarchar**(**20**);  SET @nmAccountYear = '%2017'; --201  SELECT SUM( int\_total\_amount) Total\_OppAmount FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period LIKE @nmAccountYear AND cd\_code\_stage=  ( SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name = N'수주확정' ); |

SP 생성하여 임의 파라미터로 호출하면 바로 결과값 얻을 수 있다

|  |
| --- |
| CREATE **PROCEDURE** sp\_totalAmountClosedCurrent(@nmAccountYear **nvarchar**(**20**))  AS SELECT SUM( int\_total\_amount) Total\_OppAmount FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period LIKE @nmAccountYear AND cd\_code\_stage=  ( SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name = N'수주확정' );  sp\_totalAmountClosedCurrent '%2017'    sp\_totalAmountClosedCurrent '%2018’ |

단계별 파이프라인 누적 총계와 전체 총액

|  |
| --- |
| -- 단계별 파이프라인 누적  -- GROUP BY STAGE  CREATE **PROCEDURE** sp\_totalAmountByStage(@nmAccountYear **nvarchar**(**20**))  AS  SELECT sum( int\_total\_amount), nm\_code\_stage\_name FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period LIKE @nmAccountYear  GROUP BY nm\_code\_stage\_name ORDER BY nm\_code\_stage\_name;  -- 연간 누적금액  SELECT sum( int\_total\_amount) FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period LIKE @nmAccountYear |

|  |
| --- |
| sp\_totalAmountByStage '%2018' |

2개 이상의 파라미터 전달

그런데 **TOP N**에 숫자를 넣으려면 괄호( ‘( )’)를 사용하여 묶어 주어야 한다

|  |
| --- |
| -- TOP 고객 매출 현황  -- Top Account Won  -- @nmAccountYear nvarchar(20) = '%2017'  -- @intTopRow smallint = 20/10/5  --SELECT TOP 20 nm\_account\_name, sum(int\_total\_amount) FROM OPPORTUNITY\_LIST  --WHERE nm\_accounting\_period LIKE '%2017'  --GROUP BY nm\_account\_name  --ORDER BY sum(int\_total\_amount) DESC  -- TOP 고객 매출 현황  CREATE **PROCEDURE** **sp\_topNAccountByWon** (@intTopRow **smallint**, @nmAccountYear **nvarchar**(**20**))  AS  SELECT TOP **(**@intTopRow**)** nm\_account\_name, sum(int\_total\_amount) FROM OPPORTUNITY\_LIST  WHERE nm\_accounting\_period LIKE @nmAccountYear  GROUP BY nm\_account\_name  ORDER BY sum(int\_total\_amount) DESC |

|  |
| --- |
| **sp\_topNAccountByWon** **10**, '%2017' |

VIEW 생성

각 사용자가 데이터베이스를 보는 시각에 초점을 맞추고 데이터 조작을 간소화하며 사용자 지정할 수 있습니다.

뷰는 원본이 되는 기본 테이블에 직접 액세스할 수 있는 권한을 부여하지 않고 뷰를 통해 데이터에 액세스하도록 하기 때문에 보안 메커니즘으로 사용할 수 있습니다.

VIEW 사용

|  |
| --- |
| -- 마치 테이블처럼 SELECT로 불러서 사용한다.  SELECT \* FROM {**VIEW**명}; |
| -- 3분기 2017년도 매출 집계  CREATE **VIEW** v\_totalAmount  AS  SELECT SUM( int\_total\_amount) Total\_QuatorAmount FROM OPPORTUNITY\_LIST WHERE nm\_accounting\_period = 'Q4-2017' AND yn\_loss = 'OTHR';  SELECT \* FROM v\_totalAmount |

가져올 컬럼 지정

* **SELECT FROM** {가져올 테이블 목록}
* 테이블 목록에서 { 테이블명.컬럼명 } 선택
* WHERE 조건문
* GROUP BY / ORDER BY

CREATE VIEW

|  |
| --- |
| **CREATE** OR REPLACE **VIEW** {뷰 이름}  AS  {위에서 생성한 SELECT FROM 문장} |

|  |
| --- |
| **CREATE VIEW** hiredate\_view  **AS**  SELECT p.FirstName, p.LastName, e.BusinessEntityID, e.HireDate  FROM **HumanResources**.**Employee** e  JOIN **Person**.**Person** AS p ON e.BusinessEntityID = p.BusinessEntityID ;  GO |

두 테이블 조인하는 뷰

|  |
| --- |
| CREATE VIEW view1 AS  SELECT fis.CustomerKey, fis.ProductKey, fis.OrderDateKey,  fis.SalesTerritoryKey, dst.SalesTerritoryRegion  FROM FactInternetSales AS fis  LEFT JOIN DimSalesTerritory AS dst  ON (fis.SalesTerritoryKey=dst.SalesTerritoryKey); |

기본 제공 함수를 사용할 때는 파생 열의 **열 이름을** 지정해야 합니다.

|  |
| --- |
| CREATE VIEW Sales.SalesPersonPerform  AS  SELECT TOP (100) SalesPersonID, SUM(TotalDue) AS TotalSales  FROM Sales.SalesOrderHeader  WHERE OrderDate > CONVERT(DATETIME,'20001231',101)  GROUP BY SalesPersonID;  GO |

WITH SCHEMABINDING 사용하여 분할된 데이터 (Partitioned Data)

서로 다른 국가/지역에 있는 사무실의 테이블 내역입니다.

|  |
| --- |
| --Create the view that combines all supplier tables.  CREATE VIEW dbo.all\_supplier\_view  **WITH SCHEMABINDING**  AS  SELECT supplyID, supplier  FROM dbo.SUPPLY1  UNION ALL  SELECT supplyID, supplier  FROM dbo.SUPPLY2  UNION ALL  SELECT supplyID, supplier  FROM dbo.SUPPLY3  UNION ALL  SELECT supplyID, supplier  FROM dbo.SUPPLY4; |

WITH CHECK OPTION 특정사용자에게만 보여주는 뷰

|  |
| --- |
| CREATE VIEW dbo.SeattleOnly AS  SELECT p.LastName, p.FirstName, e.JobTitle, a.City, sp.StateProvinceCode  FROM HumanResources.Employee e  INNER JOIN Person.Person p  ON p.BusinessEntityID = e.BusinessEntityID  INNER JOIN Person.BusinessEntityAddress bea  ON bea.BusinessEntityID = e.BusinessEntityID  INNER JOIN Person.Address a  ON a.AddressID = bea.AddressID  INNER JOIN Person.StateProvince sp  ON sp.StateProvinceID = a.StateProvinceID  WHERE a.City = 'Seattle'  **WITH CHECK OPTION** ;  GO |

WITH ENCRYPTION 사용하여 계산 열, 이름이 바뀐 열 및 복수 열을 보여준다.

|  |
| --- |
| CREATE VIEW Purchasing.PurchaseOrderReject  WITH ENCRYPTION  AS  SELECT PurchaseOrderID, ReceivedQty, RejectedQty,  RejectedQty / ReceivedQty AS RejectRatio, DueDate  FROM Purchasing.PurchaseOrderDetail  WHERE RejectedQty / ReceivedQty > 0  AND DueDate > CONVERT(DATETIME,'20010630',101) ;  GO |

VIEW에 파라미터를 전달 할 수 있는가? SP사용

뷰에서는 파라미터 전달을 할 수 없다.. 사용자 정의 함수(User defined Function)을 만들거나 저장 프로시져(Stored Procedure)를 사용해야 한다.

즉, 사원ID를 받아서 처리하는 뷰를 생성하더라도 작동하지 않는다.

|  |
| --- |
| **create** or replace **view** v\_emp(eno number) **as** select \* from emp where emp\_id=&eno; |

대신에 SP나 Function을 사용해서 구현해야 한다.

|  |
| --- |
| **CREATE** **PROCEDURE** s\_emp ( @enoNumber INT )  **AS** SELECT \* FROM emp WHERE emp\_id=@enoNumber;  -- Excute Stored Procedures  -- sp\_name para1{, para2}  s\_emp 1234; |

|  |
| --- |
| CREATE **FUNCTION** u\_emp ( @enoNumber INT )  **RETURNS TABLE**  **AS**  **RETURN**  **(** SELECT \* FROM emp WHERE emp\_id=@enoNumber  **)** |

ERD SW: DA# 5

구조

프로젝트 🡪 모델 🡪 주제영역 (논리 다이어그램, 물리 다이어그램)

“왜 모델폴더를 만들어 사용하는 지 알 수 없다”

주제영역은 서로 관계성이 높은 데이터들을 묶어 하나의 응집하게 하는데 목적이 있습니다. 서로 다른 주제영역간에는 최소한의 낮은 결합도를 유지해야 합니다.

주제영역의 좋은 예는 ‘AS-IS’ ‘TO-BE’처럼 나눌 때 사용한다.

##SQL Programming

기본 SQL 쿼리

SELECT

SELECT {항목 리스트} > FROM {테이블}

SELECT {항목 리스트}

**> FROM** {테이블}

> **WHERE** {검색조건}

> **ORDER** BY {ASC|DESC}

> **GROUP** BY > **ORDER** BY

> **GROUP** BY > **HAVING** > ORDER BY

SELECT > FROM {테이블}

> INNER JOIN | LETF JOIN | RIGHT JOIN {테이블}

> ON {두 테이블간 연결조건}

{> INNER JOIN | LETF JOIN | RIGHT JOIN {테이블} }

> ON {두 테이블간 연결조건}

> WHERE {조건 검색}

TOP {개수}

검색 결과를 지정된 개수만큼 가져오게 한다. 출력 순서대로 지정 개수를 출력한다.

SELECT 문 다음에 나타난다.

SELECT TOP 1 id\_emp\_name FROM {Table\_name}

변수 사용시에는 괄호로 묶어야 한다.

SELECT TOP (@noRowCounts**)** id\_emp\_name FROM {Table\_name}

DISTINCT (중복제거 구별값)

|  |
| --- |
| SELECT DISTINCT last\_name, first\_name  FROM employees; |

Employees 테이블에서 last\_name과 first\_name 조합의 결과물에서 유일한 결과물을 반환한다.

DISCTINCT 키워드 뒤에 나오는 모든 필드에 중복제거를 적용한다.

오직 SELECT 구문에서만 사용되는 키워드이다.

EXISTS / NOT EXISTS

SELECT 조회 건수가 있느냐 없느냐를 판단한다.

보통 IF 와 SELECT 구문과 함께 사용된다.

IF EXISTS()

IF NOT EXISTS()

EXCEPT & INTERSECT operator

EXCEPT는 오라클의 MINUS와 동일한 기능을 하는 연산자이다.

좌측 집합에서 우측 집합을 제외한 나머지 집합만을 출력한다. 그 값은 중복되지 않은 구별되는 값(Distinct)이다. 단, 두 질의 구문은 개수가 동일하고 데이터 유형이 같아야 한다.

|  |
| --- |
| -----------------------------------------------  -- 로그인 계정 없는 사원 찾기  -----------------------------------------------  -- 재직중인 전체 사원 집합  SELECT e.nm\_emp\_name, e.yn\_retired  FROM TBL\_COMPANY\_EMP e  WHERE e.yn\_retired = 'N'  EXCEPT  -- 로그인 계정이 있는 재직 사원 집합  SELECT e.nm\_emp\_name, e.yn\_retired  FROM TBL\_COMPANY\_EMP e  INNER JOIN TBL\_LOGIN\_AUTH a  ON e.id\_emp = a.int\_login\_user\_id AND e.yn\_retired = 'N'  ORDER BY e.nm\_emp\_name |

|  |
| --- |
| SELECT supplier\_id, supplier\_name  FROM suppliers  WHERE supplier\_id > 2000  **MINUS**  SELECT company\_id, company\_name  FROM companies  WHERE company\_id > 1000  ORDER BY 2; |

INTERSECT는 좌측 집합과 우측 집합의 결과를 모두 출력한다. 값은 중복되지 않은 값(Distinct)이다.

단, 두 질의 구문은 개수가 동일하고 데이터 유형이 같아야 한다.

|  |
| --- |
| -- 재직중인 전체 사원 집합  SELECT e.nm\_emp\_name, e.yn\_retired  FROM TBL\_COMPANY\_EMP e  WHERE e.yn\_retired = 'N'  EXCEPT  -- 로그인 계정이 있는 재직 사원 집합  SELECT e.nm\_emp\_name, e.yn\_retired  FROM TBL\_COMPANY\_EMP e  INNER JOIN TBL\_LOGIN\_AUTH a  ON e.id\_emp = a.int\_login\_user\_id AND e.yn\_retired = 'N'  ORDER BY e.nm\_emp\_name |

WHERE

<http://www.gurubee.net/lecture/1017>

비교 연산자

=, >, <,

>=, <=,

<>

AND, OR

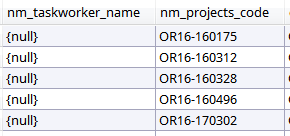
NOT

NOT검색은 <>와 동일한 결과이다.

IS NULL / NOT NULL 연산자

데이터가 아무것도 들어 있지 않은 데이터를 찾을 때 사용한다.

NULL과 ‘’는 다르다. 동일하게 작용하지 않는 경우가 있으므로 유의해야 한다.



SELECT nm\_taskworker\_name

FROM BM\_JOBS

WHERE nm\_taskworker\_name IS NULL

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the **IS NULL and IS NOT NULL** operators instead.

OR와 IN 비교

IN은 하나의 컬럼이 여러 개의 = 조건을 가지는 경우에만 가능하나

OR은 어떠한 논리합 관계도 표현 가능하다.

|  |  |
| --- | --- |
| IN | OR |
| SELECT \*  FROM TAB1 WHERE COL  IN ( '1','5','7')  AND COL2 = '111'; | SELECT \* FROM TAB1 WHERE  (COL1 ='1' OR COL1 ='5' OR COL1 ='7') AND COL2 = '111'; |
| SELECT \* FROM TAB1 WHERE ( COL1 ='1' OR COL2 = '500' OR COL2LIKE '7%' )  AND COL2 = '111'; | |

<http://wiki.gurubee.net/pages/viewpage.action?pageId=2458059>

IN(list) / NOT IN(list)

IN 목록에 제시된 값을 가진 레코드를 가져올 때 사용한다. 그리고 제시된 목록을 포함하지 않은 레코드를 가져올 때는 NOT IN을 사용한다.

strNmae **IN** (‘선배’, ‘후배’ )

strNmae **NOT IN** (‘선배’, ‘후배’)

|  |
| --- |
| --특정IP 주소가 허용되는 IP주소에 들어있는 지 검사하는 모듈  -- 단일 IP주소로 등록된 경우와  -- 범위 IP주소로 등록 관리할 수 있다.  -- 클라이언트 IP주소  DECLARE @ip **nvarchar**(**100**) ;  DECLARE @cntIP **int** ;  SET @ip = '61.83.152.148';  -- SUB='R'인 범위대역 IP주소 갯수만큼 '주소 집합'을 추가하여 다이나믹 쿼리를 생성한다.  --DECLARE @cntIPRange int ;  --SET @cntIPRange = (SELECT COUNT(\*) FROM NETWORK\_ALLOW\_IP WHERE sub\_ipaddress\_range ='R' ) ;  -- WHILE @cntIPRange >0  -- BEGIN  -- SET @cntIPRange = @cntIPRange -1;  -- END  -- 두개의 범위대역 IP주소가 등록되었다고 가정한 시뮬레이션  -- 다수의 대역폭 주소가 있다면 갯수만큼 집합 추가할 것  DECLARE @ip\_start\_001 **nvarchar**(**100**), @ip\_end\_001 **nvarchar**(**100**)  DECLARE @ip\_start\_002 **nvarchar**(**100**), @ip\_end\_002 **nvarchar**(**100**)  SET @ip\_start\_001 = '61.83.152.160'; -- 첫번째 nm\_ip\_address\_startrange FROM NETWORK\_ALLOW\_IP WHERE sub\_ipaddress\_range ='R'  SET @ip\_end\_001 = '61.83.152.250'; -- nm\_ip\_address\_finishrange  SET @ip\_start\_002 = '61.83.152.50'; -- 두번째 nm\_ip\_address\_startrange FROM NETWORK\_ALLOW\_IP WHERE sub\_ipaddress\_range ='R'  SET @ip\_end\_002 = '61.83.152.100'; -- nm\_ip\_address\_finishrange  -- 범위주소와 싱글주소에 들어있는 지 체크하는 로직  SET @cntIP = (  SELECT COUNT(@ip) FROM NETWORK\_ALLOW\_IP  WHERE @ip **IN** (  ( SELECT @ip FROM NETWORK\_ALLOW\_IP  WHERE @ip BETWEEN @ip\_start\_001 AND @ip\_end\_001  OR @ip BETWEEN @ip\_start\_002 AND @ip\_end\_002  )  )  OR nm\_ip\_address\_single=@ip  );    IF (@cntIP >**0** ) print 'Accept :' + @ip; -- 허가된 IP주소  else print 'Reject:'+ @ip; -- 미인가된 IP주소 |

BETWEEN a AND b

BETWEEN A AND B 처럼 값의 범위가 A부터 B까지의 모든 **값**을 검색한다. **문자열**이나 **숫자**도 가능함.

하지만 LIKE처럼 문자열 일부 포함하는 와일드 검색은 지원하지 않음.

해당 문장은 >= A AND <= B로 변경하여 사용할 수 있다.

|  |
| --- |
| WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'  Price BETWEEN 10 AND 20 |

LIKE 와일드검색

검색할 스트링에 대해 와일드카드 검색을 위해 LIKE 연산자를 사용한다.

% : **여러 개의 문자열**을 나타내는 와일드카드

**\_** (underline): **단 하나의 문자**를 나타내는 와일드 카드

\(escape) : 와일드 카드 문자를 문자로 사용하려고 할 때

예)

LIKE ‘A%’, LIKE ‘%A’, LIKE ‘%KIM%’, LIKE ‘%K%I%’,   
LIKE ‘\_A%’ : 두번째 문자가 A로 시작하는 문자열

GROUP BY

GROUP BY를 사용하면, 거기에 포함된 컬럼만 SELECT절에 들어와야 한다. 만약, 다른 항목을 넣으려면 반드시 집계함수를 써야 한다.

|  |
| --- |
| SELECT title\_id  , **sum**(qty) as 'sum of qty'  FROM Sales  GROUP BY title\_id  order by title\_id |

HAVING

Group by 절에서 나온 결과를 특정 조건으로 필터링 할 때 사용합니다.

따라서 HAVING 구문은 SELECT의 WHERE와 같이 GROUP BY에서 사용하는 조건문입니다.

|  |
| --- |
| -- 그룹핑한 합산 결과가 30이상인 경우만 가져오게 하려면  SELECT title\_id  , sum(qty) as 'sum of qty'  FROM Sales  GROUP BY title\_id HAVING sum(qty) > 30 |

|  |
| --- |
| -- 수량 자체가 10미만인 경우는 합산에 포함하지 않으려면 WHERE에서 제한하고..  SELECT title\_id  , sum(qty) as 'sum of qty'  FROM Sales  WHERE qty > 10 -- 10개 이상인 자료만 가져와서  GROUP BY title\_id -- 목별로 그룹핑하여  HAVING sum(qty) > 30 -- 합산이 30을 초과한 데이타만 가져온다 |

Sub 쿼리 구문

하나의 쿼리 안에 또 다른 쿼리가 들어가 있는 쿼리문을 말한다.

보통 SELECT 절, FROM 절, WHERE절에서 활용되어 진다.

SELECT 절 서브쿼리

SELECT 구문에서 서브쿼리는 하나의 단일값만을 리턴해야 한다. 보통 SUM, MIN, MAX, COUNT와같은 aggregate 함수(집계함수)는 주로 서브쿼리에서 사용되기 때문이다.

|  |
| --- |
| SELECT e1.last\_name, e1.first\_name,  ( SELECT MAX(salary)  FROM employees e2  WHERE e1.employee\_id = e2.employee\_id  ) max\_salary  FROM employees e1; |

|  |
| --- |
| --country정보를 여러개 가진 데이타 찾아내기  SELECT pname, count(country) as cnt\_country FROM (  SELECT DISTINCT a.nm\_name\_venorperson AS pname ,b.nm\_country AS country  FROM RESOURCE\_LOCALE\_LIST b  inner JOIN TBL\_RESOURCE\_LIST a  ON a.id\_tbl\_resource\_list = b.id\_tbl\_resource\_list  ) subquery1  GROUP BY pname  HAVING count(country) > **1**  order by cnt\_country desc |

FROM 절 서브쿼리

|  |
| --- |
| SELECT suppliers.supplier\_name,  **subquery1**.total\_amt  FROM **suppliers**,  ( SELECT supplier\_id, SUM(orders.amount) AS total\_amt  FROM orders  GROUP BY supplier\_id  ) **subquery1**  WHERE **subquery1**.supplier\_id = **suppliers**.supplier\_id; |

|  |
| --- |
| --country정보를 여러개 가진 데이타 찾아내기  SELECT res.nm\_name\_venorperson as rsource\_name,  count(loc.nm\_country ) AS count\_country  FROM TBL\_RESOURCE\_LIST res,  (  SELECT DISTINCT id\_tbl\_resource\_list, nm\_country  FROM RESOURCE\_LOCALE\_LIST  GROUP BY id\_tbl\_resource\_list, nm\_country  ) loc  WHERE res.id\_tbl\_resource\_list = loc.id\_tbl\_resource\_list  GROUP BY res.nm\_name\_venorperson  HAVING count(loc.nm\_country ) > **1**  ORDER BY count\_country desc |

WHERE 절 서브쿼리

|  |
| --- |
| SELECT p.product\_id, p.product\_name  FROM products p  WHERE p.product\_id IN  (  SELECT inv.product\_id  FROM inventory inv  WHERE inv.quantity > 10  ); |

보통 IINER JOIN으로 재작성된다.

|  |
| --- |
| SELECT p.product\_id, p.product\_name  FROM products p  INNER JOIN inventory inv  ON p.product\_id = inv.product\_id  WHERE inv.quantity > 10; |

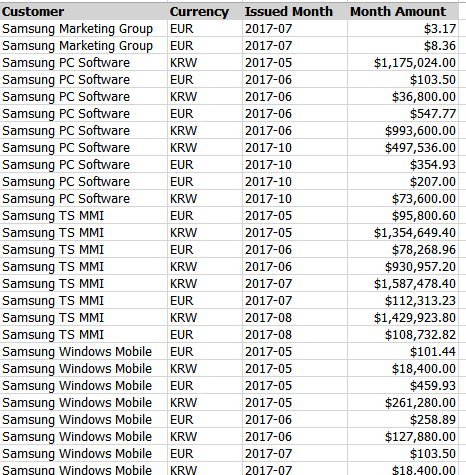
집계 쿼리 구문

GR**O**UP BY

데이터는 한 행씩db테이블에 들어가 있다.

고객명, 통화단위, 발행월, 금액을 조회합니다.

|  |
| --- |
| -- 고객별 인보이스 집계 현황: GROUP BY  SELECT nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , ym\_invoice\_issued\_month AS 'Issued Month'  , real\_invoice\_amount) AS 'Month Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %' |



여기서 **고객별, 월별** 합계를 뽑고자 한다면 어떻게 해야 하나?

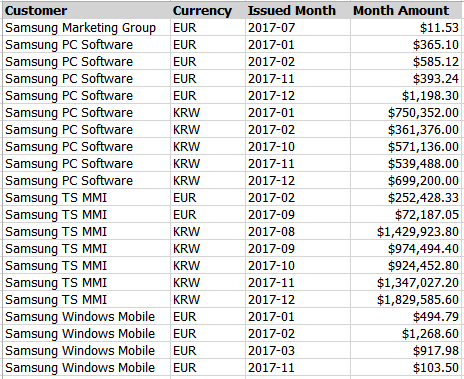
단순하게 금액을 더하는 SUM함수를 사용하면 될 것 같습니다. 아래와 같이 SUM을 사용하여 SQL를 실행시키면, 에러가 발생합니다. “집계함수를 사용하지 않거나, Group by 절에 SELECT 목록에 나온 항목대로 나열하라”고 에러 메시지가 나옵니다.

|  |
| --- |
| SELECT nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , ym\_invoice\_issued\_month AS 'Issued Month'  , SUM( real\_invoice\_amount) AS 'Month Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %' |

즉, 값을 집계하려면 SELECT 구문에 ‘고객’, ‘화폐’, ‘월’ 등의 다른 항목을 보여주려면 GROUP BY 절로 명시해야 합니다.

|  |
| --- |
| SELECT nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , ym\_invoice\_issued\_month AS 'Issued Month'  , SUM( real\_invoice\_amount) AS 'Month Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %'  GROUP BY nm\_customers\_ name,nm\_currency\_unit, ym\_invoice\_issued\_month |

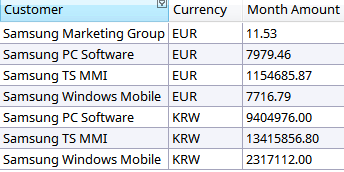
[고객명+화폐단위+월] 기준으로 그룹핑하여 금액을 더해 봅니다. (월 집계)



‘고객+화폐’ 기준으로 집계를 하면 ? 여기에 ‘고객’, ‘화폐’ 집계를 내고자 한다면 ?

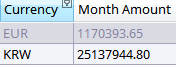
[Group by 고객, 화폐]와 [Group by 고객], [Group by 화폐]를 하면 될 것이다.

|  |
| --- |
| SELECT nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , SUM( real\_invoice\_amount) AS 'Month Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %'  GROUP BY nm\_customers\_name,nm\_currency\_unit |



화폐단위로 집계를 하면?

|  |
| --- |
| SELECT nm\_currency\_unit AS Currency  , SUM( real\_invoice\_amount) AS 'Month Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %'  GROUP BY nm\_currency\_unit |



단, Group By는 전체 합계 금액을 계산해 주지 않는다.

ROLLUP (item,,,)

Group by 는 정해진 그룹 항목들을 기준으로 집계만 해 줍니다. 그 상위 그룹별로 중간 소계나 전체 합계를 제공하지 않습니다.

그러나 우리가 그룹 항목들을 한 단계씩 올라 가면서 집계를 원한다면 어떻게 해야 할 까?

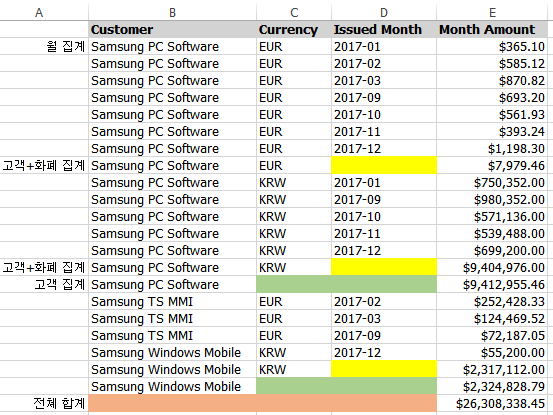
고객+화폐+월의 월별 집계에서, 고객+화폐 단위 집계, 고객 집계, 그리고 전체 집계를 뽑아야 한다. 그룹 항목을 하나씩 말아 올라가면서 집계를 내려면 **Group by ROLLUP( 항목 리스트)** 구문(또는 WITH ROLLUP**)**을 활용하면 된다.

즉, Group by가 기본적으로 제공하는 ‘고객사+화폐+월’ 집계에 ‘**고객+화폐’** 집계와 ‘**고객’** 집계 그리고 **전체 집계**에 대한 3레벨를 추가로 제공할 수 있다..

GROUP BY ROLLUP( 항목1, 항목2)은 GROUP BY 항목1, 항목2 WITH ROLLUP과 동일하다.

|  |
| --- |
| SELECT nm\_customers\_name,nm\_currency\_unit , ym\_invoice\_issued\_month, SUM(amount)  GROUP BY ROLLUP(nm\_customers\_name,nm\_currency\_unit , ym\_invoice\_issued\_month) |

* GROUP BY nm\_customers\_name, nm\_currency\_unit, ym\_invoice\_issued\_month **WITH** ROLLUP



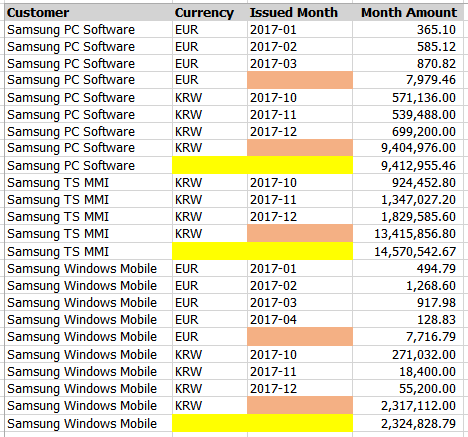
분할 ROLLUP(item,,,)

분할 ROLLUP은 ROLLUP과 같은데, 전체 집계를 하지 않는다.

예를 들어 GROUP BY 항목1, ROLLUP(항목2, 항목3)으로 집계를 하면 결과는 {항목1, 항목2, 항목3}, {항목1, 항목2}, {항목1} 별로 집계가 되고 전체 합계는 없다.

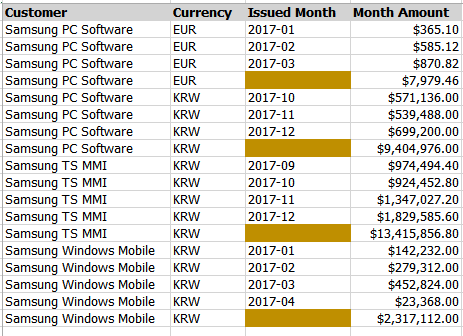
|  |
| --- |
| --분할 ROLLUP: 고객, (화폐, 월)  GROUP BY nm\_customers\_name , ROLLUP (nm\_currency\_unit, ym\_invoice\_issued\_month) |

{고객+화폐+월} , {고객+화폐}, {고객}



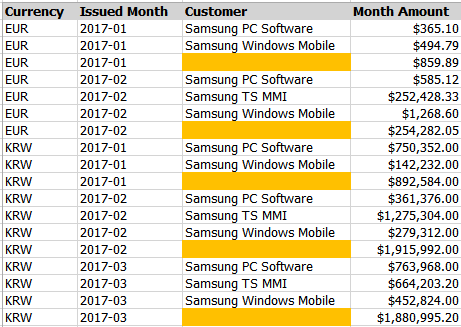
|  |
| --- |
| --분할 ROLLUP: 고객, 화폐, (월)  GROUP BY nm\_customers\_name , nm\_currency\_unit, ROLLUP( ym\_invoice\_issued\_month) |

{고객+화폐+월} , {고객+화폐}



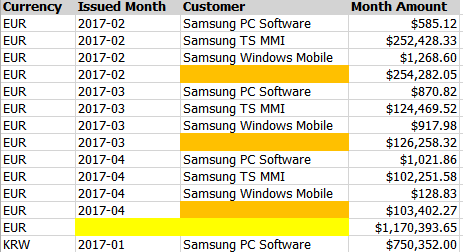
|  |
| --- |
| SELECT  nm\_currency\_unit AS Currency  , ym\_invoice\_issued\_month AS 'Issued Month'  ,nm\_customers\_name AS Customer  , SUM( real\_invoice\_amount) AS 'Month Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %'  --분할 ROLLUP: 화폐, 월, (고객) --> 월단위 화폐 금액집계  GROUP BY nm\_currency\_unit, ym\_invoice\_issued\_month, ROLLUP(nm\_customers\_name) |

{화폐+월+고객}, {화폐+월}



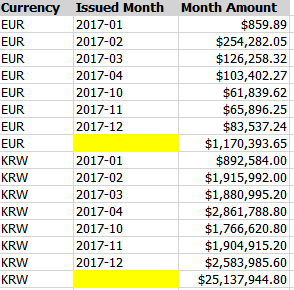
|  |
| --- |
| --분할 ROLLUP: 화폐, 월, (고객) --> 화폐 집계  GROUP BY nm\_currency\_unit, ROLLUP( ym\_invoice\_issued\_month, nm\_customers\_name) |

{화폐+월+고객}, {화폐+월}, {화폐}



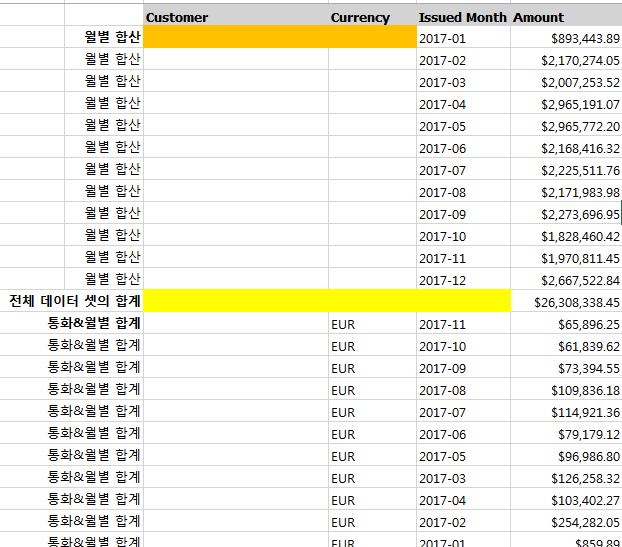
|  |
| --- |
| --분할 ROLLUP: 화폐, (월) --> 화폐 집계  GROUP BY nm\_currency\_unit, ROLLUP( ym\_invoice\_issued\_month) |

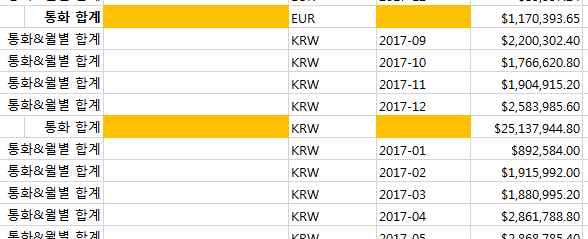
{화폐+월 }, {화폐}



CUBE

그룹핑한 항목을 모두 포함하여 집계를 내게 하는 방법도 있다. ROLLUP은 한단계씩 올라갔지만, CUBE를 사용하면 모든 항목에 대한 집계를 해준다.





GROUPING SETS(item,,,)

|  |
| --- |
| WHERE ym\_invoice\_issued\_month like '2018%' -- 연간 집계  GROUP BY GROUPING SETS(nm\_currency\_unit,nm\_customers\_name, ym\_invoice\_issued\_month ) |

****

Over (Partition by)

SQL 집계함수 **SUM**, **COUNT**, **MAX**, **MIN**과 **AVG**는 **OVER** 절을 사용할 수 있다.

Group by를 사용하면 Group by ‘항목’만이 오직 SELECT 절에 와야 한다.

하지만, OVER절을 사용하면 SELECT 절에 Group by에 없는 다른 항목을 사용할 수 있으며, 다른 조건으로 집계를 할 수 있다.

GROUP BY 구문을 사용할 필요없이 , SELECT 구문의 SUM 등의 집계함수에서 바로 사용합니다.

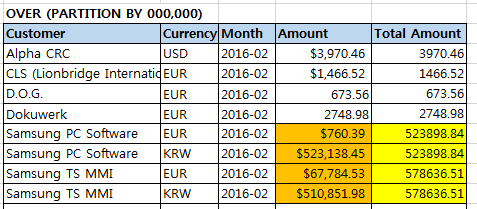
OVER 절을 사용하여 GROUP BY에 제한적이지 않은 각각의 항목을 사용할 수 있으며, 각기 다른 조건의 집계값을 도출 할 수 있다

참고: <https://www.ptr.co.uk/blog/sql-server-aggregate-functions-over-clause>

|  |
| --- |
| SELECT DISTINCT  SUM ({금액}) **OVER** (PARTITION BY {집계 항목1, 집계항목2…} ORDER BY {정렬1, 정렬2...})  AS [Column name] |

합계나 평균 등의 다양한 함수를 같이 할 수 있다.

|  |
| --- |
| --OVER 구  DECLARE @strIssuedMonth **char**(**8**);  SET @strIssuedMonth = '2016-02';    SELECT DISTINCT nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , ym\_invoice\_issued\_month AS 'Issued Month'  , SUM(real\_invoice\_amount) OVER (PARTITION BY nm\_customers\_name, nm\_currency\_unit ) AS [Amount]  , SUM(real\_invoice\_amount) OVER (PARTITION BY nm\_customers\_name ) AS [Total Amount ]  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month = @strIssuedMonth --'2017-08'  ORDER BY[nm\_custome문rs\_name] |



지정한 연도 범위에서 분기별, 고객별 수주합을 내면서, 분기 총액과 전체 금액을 집계하다. 아래와 같은 표를 만들고자 한다.

|  |
| --- |
|  |

이 리포트를 제출하기 위해 다음과 같은 Stored Procedures를 수행한다.

|  |
| --- |
| CREATE **PROCEDURE** sp\_topNAccountQuatorByWon @intTopRow **smallint**, @nmAccountYearFrom **nvarchar**(**20**), @nmAccountYearTo **nvarchar**(**20**) AS  -- 특정 연도 기간별 TOP N 고객매출 (분기별)  -- 분기/고객명/수주합/ 분기 총액/전체 수주 합을 집계한다  --sp\_topNAccountYearByWon 20, '2017', '2017'  --DECLARE @nmAccountYearFrom nvarchar(20), @nmAccountYearTo nvarchar(20);  --DECLARE @intTopRow smallint;  --SET @intTopRow = 100;  --SET @nmAccountYearFrom = '2017';  --SET @nmAccountYearTo =''  IF @nmAccountYearTo = ''  SET @nmAccountYearTo = YEAR(GETDATE());  IF @nmAccountYearFrom <> '' -- 특정 연도 기간을 정했을 때  BEGIN  SELECT DISTINCT TOP (@intTopRow) nm\_accounting\_period AS Period  , nm\_account\_name AS Customer  , SUM( int\_total\_amount) OVER (PARTITION BY nm\_**account**\_name, nm\_accounting\_**period**) AS [Quartered Amount] --고객 및 분기 수주 합  , SUM( int\_total\_amount) OVER (PARTITION BY nm\_accounting\_**period**) AS [Quartered Total Amount] -- 분기 수주 합  , SUM(int\_total\_amount) OVER () AS [Total Amount] -- 전체 수주 합  FROM OPPORTUNITY\_LIST  WHERE YEAR(ymd\_close\_date) BETWEEN @nmAccountYearFrom AND @nmAccountYearTo  AND  cd\_code\_stage = (SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name= N'수주확정' or nm\_code\_stage\_name = 'Closed Won')  ORDER BY nm\_accounting\_period, [Quartered Amount] DESC ;  END  ELSE BEGIN --ALL 전체 기간일 때  SELECT DISTINCT TOP (@intTopRow) nm\_accounting\_period AS Period  , nm\_account\_name AS Customer  , SUM( int\_total\_amount) OVER (PARTITION BY nm\_account\_name, nm\_accounting\_period) AS [Quartered Amount] --고객 및 분기 수주 합  , SUM( int\_total\_amount) OVER (PARTITION BY nm\_accounting\_period) AS [Quartered Total Amount] -- 분기 수주 합  , SUM(int\_total\_amount) OVER () AS [Total Amount] -- 전체 수주 합  FROM OPPORTUNITY\_LIST  WHERE  cd\_code\_stage = (SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name= N'수주확정' or nm\_code\_stage\_name = 'Closed Won')  ORDER BY nm\_accounting\_period, [Quartered Amount] DESC |

고객별 전체 딜의 개수와 수주액을 집계하면서 모든 고객의 수주액을 산출한다.

|  |
| --- |
|  |

해당 SP구문은 아래와 같다.

|  |
| --- |
| --TOP 고객 매출 현황  CREATE **PROCEDURE** sp\_topNAccountYearByWon @intTopRow **smallint**, @nmAccountYearFrom **nvarchar**(**20**), @nmAccountYearTo **nvarchar**(**20**) AS  --특정 연도 기간별 TOP N 고객매출  --sp\_topNAccountYearByWon 20, '2017', '2017'  IF @nmAccountYearTo = ''  SET @nmAccountYearTo = YEAR(GETDATE());  IF @nmAccountYearFrom <> '' -- 특정 연도 기간을 정했을 때  BEGIN  SELECT DISTINCT TOP (@intTopRow) nm\_account\_name AS Account  , SUM(int\_total\_amount) OVER (PARTITION BY nm\_account\_name) AS [Amount] -- 고객별 합계  , COUNT( id\_opportunity\_list) OVER(PARTITION BY nm\_account\_name ) AS [Counts] --영업기회 -건수  , SUM(int\_total\_amount) OVER () AS [Total Amount] -- 전체 수주 합  FROM OPPORTUNITY\_LIST  WHERE YEAR(ymd\_close\_date) BETWEEN @nmAccountYearFrom AND @nmAccountYearTo  AND cd\_code\_stage = (SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name= N'수주확정' or nm\_code\_stage\_name = 'Closed Won')  ORDER BY [Amount] DESC;  END  ELSE BEGIN --ALL 전체 기간일 때  SELECT DISTINCT TOP (@intTopRow) nm\_account\_name AS Account  , SUM(int\_total\_amount) OVER (PARTITION BY nm\_account\_name) AS [Amount] -- 고객별 합계  , COUNT(id\_opportunity\_list) OVER(PARTITION BY nm\_account\_name) AS [Counts] --영업기회 -건수  , SUM(int\_total\_amount) **OVER ()** AS [Total Amount] -- 전체 수주 합  FROM OPPORTUNITY\_LIST  WHERE cd\_code\_stage = (SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name= N'수주확정' or nm\_code\_stage\_name = 'Closed Won')  ORDER BY [Amount] DESC  END |

* 고객명, 발행월, 화폐기준으로 매출합
* 고객명, 화폐기준으로 2017년도 전체 매출합
* 월별, 화폐 기준으로 매출합,
* 화폐기준 2017년도 매출합

|  |
| --- |
| SELECT nm\_customers\_name AS Customer, ym\_invoice\_issued\_month AS Month , nm\_currency\_unit AS Currency  ,real\_invoice\_amount  , SUM(real\_invoice\_amount)  OVER(PARTITION BY **nm\_customers\_name**, **ym\_invoice\_issued\_month**, **nm\_currency\_unit**  ORDER BY nm\_customers\_name) AS 'Customer Monthly Total Amount'  , SUM(real\_invoice\_amount)  OVER(PARTITION BY **nm\_customers\_name**, **nm\_currency\_unit**  ORDER BY nm\_customers\_name) AS 'Customer Yearly Total Amount'    , SUM(real\_invoice\_amount)  OVER(PARTITION BY **ym\_invoice\_issued\_month,nm\_currency\_unit**) AS 'Monthly Amount'  , SUM(real\_invoice\_amount)  OVER(PARTITION BY **nm\_currency\_unit**) AS 'Yearly Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month LIKE '2017%'  ORDER BY nm\_customers\_name, ym\_invoice\_issued\_month |

|  |
| --- |
|  |

**DISTINCT** 키워드를 사용하여 중복을 제거하면 더욱 깔끔한 통계 자료를 하나로 뽑을 수 있다.

고객별 월 및 연도별 매출

월 및 연간 매출

|  |
| --- |
| SELECT DISTINCT nm\_customers\_name AS Customer, ym\_invoice\_issued\_month AS Month , nm\_currency\_unit AS Currency  , SUM(real\_invoice\_amount)  **OVER**(**PARTITION BY** nm\_customers\_name, ym\_invoice\_issued\_month, nm\_currency\_unit  ORDER BY nm\_customers\_name) AS 'Customer Monthly Total Amount'  , SUM(real\_invoice\_amount)  OVER(PARTITION BY nm\_customers\_name, nm\_currency\_unit  ORDER BY nm\_customers\_name) AS 'Customer Yearly Total Amount'    , SUM(real\_invoice\_amount)  OVER(PARTITION BY ym\_invoice\_issued\_month,nm\_currency\_unit) AS 'Monthly Amount'  , SUM(real\_invoice\_amount)  OVER(PARTITION BY nm\_currency\_unit) AS 'Yearly Amount'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month LIKE '2017%'  ORDER BY nm\_customers\_name, ym\_invoice\_issued\_month |

|  |
| --- |
|  |

UPDATE SET 구문

Update Set Where

|  |
| --- |
| **UPDATE** table\_name  **SET** column\_name = value  [**,** column\_name = value ... ]  [**WHERE** condition] --Where 조건에 해당하는 레코드만 수정한다. Where절 없으면 전체 컬럼 변경 |

-- Update의 기준

-- **업데이트의 기준은 어디까지나 WHERE 절이라는 거죠..**

UPDATE SET FROM JOIN

|  |
| --- |
| -- 임시테이블TEMP\_STD테이블에 스타일ID를 갱신하라.  -- TB\_SNTC\_STYLE 테이블에서 매뉴얼ID와 스타일명을 찾아 값을 일괄 갱신한다.  UPDATE t  **SET** t.style\_id = s.style\_id  **FROM** TEMP\_STD t  INNER JOIN TB\_SNTC\_STYLE s  ON t.style\_nm = s.style\_nm AND t.manual\_id = s.manual\_id; |

|  |
| --- |
| -- 표준문장 임시테이블에서 상태코드를 일괄 부여한다.  -- 챕터ID,문장ID, 언어코드  UPDATE t  SET t.status\_cd = (  CASE WHEN a.confirm\_yn ='N' THEN 'S903' --미정  WHEN a.confirm\_yn is null THEN 'S901' -- 승인  ELSE 'S902' -- 거절  END  )  FROM TEMP\_STD t |

사례1: Inner Join을 통한 업데이트

단가 테이블에서 UM통합 단가를 'QRG','QSG',’Leaflet’ 매뉴얼에도 동일하게 업데이트하기.

SQL UPDATE 구문

|  |
| --- |
| **UPDATE** qr  **SET** qr.BILL\_UNIT\_AMT = um.BILL\_UNIT\_AMT  **FROM** HSP2021 qr  **INNER JOIN** HSP2021 um  ON qr.CUST\_ID=um.CUST\_ID AND qr.CUST\_DEPT\_ID=um.CUST\_DEPT\_ID AND qr.BILL\_FORM=um.BILL\_FORM AND qr.BILL\_CODE\_YEAR=um.BILL\_CODE\_YEAR  AND qr.UNIT\_CODE1=um.UNIT\_CODE1 AND qr.BILL\_ITEM = um.BILL\_ITEM  WHERE um.CUST\_ID=36 AND um.CUST\_DEPT\_ID=34 AND um.BILL\_FORM='GL\_MU' AND um.BILL\_CODE\_YEAR='2018' AND um.MANU\_TYPE='UM\_UNI'  AND qr.MANU\_TYPE IN ('QRG','QSG','Leaflet') |

-- AA.a가 1이고 AA.b가 10인 모든데이타를 대상으로 BB.a가 1이고 BB.b가 10보다 작은 BB테이블의 BB.c값으로 업데이트함

-- SET절의 Subquery가 없으면 모두 NULL 값이 들어가는 것을 방지해야 한다.

|  |
| --- |
| --- SQL기준  UPDATE AA  SET c = (SELECT BB.c FROM BB WHERE a = AA.a AND b < AA.B)  WHERE a=1 AND b=10;  UPDATE tt  SET tt.c= jj.c  FROM AA tt  INNHER JOIN BB jj  ON tt.a = jj.a  WHERE tt.b > jj.b  WHERE tt.a = 1 AND tt.b = 10; |

사례2: 오라클 UPDATE 구문

* 수정가능한 조인뷰를 사용하는 경우

|  |
| --- |
| -- 같은 테이블에서 값을 가져와서 다른 조건의 레코드를 업데이트하기  --JOIN을 이용하는 경우에는  /\*+ BYPASS\_UJVC \*/ 라는 힌트를 이용해서 UPDATE한다.  **UPDATE /\*+ BYPASS\_UJVC \*/**  **(**  SELECT qr.BILL\_UNIT\_AMT **qrBill**  , qr.BILL\_UNIT\_CD **qrCD**  , um.BILL\_UNIT\_AMT **umBill**  , um.BILL\_UNIT\_CD **umCD**  **FROM** HSP2021 qr  , HSP2021 um  **WHERE** qr.CUST\_ID=um.CUST\_ID AND qr.CUST\_DEPT\_ID=um.CUST\_DEPT\_ID AND qr.BILL\_FORM=um.BILL\_FORM AND qr.BILL\_CODE\_YEAR=um.BILL\_CODE\_YEAR  AND qr.UNIT\_CODE1=um.UNIT\_CODE1 AND qr.BILL\_ITEM = um.BILL\_ITEM  AND um.CUST\_ID=36 AND um.CUST\_DEPT\_ID=34 AND um.BILL\_FORM='GL\_MU' AND um.BILL\_CODE\_YEAR='2018' AND um.MANU\_TYPE='UM\_UNI'  AND qr.MANU\_TYPE IN ('QRG','QSG','Leaflet')  **)**  **SET** **qrBill** = **umBill**  , **qrCD** = **umCD**  COMMIT; |

-- AA.a가 1이고 AA.b가 10인 모든데이타를 대상으로 BB.a가 1이고 BB.b가 10보다 작은 BB테이블의 BB.c값으로 업데이트함

-- SET절의 Subquery가 없으면 모두 NULL 값이 들어가는 것을 방지해야 한다.

|  |
| --- |
| UPDATE AA  SET c = NVL(( select BB.c  from BB  where BB.a = AA.a  and BB.b < AA.b ), AA.c)  WHERE AA.a = 1  AND AA.b = 10;      또는 불필요한 업데이틀 방지하기 위해 EXISTS조건을 줄 수 있다.  UPDATE AA  SET c = ( select BB.c  from BB  where BB.a = AA.a  and BB.b < AA.b )  WHERE AA.a = 1  AND AA.b = 10;  AND EXISTS( SELECT 1 from BB  where BB.a = AA.a  and BB.b < AA.b )  --수정 가능 조인뷰  UPDATE ( select AA.c AA\_C,  BB.c BB\_C  from AA, BB  where AA.a = 1  and AA.b = 10  and BB.a = AA.a  and BB.b < AA.b )  SET AA\_C = BB\_C |

INSERT INTO VALUES 구문

|  |
| --- |
| **INSERT INTO** CODE\_LOSSREASON (  cd\_code\_rossreason  ,nm\_code\_rossreason\_name  ,no\_disp\_order  ) VALUES (  **5** -- cd\_code\_rossreason - smallint  ,'No Budget / Lost Fundinig' -- nm\_code\_rossreason\_name - nvarchar(100)  ,'5' -- no\_disp\_order - varchar(2)  ) |

JOIN 복수 테이블 연계

SQL: INNER JOIN

INNER JOIN은 두개의 테이블에서 검색 조건을 서로 만족하는 레코드를 가져옵니다. 집합으로 표현하면 교집합에 해당합니다.

두개 이상의 테이블을 INNER JOIN하는 경우, Inline sql문을 컬럼이나 @변수로 처리하곤 합니다. 하지만 조금만 더 생각하면 **다중 INNER JOIN**으로 간단하게 해결 가능하답니다.

SELECT

**FROM** A-TABLE **a**

**INNER JOIN** B-TABLE b

**ON** {결합관계식:a.key = b.key}

**INNER JOIN** C-TABLE c

**ON** b.key = c.key

**WHERE** {결합관계로 인한 결과값 필터링식}

|  |
| --- |
| -- UPDATE구분에서 복수 INNER JOIN  -- 고객사, 화폐코드, 직원을 동시에 갱신하기  UPDATE i SET  i.id\_tbl\_customers\_list = c.id\_tbl\_customers\_list -- smallint  ,i.id\_code\_resource\_currency=d.id\_code\_resource\_detail  , i.id\_emp = e.id\_emp  FROM BM\_INVOICE **i**  INNER JOIN TBL\_CUSTOMERS\_LIST c  ON i.Customer = c.nm\_customers\_name  INNER JOIN CODE\_RESOURCE\_DETAIL d  ON d.id\_code\_resource = (SELECT id\_code\_resource FROM CODE\_RESOURCE WHERE code\_code\_resource =N'CURRENCY' )  AND i.Currencies = d.tx\_comment  INNER JOIN TBL\_COMPANY\_EMP e  ON i.Creator = e.nm\_emp\_name    SELECT DISTINCT Customer, id\_tbl\_customers\_list FROM BM\_INVOICE  SELECT DISTINCT Currencies, id\_code\_resource\_currency FROM BM\_INVOICE  SELECT DISTINCT id\_emp, Creator FROM BM\_INVOICE |

실제로는 3개의 업데이트를 수행하는 걸 줄일 수 있다.

|  |
| --- |
| -- 1.고객사 ID 갱신  UPDATE i SET  i.id\_tbl\_customers\_list = c.id\_tbl\_customers\_list -- smallint  FROM BM\_INVOICE i  INNER JOIN TBL\_CUSTOMERS\_LIST c  ON i.Customer = c.nm\_customers\_name  --SELECT DISTINCT Customer, id\_tbl\_customers\_list FROM BM\_INVOICE    -- 2.화폐코드 갱신  UPDATE i SET  i.id\_code\_resource\_currency=c.id\_code\_resource\_detail  FROM BM\_INVOICE i  INNER JOIN CODE\_RESOURCE\_DETAIL c  ON c.id\_code\_resource = (SELECT id\_code\_resource FROM CODE\_RESOURCE WHERE code\_code\_resource =N'CURRENCY' )  AND i.Currencies = c.tx\_comment    --SELECT DISTINCT Currencies, id\_code\_resource\_currency FROM BM\_INVOICE  /\* EUR 396  KRW 835  USD 394  \*/  -- 3.인보이스 발행자 ID 갱신  UPDATE i SET  i.id\_emp = e.id\_emp  FROM BM\_INVOICE i  INNER JOIN TBL\_COMPANY\_EMP e  ON i.Creator = e.nm\_emp\_name    -- SELECT DISTINCT id\_emp, Creator FROM BM\_INVOICE |

|  |
| --- |
| --출발어와 도착어를 기준으로 리소스 검색  --벤더명, 분야, 역할별로 정렬  DECLARE @slang **smallint**  DECLARE @tlang **smallint**  SET @slang = (select id\_code\_langs\_list FROM CODE\_LANGS\_LIST where nm\_langs\_code like 'English')  SET @tlang = (select id\_code\_langs\_list FROM CODE\_LANGS\_LIST where nm\_langs\_code like 'KOREAN')  --SELECT @slang, @tlang  -- Multi-Inner Join  SELECT r.sub\_resource\_type ,r.nm\_region\_info  , c.nm\_country  , r.nm\_name\_venorperson  ,l.nm\_locale\_start\_lang,l.nm\_locale\_target\_lang, l.nm\_translation\_field, l.nm\_tranreview\_role ,l.nm\_years\_experience, l.no\_translation\_rate, l.nm\_job\_unit, l.nm\_currency\_name  FROM TBL\_RESOURCE\_LIST r  INNER JOIN RESOURCE\_LOCALE\_LIST l  ON r.id\_tbl\_resource\_list = l.id\_tbl\_resource\_list  INNER JOIN RESOURCE\_COUNTRY c  ON l.id\_tbl\_resource\_list = c.id\_tbl\_resource\_list  WHERE l.id\_code\_langs\_start = @slang AND l.id\_code\_langs\_target = @tlang  ORDER BY r.nm\_name\_venorperson ASC, l.nm\_translation\_field ASC, l.nm\_tranreview\_role |

엑셀데이타를 DB에 적재하는 쿼리문

|  |
| --- |
| INSERT INTO RESOURCE\_COUNTRY (  id\_tbl\_resource\_list  ,id\_code\_resource\_detail\_country  ,nm\_country  )  -- 엑셀자료에서 벤더명과 국가정보를 얻어 리소스\_국가 테이블에 자료 등록하기 06/22/2017  SELECT  r.id\_tbl\_resource\_list -- 벤더ID  ,c.id\_code\_resource\_detail as country\_code --국가ID  , c.nm\_code\_resource\_detail as country\_name --국가명  FROM TBL\_RESOURCE\_LIST r  INNER JOIN  ( --엑셀자료 필터링한 메모리 테이블  SELECT DISTINCT NameofVendor\_Translator as name, Country as country FROM RESOURCE\_BATCH\_MIG1  GROUP BY NameofVendor\_Translator, Country  ) as excel  ON r.nm\_name\_venorperson = excel.[name] --엑셀 벤더명으로 리소스 테이블 조인  LEFT JOIN CODE\_RESOURCE\_DETAIL c  ON c.nm\_code\_resource\_detail = excel.Country and c.id\_code\_resource= **47** --엑셀 국가명으로 국가테이블 조인 |

SQL: LEFT JOIN

SELECT A.a, A.b, B.c, B.d

FROM A

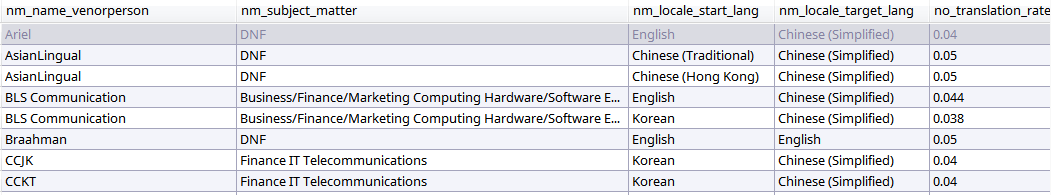
LEFT JOIN B

ON A.id = B.id

집합 A의 전체 레코드를 가져오되, A레코드에서 B에 연결된 레코드가 있다면 해당 컬럼 값을 보여줍니다. 연결된 B의 정보가 없으면 NULL로 값을 표시합니다.

집합 A에 n개, 집합 B에 m개의 레코드가 있다면 n\*m개의 전체 레코드를 가져올 수 있습니다.

|  |
| --- |
| SELECT r.nm\_name\_venorperson, r.nm\_subject\_matter  , l.nm\_locale\_start\_lang, l.nm\_locale\_target\_lang, l.no\_translation\_rate  FROM TBL\_RESOURCE\_LIST r  LEFT JOIN RESOURCE\_LOCALE\_LIST l  On l.id\_tbl\_resource\_list = r.id\_tbl\_resource\_list  WHERE l.no\_translation\_rate between '0.02' and '0.05' /\*리소스별로 다수의 로케일 정보를 가지고 있다. \*/  ORDER BY r.nm\_name\_venorperson asc |



SQL: RIGHT JOIN

LEFT JOIN의 반대 개념으로 오른쪽 테이블 내용을 전체 보여주고, 교차된 영역의 데이터는 추가로 가져오게 된다.

UNION: 검색결과의 합

UNION

두 테이블의 검색 결과를 **중복을 제거하여 보여줍니다**.

쿼리로 가져오는 컬럼에 **DISTINCT 중복제거를 실시하여 가져옵니다**.

|  |
| --- |
| SELECT id\_tbl\_resource\_list, nm\_name\_venorperson FROM TBL\_RESOURCE\_LIST  **UNION**  SELECT id\_tbl\_resource\_list, nm\_locale\_start\_lang FROM RESOURCE\_LOCALE\_LIST  ORDER BY id\_tbl\_resource\_list ASC |

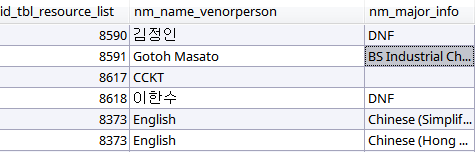


UNION ALL

두 테이블의 검색 결과를 단순히 합하여 보여줍니다.

이때 주의할 것은 두 테이블에서 불러오는 컬럼의 개수가 서로 동일해야 합니다.

|  |
| --- |
| SELECT id\_tbl\_resource\_list, nm\_name\_venorperson, nm\_major\_info FROM TBL\_RESOURCE\_LIST  **UNION ALL**  SELECT id\_tbl\_resource\_list, nm\_locale\_start\_lang, nm\_locale\_target\_lang FROM RESOURCE\_LOCALE\_LIST |



|  |
| --- |
| SELECT chapter\_id, sntc\_std\_id, lang\_cd, sntc\_std\_nm  FROM TB\_STD\_SNTC  WHERE lang\_cd = 'en-US'  GROUP BY chapter\_id, sntc\_std\_id, lang\_cd, sntc\_std\_nm  --ORDER BY chapter\_id, sntc\_std\_id  UNION ALL  SELECT chapter\_id, sntc\_std\_id, lang\_cd, sntc\_std\_nm  FROM TB\_STD\_SNTC  WHERE lang\_cd = 'ko-KR'  GROUP BY chapter\_id, sntc\_std\_id, lang\_cd, sntc\_std\_nm  --ORDER BY chapter\_id, sntc\_std\_id |

ALTER TABLE 테이블 변경

테이블 관련 작업 요약

|  |  |  |  |
| --- | --- | --- | --- |
| **작업 유형** | | **SQL 문장** | **비고** |
| 컬럼 추가 / 변경 | |  |  |
| 1 | 컬럼 추가 | --컬럼 추가  ALTER TABLE OPPORTUNITY\_LIST  ADD nm\_code\_rossreason\_name nvarchar(100) null ; |  |
|  |  | ALTER TABLE OPPORTUNITY\_LIST  ADD opp\_list\_id int IDENTITY(1,1); | IDENTITY(1,1) |
| 2 | 컬럼 변경 | --컬럼 타입 변경  ALTER TABLE PROJECT\_LANGS\_TASKS  ALTER COLUMN yn\_use CHAR(1) DEFAULT ‘Y’;  ALTER **TABLE** BM\_INVOICE  ALTER COLUMN [Invoice amount] **REAL** NULL; |  |
| 제한 조건 추가 | |  |  |
| 1 | 식별키 PK | ALTER TABLE [COMPANY\_OWNER]  ADD CONSTRAINT COMPANY\_OWNER\_PK  PRIMARY KEY ( [id\_company\_owner] ) |  |
| 2 | 외래키 FK | ALTER TABLE OPPORTUNITY\_LIST  ADD CONSTRAINT OPPORTUNITY\_LIST\_FK9  FOREIGN KEY (cd\_code\_rossreason) REFERENCES CODE\_LOSSREASON(cd\_code\_rossreason); |  |
| 3 | 값 체크 | ALTER TABLE OPPORTUNITY\_LIST  ADD CONSTRAINT OPPORTUNITY\_LIST\_CK  CHECK(yn\_loss = 'LOSS' OR yn\_loss='OTHR'); |  |
| 4 | 고유 컬럼 | CREATE UNIQUE INDEX COMPANY\_OWNER\_UK  ON COMPANY\_OWNER( nm\_company\_owner\_name ) | 인덱스 생성 |
| ALTER TABLE COMPANY\_OWNER  ADD CONSTRAINT COMPANY\_OWNER\_UK  UNIQUE ( nm\_company\_owner\_name ) |  |
|  |  |  |  |
|  |  |  |  |

ADD 컬럼 추가

|  |
| --- |
| -- ALTER **TABLE** '테이블명' ADD '컬럼명' '데이타타입' [NULL/NOT NULL];  ALTER **TABLE** TBL\_CUSTOMERS\_LIST  ADD nm\_contact\_persons **NVARCHAR**(**100**) NULL;  ALTER **TABLE** TBL\_CUSTOMERS\_LIST  ADD nm\_email\_address **NVARCHAR**(**200**) NOT NULL; |

ALTER COLUMN 타입 변경

|  |
| --- |
| --테이블의 컬럼 타입 변경하기  -- ALTER **TABLE** '테이블명' ALTER COLUMN '컬럼명' '데이타타입' [NULL/NOT NULL];  예시:  ALTER **TABLE** PROJECT\_LANGS\_TASKS ALTER COLUMN int\_work\_amout\_remained **INT** NULL ;  ALTER **TABLE** PROJECT\_LANGS\_TASKS ALTER COLUMN int\_work\_amount\_acclumulated **VARCHAR**(**100**) NOT NULL;  ALTER **TABLE** PROJECT\_LANGS\_TASKS ALTER COLUMN yn\_use **CHAR**(**1**) DEFAULT ‘Y’; |

ADD CONSTRAINT 제약조건 추가

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

|  |  |
| --- | --- |
| PRIMARY KEY | ALTER **TABLE** [COMPANY\_OWNER]  ADD CONSTRAINT [COMPANY\_OWNER\_PK]  **PRIMARY** **KEY** ( [id\_company\_owner] ) |
| FOREIGN KEY | ALTER **TABLE** OPPORTUNITY\_OWNER  ADD CONSTRAINT OPPORTUNITY\_OWNER\_FK  **FOREIGN** **KEY** (id\_company\_owner) **REFERENCES** COMPANY\_OWNER (id\_company\_owner) |
| CHECK | ALTER **TABLE** NETWORK\_ALLOW\_IP  ADD CONSTRAINT NETWORK\_ALLOW\_IP\_C  **CHECK** (sub\_ipaddress\_range = 'S' OR sub\_ipaddress\_range = 'R' ) |
| UNIQUE | CREATE UNIQUE **INDEX** [COMPANY\_OWNER\_UK] ON [COMPANY\_OWNER]  ( [nm\_company\_owner\_name] )  ALTER **TABLE** [COMPANY\_OWNER]  ADD CONSTRAINT [COMPANY\_OWNER\_UK]  **UNIQUE** ( [nm\_company\_owner\_name] ) |
| DEFAULT |  |

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

* **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* **FOREIGN KEY** - Uniquely identifies a row/record in another table
* **CHECK** - Ensures that all values in a column satisfies a specific condition
* **DEFAULT** - Sets a default value for a column when no value is specified
* **INDEX** - Use to create and retrieve data from the database very quickly
* **UNIQUE** - Ensures that **all values in a column are different**
* **NOT NULL** - Ensures that a column cannot have a NULL value

PRIMARY KEY

* 테이블 내에 각 레코드를 식별하는데 사용한다.
* 고유한 값을 가져야 한다.
* NULL값을 가질 수 없다.

ID int NOT NULL PRIMARY KEY 또는 PRIMARY KEY (ID, ORDER\_NUMBER)

|  |
| --- |
| ALTER **TABLE** [COMPANY\_OWNER]  ADD CONSTRAINT [COMPANY\_OWNER\_PK] PRIMARY **KEY** ( [id\_company\_owner] ) |

FOREIGN KEY REFERENCES

외래키로 인해 하나의 테이블에 존재하는 레코드는 다른 테이블에 의존적이게 된다. 외래키를 설정할 때는 반드시 Primary key와 Unique 제약조건이 설정이 되어야 가능하다.

|  |
| --- |
| --컬럼이 없는 경우, ADD 컬럼해야 한다.  ALTER **TABLE** OPPORTUNITY\_OWNER  ADD CONSTRAINT OPPORTUNITY\_OWNER\_FK  FOREIGN **KEY** (id\_company\_owner) REFERENCES COMPANY\_OWNER (id\_company\_owner);  -- id\_code\_resource\_role  ALTER TABLE RESOURCE\_LOCALE\_LIST  ADD CONSTRAINT RESOURCE\_LOCALE\_LIST\_FK6  FOREIGN KEY (id\_code\_resource\_role)  REFERENCES CODE\_RESOURCE\_DETAIL ( id\_code\_resource\_detail )  -- id\_code\_resouce\_jobunit  ALTER TABLE RESOURCE\_LOCALE\_LIST  ADD CONSTRAINT RESOURCE\_LOCALE\_LIST\_FK7  FOREIGN KEY (id\_code\_resouce\_jobunit)  REFERENCES CODE\_RESOURCE\_DETAIL ( id\_code\_resource\_detail )  -- id\_code\_resource\_country  ALTER TABLE RESOURCE\_LOCALE\_LIST  ADD CONSTRAINT RESOURCE\_LOCALE\_LIST\_FK8  FOREIGN KEY (id\_code\_resource\_country)  REFERENCES CODE\_RESOURCE\_DETAIL ( id\_code\_resource\_detail )  -- id\_code\_resource\_yearsexp  ALTER TABLE RESOURCE\_LOCALE\_LIST  ADD CONSTRAINT RESOURCE\_LOCALE\_LIST\_FK9  FOREIGN KEY (id\_code\_resource\_yearsexp)  REFERENCES CODE\_RESOURCE\_DETAIL ( id\_code\_resource\_detail ) |

CHECK

테이블의 특정 컬럼에 값을 체크하는 조건을 지정합니다. 이렇게 함으로 해당 컬럼에 적합치 않은 값을 사전에 방지할 수 있습니다.

**sub\_resource\_type = 'Vendors' OR sub\_resource\_type = 'Freelancers' OR sub\_resource\_type = 'Team Vendors'**

Toad Tools에서 Constraints의 Column List에 넣을 값

|  |
| --- |
| (  [sub\_resource\_type] = 'Vendors' OR [sub\_resource\_type] = 'Freelancers' OR [sub\_resource\_type] = 'Team Vendors'  ) |

SQL 스크립트

|  |
| --- |
| ALTER TABLE [TABLE\_NAME1]  ADD CONSTRAINT [C\_TABLE\_NAME1]  **CHECK**  (  [sub\_resource\_type] = 'Vendors' OR [sub\_resource\_type] = 'Freelancers' OR [sub\_resource\_type] = 'Team Vendors'  ) |

UNIQUE

특정 열에 중복된 값이 입력되는 것을 막아 고유한 값만을 갖게 하는 것이다.

여러 컬럼에 유니크를 설정할 때는 UNIQUE( column1, column2)

단일 컬럼에 유니크를 설정할 때는 UNIQUE 붙임

|  |
| --- |
| CREATE UNIQUE **INDEX** [COMPANY\_OWNER\_UK] ON [COMPANY\_OWNER]  ( [nm\_company\_owner\_name] )  ALTER **TABLE** [COMPANY\_OWNER]  ADD CONSTRAINT [COMPANY\_OWNER\_UK]  UNIQUE ( [nm\_company\_owner\_name] )  -- 복수 컬럼에 유니크 지정하기  ALTER TABLE Persons  ADD **CONSTRAINT** Person\_UC **UNIQUE** **(**ID, LastName**);** |

|  |
| --- |
| --테이블 생성할 때  CREATE TABLE tableName (  ID int NOT NULL **UNIQUE** ,  Name varchar(200) NOT NULL,  ); |

IDENTITY 키값 자동 증가

PK가 되는 컬럼 속성에 Identity를 부여해야 한다.

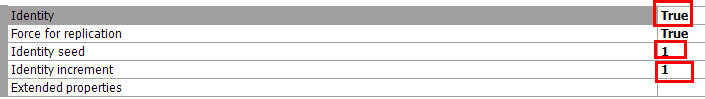
**설정: 초기값 10부터 시작하여 5씩 증가하기**

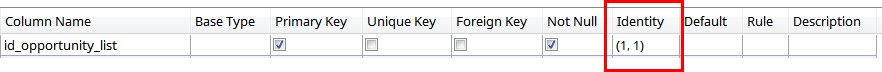
|  |  |  |
| --- | --- | --- |
| Identity Seed | increment | Seed: 시작값  Inc. : 증감값 |
| 10 | 5 |

**SQL구문**

|  |
| --- |
| CREATE TABLE PERSONS(  ID int **IDENTITY(10,5)** PRIMARY KEY,  LastName varchar(255) NOT NULL,  Age int  ); |

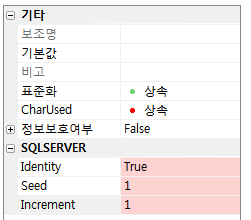
**Toad 툴에서** 설정





**DA# 툴에서 설정**

**물리설**계 > 컬럼 > [Indentity, Seed, Increment]



## T-SQL(Transact-SQL) 프로그래밍

[SQL 함수]

<https://docs.microsoft.com/ko-kr/sql/t-sql/functions/functions>

문자열(STRING) 함수

|  |
| --- |
| -- String Functions  RIGHT(character\_expression, integer\_expression)  RTRIM(character\_expression)  PATINDEX('%pattern%', expression) -- returns the position of ther first occurence of a pattern in a string  SELECT PATINDEX('% (%', 'IN00-0120088 (2017-08-30)') -- result '13'  SUBSTRING(expression, start, \_length)  /\* extract a part of a string  특정문자열 'IN00-0120088 (2017-08-30)'  추출하고자 하는 문자열 : IN00-0120088  patern : ' %'  \*/  SELECT SUBSTRING('IN00-0120088 (2017-08-30)', **1**, (PATINDEX('% (%', 'IN00-0120088 (2017-08-30)'))-**1**) -- result 'IN00-0120088'  SELECT PATINDEX('% (%', 'IN00-0120088 (2017-08-30)') -- result '13'  SELECT SUBSTRING('IN00-0120088 (2017-08-30)', **1**, **12**) -- result 'IN00-0120088'    STR(float\_expression)/STR(float\_expression, length)/STR(float\_expression, length, **decimal**) -- convert numeric data to character strings  TRY\_PARSE(**$string\_value$** as data\_type) |

CONCAT: 문자열 합치기

두개 이상의 문자열을 연결하여 하나의 문자열로 반환합니다.. CONCATENATION하는 동안 문자열 분리값을 추가하려면 CONCAT\_WS를 사용하세요.

CONCAT: SQL Server(**2012부터 시작**) 사용할 수 있는 함수입니다. 둘이상의 문자열을 연결한 문자열을 반환합니다.

|  |
| --- |
| CONCAT ( string\_value1, string\_value2 [, string\_valueN ] )  Return Types:  스트링이며, 길이와 타입은 입력되는 것에 따라 다르다 |

|  |
| --- |
| SELECT CONCAT( emp\_name, emp\_middlename, emp\_lastname ) AS Result  FROM #temp; |

구분자(Delimeter)로 문자열 나누기

문자열 추출하기(PATINDEX & SUBSTRING)

|  |
| --- |
| PATINDEX('%pattern%', expression) -- returns the position of ther first occurence of a pattern in a string  SELECT PATINDEX('% (%', 'IN00-0120088 (2017-08-30)') -- result '13'  --찾지 못하면 0을 리턴합니다.  SELECT PATINDEX('% (%','IN00-0120028(2017-08-09)') -- 0 붙어있는 경우  SELECT PATINDEX('% (%','IN00-0120028 2017-08-09)') -- 0 공백만 있는 경우  SELECT PATINDEX('% (%','IN00-0120028)') -- 0 뒤가 없는 경우  SUBSTRING(expression, start, \_length)  /\* extract a part of a string  expression: 'IN00-0120088 (2017-08-30)'  start: 추출할 시작위치  length: 가져올 문자수  원하는 문자열 : IN00-0120088  구분자 : ' %'  \*/  SELECT SUBSTRING(expression, **1**, PATINDEX('%pattern\_delimeter%',expression)-**1**)  **int** strindex = PATINDEX('%pattern\_delimeter%',expression);  IF strindex == **0** OR strindex < **12**  SELECT expression  ELSE  BEGIN  SELECT SUBSTRING('IN00-0120088 (2017-08-30)', **1**, (PATINDEX('% (%', 'IN00-0120088(2017-08-30)'))-**1**) -- result 'IN00-0120088'  SELECT PATINDEX('% (%', 'IN00-0120088(2017-08-30)') -- result '13'  SELECT SUBSTRING('IN00-0120088 (2017-08-30)', **1**, **12**) -- result 'IN00-0120088'  END |

문자열 나누기(CHARNDEX & RIGHT/LEFT)

|  |
| --- |
| -- 구분자 '.'을 기준으로 두개의 문자열로 나눠지는 경우.  -- excel\_items\_no = OR16-190010.05  -- @langSeqNo = '005'  SET @strLength = LEN(@excelItemsNo);  SET @startIndex = CHARINDEX('.',@excelItemsNo);  SET @langSeqNo = '0' + RIGHT(@excelItemsNo, @strLength-@startIndex);    -- @prjCode = 'OR16-190010'  SET @prjCode = LEFT(@excelItemsNo,@startIndex-**1**);  PRINT @prjCode + ':' + @langSeqNo; |

|  |
| --- |
| -- 구분자 '.'을 기준으로 두개의 문자열로 나눠지는 경우.  -- OR16-200329.00015  DECLARE @srcString VARCHAR(100);  DECLARE @strLength smallint;  DECLARE @langSeqNo VARCHAR(100);  DECLARE @prjCode VARCHAR(100);  SET @srcString = 'OR16-200329.00015';  SET @strLength = LEN(@srcString);  -- Get a code\_sequence\_no  SET @langSeqNo = RIGHT(@srcString, @strLength-CHARINDEX('.',@srcString));  SET @prjCode = (LEFT(@srcString, CHARINDEX('.',@srcString)-1));  SELECT @prjCode AS PRJ\_CODE, @langSeqNo AS LANG\_SEQ; |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | OR16-200329 | 00015 | |

여러 문자열을 나누기(CHARINDEX & SUBSTRING)

|  |
| --- |
| -- 구분자 '-'를 기준으로 여러개의 문자열로 나누어지는 경우  -- OR16-200329-TRA-015  DECLARE @srcString **VARCHAR**(**100**);  DECLARE @strLength **smallint**;  DECLARE @startIndex **smallint**;  SET @srcString = 'OR16-200329-TRA-015';  SET @strLength = LEN(@srcString);  SET @startIndex = CHARINDEX('-',@srcString);  SELECT @srcString, @strLength, @startIndex;  -- 1차 가공: '200329-TRA-015'  SET @srcString = (SELECT SUBSTRING(@srcString, @startIndex+**1**, @strLength-@startIndex));  SET @strLength = (SELECT LEN(@srcString));  SET @startIndex = (SELECT CHARINDEX('-',@srcString));  SELECT @srcString, @strLength, @startIndex;  -- 2차 가공: 'TRA-015';  SET @srcString = (SELECT SUBSTRING(@srcString,@startIndex+**1**, @strLength-@startIndex));  SET @strLength = (SELECT LEN(@srcString));  SET @startIndex = (SELECT CHARINDEX('-',@srcString));  SELECT @srcString, @strLength, @startIndex; |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| String Length Index   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | OR16-200329-TRA-015 | | | | | | | 19 | 5 | | 200329-TRA-015 | | | | 14 | 7 | | TRA-015 | 7 | 4 | |

PATINDEX vs CHARINDEX

|  |
| --- |
| -- returns the position of ther first occurence of a **pattern** in a string  PATINDEX('%pattern%', expression)  SELECT PATINDEX('**%** (**%**', 'IN00-0120088 (2017-08-30)')  -- result '13' |

|  |
| --- |
| --Returns the position of the first occurrence of a **substring** within a string  CHARINDEX(expression1, expression2)  CHARINDEX(expression1, expression2, start\_location)  SELECT CHARINDEX(' (', 'IN00-0120088 (2017-08-30)')  -- result '13' |

SUBSTRING

|  |
| --- |
| SUBSTRING(expression, start, \_length)  /\* extract a part of a string  expression: 'IN00-0120088 (2017-08-30)'  start: 추출할 시작위치  length: 가져올 문자수  원하는 문자열 : IN00-0120088  구분자 : ' %'  \*/  SELECT SUBSTRING('OR16-200329-TRA-015', **13**, LEN('OR16-200329-TRA-015')-13);  -- 출력 : TRA-015 |

LTRIM & RTRIM

|  |
| --- |
| -- Items컬럼 좌우측 공백 지우기  UPDATE BM\_INVOICE\_DETAIL SET  Items = RTRIM(LTRIM(Items)) |

문자열 앞에 있는 공백을 제거한 문자열을 반환합니다.

|  |
| --- |
| SELECT LTRIM(' Five spaces are at the beginning of this string.') FROM sys.databases; |

문자열 끝에 있는 공백을 모두 잘라낸 문자열을 반환합니다

|  |
| --- |
| SELECT RTRIM('Removes trailing spaces. '); |

LEFT

지정된 문자수 만큼 문자열의 왼쪽을 반환한다.

|  |
| --- |
| -- LEFT  -- 첫인자는 문자열  -- 두번째 인자는 가져올 문자수  SELECT LEFT('OR16-200329-TRA-015', **5**)  -- OR16- |

RIGHT

지정된 문자수 만큼 문자열 오른쪽을 반환한다

|  |
| --- |
| --RIGHT  -- 첫인자는 문자열  -- 두번째 인자는 가져올 문자수  SELECT RIGHT('OR16-200329-TRA-015', **7**)  -- TRA-015 |

|  |
| --- |
| SELECT RIGHT(‘abcdefg’, 2) – ‘fg’ |

ASCII

입력된 표현식에서 가장 왼쪽 문자에 해당하는 ASCII코드값을 반환한다.

* SELECT ASCII('A') AS A, ASCII('ABC') AS ABC --결과는 동일한 65

CHAR

int ASCII 코드값을 해당 문자로 변환하여 반환한다. 특히, 제어문자를 문자열에 삽입하는데 주로 사용한다.

|  |  |
| --- | --- |
| 제어문자 | 값 |
| 탭 | CHAR(9) |
| 줄바꿈 | CHAR(10) |
| 캐리지 리턴 | CHAR(13) |

LOWER

대문자를 소문자로 변환한다.

NCHAR

PATINDEX

QUOTENAME

REPLACE

REPLICATE

REVERSE

SOUNDEX

SPACE: 지정된 개수만큼 반복되는 공백을 반환합니다. 8000개 이상은 REPLICATE를 사용합니다.  
STR

STRING\_AGG

STRING\_ESCAPE

STRING\_SPLIT

STUFF

CONCAT ( string\_value1, string\_value2 [, string\_valueN ] )

CONCAT\_WS

DIFFERENCE

FORMAT

**TRANSLATE**

**TRIM**

UNICODE

UPPER: 소문자를 대문자로 변환한다.

집계(Aggregate) 함수

AVG/CHECKSUM\_AGG/COUNT/COUNT\_BIG/GROUPING/GROUPING\_ID/MAX/MIN/SUM/STDEV/STDEVP/VAR/VARP

분석(Analytic) 함수

CUME\_DIST/FIRST\_VALUE/LAG/LAST\_VALUE/LEAD/PERCENTILE\_CONT/PERCENTILE\_DISC/PERCENT\_RANK

순위(Ranking) 함수

RANK/DENSE\_RANK/NTILE/ROW\_NUMBER

행집합(Rowset) 함수

**OPENDATASOURCE**/**OPENROWSET**/OPENXML/**OPENJSON**/OPENQUERY

스칼라(Scalar) 함수

Configuration/Conversion/**Cursor**/**DateTime**/**JSON/**Logical/Metadata/Security/System Fn/Text&Image

트리거(Trigger) 함수

스칼라 함수가 트리거를 정의하는데 사용된다. 데이터가 변경되었는지 확인한다.

COLUMNS\_UPDATED/EVENTDATA/TRIGGER\_NESTLEVEL/UPDATE

암호화(cryptographic) 함수

<https://docs.microsoft.com/ko-kr/sql/t-sql/functions/cryptographic-functions-transact-sql>

ENCRYPTBYKEY/ENCRYPTBYPASSPHRASE/KEY\_ID/DECRYPTBYKEYAUTOASYMKEY/SYMKEYPROPERTY/DECRYPTBYKEY/DECRYPTBYPASSPHRASE/KEY\_GUID/KEY\_NAME

비대칭암호화 함수

서명(Signing, signature) 함수

자동키 처리 암호해독

암호화 해시

인증서 복사

Date & Time 함수

|  |
| --- |
| -- Date and Time Funcions  DATEADD(\_datepart, number, \_date)  DATEDIFF(\_datepart, startdate, enddate)  DATENAME(\_datepart, \_date)  DATEPART(\_datepart, \_date)  GETDATE();  SELECT GETDATE() -- result : '2018-02-28 오전 9:06:14'  GETUTCDATE()  -- '2018-05-01 01:02:33', '2018/05/01 21:30'The Favorites folder is provided for convenient access to a list of  SELECT DAY('2018-05-01 01:02:33.005') -- result 1  SELECT MONTH('2018-05-01 01:02:33') -- resut: 5  SELECT YEAR('2018/05/01 21:30') -- result 2018 |

특정 월의 첫날과 마지막 날 가져오기

|  |  |
| --- | --- |
| 첫날 | 마지막날 |
| - 지난달 1일  SELECT DATEADD(mm, DATEDIFF(mm, **0**, GETDATE()) - **1**, **0**) | -- 지난달 마지막 일  SELECT DATEADD(dd, -**1**, DATEADD(mm, DATEDIFF(mm,**0**, getdate()), **0**)) |
| -- 이번달 첫날  SELECT DATEADD(mm, DATEDIFF(mm, **0**, GETDATE()) , **0**) | --이번달 마지막날  SELECT DATEADD(dd, -**1**, DATEADD(mm, DATEDIFF(mm,**0**, getdate())+**1**, **0**)) |
| -- 다음달 첫날  SELECT DATEADD(mm, DATEDIFF(mm, **0**, GETDATE()) +**1**, **0**) | -- 다음달 마지막날  SELECT DATEADD(dd, -**1**, DATEADD(mm, DATEDIFF(mm,**0**, getdate())+**2**, **0** ) ) |

현재 시스템의 날짜 및 시간 가져오기

|  |
| --- |
| SELECT SYSDATETIME()  ,SYSDATETIMEOFFSET()  ,SYSUTCDATETIME()  ,CURRENT\_TIMESTAMP  ,GETDATE()  ,GETUTCDATE(); |

|  |
| --- |
| SYSDATETIME() 2007-04-30 13:10:02.0474381  SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00  SYSUTCDATETIME() 2007-04-30 20:10:02.0474381  CURRENT\_TIMESTAMP 2007-04-30 13:10:02.047  GETDATE() 2007-04-30 13:10:02.047  GETUTCDATE() 2007-04-30 20:10:02.047 |

현재 시스템의 날짜 가져오기

|  |
| --- |
| SELECT CONVERT (date, SYSDATETIME())  ,CONVERT (date, SYSDATETIMEOFFSET())  ,CONVERT (date, SYSUTCDATETIME())  ,CONVERT (date, CURRENT\_TIMESTAMP)  ,CONVERT (date, GETDATE())  ,CONVERT (date, GETUTCDATE()); |

|  |
| --- |
| SYSDATETIME() 2007-05-03  SYSDATETIMEOFFSET() 2007-05-03  SYSUTCDATETIME() 2007-05-04  CURRENT\_TIMESTAMP 2007-05-03  GETDATE() 2007-05-03  GETUTCDATE() 2007-05-04 |

현재 시스템의 시간 가져오기

|  |
| --- |
| SELECT CONVERT (time, SYSDATETIME())  ,CONVERT (time, SYSDATETIMEOFFSET())  ,CONVERT (time, SYSUTCDATETIME())  ,CONVERT (time, CURRENT\_TIMESTAMP)  ,CONVERT (time, GETDATE())  ,CONVERT (time, GETUTCDATE()); |

|  |
| --- |
| SYSDATETIME() 13:18:45.3490361  SYSDATETIMEOFFSET()13:18:45.3490361  SYSUTCDATETIME() 20:18:45.3490361  CURRENT\_TIMESTAMP 13:18:45.3470000  GETDATE() 13:18:45.3470000  GETUTCDATE() 20:18:45.3470000 |

집계함수 Aggregate: sum/count/avg

|  |
| --- |
| --Aggregate Functions  AVG(expression) -- computes the average of a column, ignoring any NULL values  AVG(ALL | DISTINCT expression)    SUM(expression) -- computes the sum of values for numeric columns only. ignoring NULL values.  SUM(ALL | DISTINCT expression)    COUNT(expression)  COUNT\_BIG(expression)  COUNT\_BIG(ALL | DISTINCT expression)    MAX(expression) -- finds the maximum value in a column  MAX(ALL | DISTINCT expression)  MIN(expression)    STDEV(expression) -- standard deviation of values in the column  VAR(expression) -- statistical variance in a column |

집계함수는 SUM, AVG, MIN, MAX 등 다양한 함수가 존재한다.

Syntax Convention

|  |  |
| --- | --- |
| Convention | Used for |
| UPPERCASE | Transact-SQL keywords |
| italic | User-supplied parameters of Transact-SQL syntax |
| bold | Database names, table names, column names, index names, stored procedures, utilities, data type names, and text that must be typed exactly |
| underline | Indicates the default value applied when the clause that contains the underlined value is omitted from the statement. |
| |  (vertical bar) | Separates syntax items enclosed in brackets or braces. You can use only one of the items. |
|  |  |
|  |  |
| [ ] (brackets) | Optional syntax items. |
| { } (braces) | Required syntax items. |
| [...n] | Indicates the preceding item can be repeated n number of times. The occurrences are separated by blanks. |
| [,...n] | Indicates the preceding item can be repeated n number of times. The occurrences are separated by commas. |
| ;  (semicolon) | Transact-SQL statement terminator. Although the semicolon is not required for most statements in this version of SQL Server, it will be required in a future version. |
| <label> ::= | he name for a block of syntax. This convention is used to group and label sections of lengthy syntax or a unit of syntax that can be used in more than one location within a statement. Each location in which the block of syntax can be used is indicated with the label enclosed in chevrons: <label>. |

USE

USE { database\_name }

GO

SQL Server utilities interpret GO as a signal that they should send the current batch of Transact-SQL statements to an instance of SQL Server. The current batch of statements is composed of all statements entered since the last GO, or since the start of the ad hoc session or script if this is the first GO.

[T-SQL 프로시져 ]

DECLARE 변수/테이블/커서 선언

DECLARE @변수명 선언

DECLARE @변수명 데이터타입 [,@변수명 데이터타입];

|  |
| --- |
| DECLARE @oppName **nvarchar**(**200**), @accName **nvarchar**(**200**), @ownerName **nvarchar**(**200**);  DECLARE @typeName **nvarchar**(**100**), @sourceNmae **nvarchar**(**100**), @stageName **nvarchar**(**100**); |

SET @변수명 = {값}

|  |
| --- |
| **SET** @keyword = 'parameter' -- 사용자 입력값  **SET** @dept\_id = **(** select id\_tbl\_dept FROM TBL\_DEPT where nm\_dept\_name = @dept\_name **)** |

활용사례: LIKE 검색에서 스트링 합하기

|  |
| --- |
| WHERE text\_body\_topic like **'%'** **+** @keyword **+** **'%'**  OR nm\_headline\_name like '%' + @keyword + '%'  OR nm\_summary like '%' + @keyword + '%' |

T-SQL을 통한 SQL실행

|  |
| --- |
| DECLARE @parent **smallint**  DECLARE @nm\_code\_resource\_detail **nvarchar**(MAX)  SET @parent = (  SELECT id\_code\_resource\_detail FROM CODE\_RESOURCE\_DETAIL  WHERE nm\_code\_resource\_detail = 'Arts, Music, Language, Linguistics, Literature, Recreation, Entertainment, Architecture, Photography, Cinematography'  )  SET @nm\_code\_resource\_detail = 'Visual and Performing Arts'      --SELECT \* FROM CODE\_RESOURCE\_DETAIL WHERE id\_code\_resource\_detail = @parent    INSERT INTO CODE\_RESOURCE\_DETAIL (  id\_code\_resource  ,nm\_code\_resource\_detail  ,tx\_comment  , id\_code\_resource\_detail\_parent  ) VALUES (  NULL  ,@nm\_code\_resource\_detail  , @nm\_code\_resource\_detail  ,@parent  ) |

Syntax detail: CURSOR Variables

After a cursor name is associated with a cursor, the name cannot be used for another cursor of the same scope (GLOBAL or LOCAL) until this cursor has been deallocated.

A cursor variable is associated with a cursor using one of two methods:

|  |
| --- |
| DECLARE abc SCROLL CURSOR FOR  SELECT \* FROM Person.Person; |

* By name using a SET statement that sets a cursor to a cursor variable.

|  |
| --- |
| DECLARE @MyCrsrRef CURSOR;  SET @MyCrsrRef = abc; |

* A cursor can also be created and associated with a variable without having a cursor name defined.

|  |
| --- |
| **DECLARE** @MyCursor **CURSOR**;  **SET** @MyCursor = **CURSOR** **LOCAL SCROLL FOR**  SELECT \* FROM Person.Person; |

A DEALLOCATE @*cursor\_variable\_name* statement removes only the reference of the named variable to the cursor. The variable is not deallocated until it goes out of scope at the end of the batch, stored procedure, or trigger. After a DEALLOCATE @*cursor\_variable\_name* statement, the variable can be associated with another cursor using the SET statement.

|  |
| --- |
| DECLARE @MyCursor CURSOR;  SET @MyCursor = CURSOR LOCAL SCROLL FOR  SELECT \* FROM Sales.SalesPerson;  DEALLOCATE @MyCursor;  SET @MyCursor = CURSOR LOCAL SCROLL FOR  SELECT \* FROM Sales.SalesTerritory;  GO |

A cursor variable does not have to be explicitly deallocated. The variable is implicitly deallocated when it goes out of scope.

DECLARE CURSOR : 레코드 단위 처리

SQL서버 쿼리는 완전한 결과 집합(result set)으로 보여주는데, 경우에 따라 한번에 하나의 행(row)단위로 처리해야 할 때가 필요하다. 그때는 결과집합에 커서를 열고, 결과집합을 한번에 하나의 행씩 처리해야 한다. 커서를 변수나 인자에 cursor data type으로 할당할 수 있다.

참고: <https://docs.microsoft.com/en-us/sql/t-sql/functions/cursor-functions-transact-sql>

DECLARE CURSOR FOR 커서 선언

|  |
| --- |
| **--커서에서 값을 받을 변수 선언**  **DECLARE** [cursor\_name] **CURSOR** **FOR** -- Result Set  SELECT arg1, arg2, … FROM Purchasing.Vendor  **;**  **OPEN** [cursor\_name] ;  **FETCH NEXT FROM** [cursor\_name] **INTO** [arg1, arg1, … in result set] **;** -- select 결과값을 arg1, arg2에 대입한다  --WHILE문 안에 FETCH NEXT FROM을 넣어야 한다.  **WHILE @@FETCH\_STATUS= 0**  **BEGIN**  **FETCH NEXT FROM** [cursor\_name] **INTO** [arg1, arg1, … in result set];  **END**  **CLOSE** [cursor\_name];  **DEALLOCATE** [cursor\_name]; |

CURSOR의 생존범위

how cursors persist until the last name or until the variable referencing them has been deallocated.

|  |
| --- |
| -- Create and open a global named cursor that  -- is visible outside the batch.  DECLARE abc CURSOR GLOBAL SCROLL FOR  SELECT \* FROM Sales.SalesPerson;  OPEN abc;  GO  -- Reference the named cursor with a cursor variable.  DECLARE @MyCrsrRef1 CURSOR;  SET @MyCrsrRef1 = abc;  -- Now deallocate the cursor reference.  DEALLOCATE @MyCrsrRef1;  -- Cursor abc still exists.  FETCH NEXT FROM abc;  GO  -- Reference the named cursor again.  DECLARE @MyCrsrRef2 CURSOR;  SET @MyCrsrRef2 = abc;  -- Now deallocate cursor name abc.  DEALLOCATE abc;  -- Cursor still exists, referenced by @MyCrsrRef2.  FETCH NEXT FROM @MyCrsrRef2;  -- Cursor finally is deallocated when last referencing  -- variable goes out of scope at the end of the batch.  GO  -- Create an unnamed cursor.  DECLARE @MyCursor CURSOR;  SET @MyCursor = CURSOR LOCAL SCROLL FOR  SELECT \* FROM Sales.SalesTerritory;  -- The following statement deallocates the cursor  -- because no other variables reference it.  DEALLOCATE @MyCursor;  GO |

중첩 커서 사용 및 보고서 출력

|  |
| --- |
| SET NOCOUNT ON;  DECLARE @vendor\_id int, @vendor\_name nvarchar(50),  @message varchar(80), @product nvarchar(50);  PRINT '-------- Vendor Products Report --------';  **DECLARE** vendor\_cursor **CURSOR FOR**  SELECT VendorID, Name  FROM Purchasing.Vendor  WHERE PreferredVendorStatus = 1  ORDER BY VendorID;  **OPEN** vendor\_cursor ;  **FETCH NEXT FROM** vendor\_cursor  **INTO** @vendor\_id, @vendor\_name ;  --WHILE문 안에 FETCH NEXT FROM을 넣어야 한다.  **WHILE @@FETCH\_STATUS = 0**  **BEGIN**  PRINT ' '  SELECT @message = '----- Products From Vendor: ' +  @vendor\_name  PRINT @message  -- Declare an inner cursor based  -- on vendor\_id from the outer cursor.  **DECLARE** product\_cursor **CURSOR FOR**  SELECT v.Name  FROM Purchasing.ProductVendor pv, Production.Product v  WHERE pv.ProductID = v.ProductID AND  pv.VendorID = @vendor\_id -- Variable value from the outer cursor  **OPEN** product\_cursor ;  **FETCH NEXT FROM** product\_cursor  **INTO** @product ;  IF @@FETCH\_STATUS <> 0  PRINT ' <<None>>'  **WHILE @@FETCH\_STATUS = 0**  **BEGIN**  SELECT @message = ' ' + @product  PRINT @message  **FETCH NEXT FROM** product\_cursor **INTO @product**  **END**  CLOSE product\_cursor  DEALLOCATE product\_cursor  -- Get the next vendor.  **FETCH NEXT FROM** vendor\_cursor  **INTO** @vendor\_id, @vendor\_name  **END**  **CLOSE** vendor\_cursor;  **DEALLOCATE** vendor\_cursor; |

PRINT 메시지 출력

|  |
| --- |
| ----------------------------------------------------------------------------------------------------------------------------------------------  -- BM으로 받은 데이타를 프로젝트 테이블에 추가하거나 갱신하기  ----------------------------------------------------------------------------------------------------------------------------------------------  -- 엑셀자료를 하나씩 읽어들이면서 관련된 FK 값을 부여해야 한다. 이럴경우는 행단위 작업인 Cursor를 사용할 수 밖에 없다.  -- id\_projects\_type smallint 부여  -- id\_code\_status\_project smallint FK부여  -- id\_code\_resource\_currency smallint 부여  DECLARE @idPrjTypeCode **smallint**, @idPrjStatus **smallint**, @idPrjCurr **smallint** ;  DECLARE @nmPrjTypeName **varchar**(**100**), @nmPrjOrder **varchar**(**100**), @nmPrjStatus **varchar**(**100**), @nmPrjCurrency **varchar**(**100**);  DECLARE Orders\_Cursor **CURSOR** FOR  SELECT p.id\_code\_project\_detail, p.nm\_code\_project\_detail  , o.nm\_projects\_code, o.nm\_status\_project, o.nm\_currency\_unit  FROM CODE\_PROJECT\_DETAIL p  RIGHT JOIN BM\_ORDERS o  ON p.nm\_code\_project\_detail = o.nm\_projects\_type;  OPEN Orders\_Cursor;  FETCH NEXT FROM Orders\_Cursor  INTO @idPrjTypeCode, @nmPrjTypeName, @nmPrjOrder , @nmPrjStatus, @nmPrjCurrency;  IF **@@FETCH\_STATUS** <> **0**  **PRINT ' <NONE> NO FETCH ROWS ' + @nmPrjTypeName;**  WHILE **@@FETCH\_STATUS** = **0**  BEGIN    SET @idPrjStatus = (SELECT id\_code\_status  FROM CODE\_STATUS  WHERE sub\_status\_type = N'PROJ' AND nm\_status\_name = @nmPrjStatus) ;  SET @idPrjCurr = (SELECT r.id\_code\_resource\_detail  --, id\_code\_resource, nm\_code\_resource\_detail, tx\_comment  FROM CODE\_RESOURCE\_DETAIL r  WHERE r.tx\_comment =@nmPrjCurrency AND r.id\_code\_resource = (  SELECT id\_code\_resource FROM CODE\_RESOURCE WHERE nm\_code\_resource='화폐단위' ) );    SELECT @idPrjTypeCode AS PRJ\_TYPE, @nmPrjTypeName AS PRJ\_TYPENAME, @nmPrjOrder AS PRJ\_ORDER  , @nmPrjStatus AS STATUS, @idPrjStatus AS STATUS\_ID  , @nmPrjCurrency AS CURRENCY, @idPrjCurr AS CURR\_ID ;    UPDATE BM\_ORDERS  SET id\_projects\_type = @idPrjTypeCode  , id\_code\_status\_project =@idPrjStatus  , id\_code\_resource\_currency = @idPrjCurr  WHERE nm\_projects\_code = @nmPrjOrder ;    FETCH NEXT FROM Orders\_Cursor  INTO @idPrjTypeCode, @nmPrjTypeName, @nmPrjOrder , @nmPrjStatus, @nmPrjCurrency;  END  CLOSE Orders\_Cursor;  DEALLOCATE Orders\_Cursor; |

|  |
| --- |
|  |

Syntax detal: DECLARE CURSOR

Defines the attributes of a Transact-SQL server cursor, such as its scrolling behavior and the query used to build the result set on which the cursor operates

|  |
| --- |
| **DECLARE** cursor\_name **CURSOR** [ LOCAL | GLOBAL ]  [ FORWARD\_ONLY | SCROLL ]  [ STATIC | KEYSET | DYNAMIC | FAST\_FORWARD ]  [ READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC ]  [ TYPE\_WARNING ]  **FOR** **select**\_statement  [ FOR **UPDATE** [ OF column\_name [ ,...n ] ] ]  [;] |

참고: <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/declare-cursor-transact-sql>

SCROLL

Specifies that all fetch options (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) are available. If SCROLL is not specified in an ISO DECLARE CURSOR, NEXT is the only fetch option supported. SCROLL cannot be specified if FAST\_FORWARD is also specified.

UPDATE [OF column\_name [,...n]]

Defines updatable columns within the cursor. If OF column\_name [,...n] is specified, only the columns listed allow modifications. If UPDATE is specified without a column list, all columns can be updated.

LOCAL

Specifies that the scope of the cursor is local to the batch, stored procedure, or trigger in which the cursor was created. The cursor name is only valid within this scope.

Syntax detal: OPEN

Opens a Transact-SQL server cursor and populates the cursor by executing the Transact-SQL statement specified on the DECLARE CURSOR or SET cursor\_variable statement.

|  |
| --- |
| OPEN { { [ GLOBAL ] cursor\_name } | cursor\_variable\_name } |

|  |
| --- |
| DECLARE Employee\_Cursor CURSOR FOR  SELECT LastName, FirstName  FROM AdventureWorks2012.HumanResources.vEmployee  WHERE LastName like 'B%';  OPEN Employee\_Cursor;  FETCH NEXT FROM Employee\_Cursor;  WHILE @@FETCH\_STATUS = 0  BEGIN  FETCH NEXT FROM Employee\_Cursor  END;  CLOSE Employee\_Cursor;  DEALLOCATE Employee\_Cursor; |

Syntax detal: FETCH

Retrieves a specific row from a Transact-SQL server cursor.

|  |
| --- |
| FETCH  [ [ **NEXT** | PRIOR | FIRST | LAST  | ABSOLUTE { n | @nvar }  | RELATIVE { n | @nvar }  ]  FROM  ]  { { [ GLOBAL ] cursor\_name } | @cursor\_variable\_name }  [ **INTO** @variable\_name [ ,...n ] ] |

Syntax detal: @@FETCH\_STATUS

Returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | Return value | Description | | --- | --- | | 0 | The FETCH statement was successful. | | -1 | The FETCH statement failed or the row was beyond the result set. | | -2 | The row fetched is missing. | | -9 | The cursor is not performing a fetch operation. | |

Syntax detal: @@CURSOR\_ROWS

eturns the number of qualifying rows currently in the last cursor opened on the connection. To improve performance, SQL Server can populate large keyset and static cursors asynchronously. @@CURSOR\_ROWS can be called to determine that the number of the rows that qualify for a cursor are retrieved at the time.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | Return value | Description | | --- | --- | | -*m* | The cursor is populated asynchronously. The value returned (-*m*) is the number of rows currently in the keyset. | | -1 | The cursor is dynamic. Because dynamic cursors reflect all changes, the number of rows that qualify for the cursor is constantly changing. It can never be definitely stated that all qualified rows have been retrieved. | | 0 | No cursors have been opened, no rows qualified for the last opened cursor, or the last-opened cursor is closed or deallocated. | | *n* | The cursor is fully populated. The value returned (*n*) is the total number of rows in the cursor. | |

Syntax detal: NEXT

Returns the result row immediately following the current row and increments the current row to the row returned. If FETCH NEXT is the first fetch against a cursor, it returns the first row in the result set. NEXT is the default cursor fetch option.

Syntax detal: PRIOR

Returns the result row immediately preceding the current row, and decrements the current row to the row returned. If FETCH PRIOR is the first fetch against a cursor, no row is returned and the cursor is left positioned before the first row.

@cursor\_variable\_name

Is the name of a cursor variable referencing the open cursor from which the fetch should be made.

Syntax detal: INTO

INTO @variable\_name[ ,...n]

Allows data from the columns of a fetch to be placed into local variables. Each variable in the list, from left to right, is associated with the corresponding column in the cursor result set. The data type of each variable must either match or be a supported implicit conversion of the data type of the corresponding result set column. The number of variables must match the number of columns in the cursor select list.

Syntax detal: CLOSE

Closes an open cursor by releasing the current result set and freeing any cursor locks held on the rows on which the cursor is positioned. CLOSE leaves the data structures available for reopening, but fetches and positioned updates are not allowed until the cursor is reopened. CLOSE must be issued on an open cursor; CLOSE is not allowed on cursors that have only been declared or are already closed.

|  |
| --- |
| CLOSE { { [ GLOBAL ] cursor\_name } | cursor\_variable\_name } |

Syntax detal: DEALLOCATE

Removes a cursor reference. When the last cursor reference is deallocated, the data structures comprising the cursor are released by Microsoft SQL Server.

|  |
| --- |
| DEALLOCATE { { [ GLOBAL ] cursor\_name } | @cursor\_variable\_name } |

DECLARE @테이블명

DECLARE @테이블명 TABLE(컬럼명 데이터타입(사이즈))

|  |
| --- |
| DECLARE @tbl TABLE(Column1 **VARCHAR**(**100**))  INSERT INTO @tbl  SELECT '1/1' UNION ALL  SELECT '1/20' UNION ALL  SELECT '1/2' |

WHILE 루프

|  |
| --- |
| WHILE Boolean\_expression  { sql\_statement | statement\_block | **BREAK** | **CONTINUE** } |

|  |
| --- |
| **WHILE** (SELECT AVG(ListPrice) FROM Production.Product) < $300  **BEGIN**  UPDATE Production.Product  SET ListPrice = ListPrice \* 2  SELECT MAX(ListPrice) FROM Production.Product  IF (SELECT MAX(ListPrice) FROM Production.Product) > $500  **BREAK**  ELSE  **CONTINUE**  **END**  PRINT 'Too much for the market to bear'; |

Using WHILE in a cursor

|  |
| --- |
| **DECLARE** Employee\_Cursor **CURSOR FOR**  SELECT EmployeeID, Title  FROM AdventureWorks2012.HumanResources.Employee  WHERE JobTitle = 'Marketing Specialist';  **OPEN** Employee\_Cursor**;**  **FETCH NEXT** **FROM** Employee\_Cursor;  WHILE @@FETCH\_STATUS = 0  BEGIN  **FETCH NEXT FROM** Employee\_Cursor;  END;  **CLOSE** Employee\_Cursor;  **DEALLOCATE** Employee\_Cursor; |

CASE … WHEN … THEN END

문법

|  |
| --- |
| **CASE** {조건}  **WHEN** {값1} **THEN** {값1}  **WHEN** {값2} **THEN** {값2}  **ELSE** {값3}  **END**  SELECT  nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , SUM( real\_invoice\_amount) AS 'Month Amount'  , CASE nm\_currency\_unit  WHEN 'EUR' THEN SUM( real\_invoice\_amount \* **1300** )  WHEN 'USD' THEN SUM( real\_invoice\_amount\* **1100** )  WHEN 'CNY' THEN SUM( real\_invoice\_amount\* **165** )  ELSE SUM( real\_invoice\_amount)  END AS 'KOR Amount'  , ym\_invoice\_issued\_month AS 'Issued Month'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %'  GROUP BY nm\_customers\_name,nm\_currency\_unit , ym\_invoice\_issued\_month |

|  |
| --- |
| **CASE**  **WHEN** {조건=값1} **THEN** {값1}  **WHEN** {조건=값22} **THEN** {값2}  **ELSE** {값3}  **END**  SELECT  nm\_customers\_name AS Customer  , nm\_currency\_unit AS Currency  , SUM( real\_invoice\_amount) AS 'Month Amount'  , CASE  WHEN nm\_currency\_unit = 'EUR' THEN SUM( real\_invoice\_amount \* **1300** )  WHEN nm\_currency\_unit = 'USD' THEN SUM( real\_invoice\_amount\* **1100** )  WHEN nm\_currency\_unit = 'CNY' THEN SUM( real\_invoice\_amount\* **165** )  ELSE SUM( real\_invoice\_amount)  END AS 'KOR Amount'  , ym\_invoice\_issued\_month AS 'Issued Month'  FROM CUSTOMER\_INVOICES\_LIST  WHERE ym\_invoice\_issued\_month like '2017%' AND nm\_customers\_name like 'Samsung %'  GROUP BY nm\_customers\_name,nm\_currency\_unit , ym\_invoice\_issued\_month |

WHERE에서 사용하기

|  |
| --- |
| -- 1 또는 0개  IF NOT EXISTS (SELECT cd\_code\_stage FROM CODE\_STAGE WHERE nm\_code\_stage\_name = @stageName )  SET @cdStage = ( SELECT cd\_code\_stage FROM CODE\_STAGE  WHERE nm\_code\_stage\_name =  CASE @stageName WHEN 'Closed Won' THEN '수주확정'  WHEN 'Proposal' THEN '견적제안'  WHEN 'Negotiation' THEN '견적제안'  WHEN 'Needs Analysis' THEN '니즈분석'  WHEN 'Qualification' THEN '니즈분석'  WHEN 'Closed Lost' THEN '수주실패'  ELSE '수주실패'  END  )  ELSE SET @cdStage = ( SELECT cd\_code\_stage FROM CODE\_STAGE  WHERE nm\_code\_stage\_name = @stageName ); |

INSERT 문에서 사용하기

|  |
| --- |
| INSERT INTO OPPORTUNITY\_LIST (  nm\_opportunity\_name  ,id\_account\_list  ,nm\_account\_name  ,id\_account\_contacts  ,nm\_contacts\_name  ,yn\_loss  ,cd\_code\_lossreason  ,nm\_code\_lossreason\_name  ,tx\_opportunity\_loss\_reason  ) SELECT  nm\_opportunity\_name AS nm\_opportunity\_name -- nm\_opportunity\_name - nvarchar(200)  , id\_account\_list AS id\_account\_list -- id\_account\_list - int  , nm\_account\_name AS nm\_account\_name -- nm\_account\_name - nvarchar(200)  , NULL AS id\_account\_contacts -- id\_account\_contacts - int  ,NULL AS nm\_contacts\_name -- nm\_contacts\_name - nvarchar(200)  ,CASE  WHEN cd\_code\_stage = **3** THEN 'LOSS' -- 수주실패  ELSE 'OTHR' -- 기타  END AS yn\_loss -- yn\_loss - char(4)  , NULL AS cd\_code\_lossreason -- cd\_code\_lossreason - smallint  ,NULL AS nm\_code\_lossreason\_name -- nm\_code\_lossreason\_name - nvarchar(100)  ,NULL AS tx\_opportunity\_loss\_reason -- tx\_opportunity\_loss\_reason - ntext  FROM EXCEL\_BATCH |

SELECT에서 사용하기

|  |
| --- |
| -- 프로젝트 스탭 조회하기  -- PROJECTS\_LIST, PROJECT\_LANGS\_TASKS  -- 대문자 '872'로 시작하는 프로젝트명을 가진 프로젝트 스탭 및 외부 리소스를 검색한다.  -- 외부 리소스의 경우 배정 금액을 함께 보여준다.  SELECT p.nm\_projects\_name AS 프로젝트명 , p.nm\_pm\_name AS PM명  , **CASE WHEN** t.sub\_taskworker\_name='RES' **THEN** '외부리소스'  **WHEN** t.sub\_taskworker\_name='EMP' **THEN** '직원'  **END** AS 직원구분  , t.nm\_taskworker\_name AS 작업자명  , t.id\_taskworker\_id  , t.nm\_currency\_name, t.real\_workprice\_system  , t.nm\_code\_resource\_detail, t.real\_workprice\_fixed  FROM PROJECTS\_LIST p  INNER JOIN PROJECT\_LANGS\_TASKS t  ON p.id\_projects\_list = t.id\_projects\_list  WHERE p.nm\_projects\_name LIKE '%872%' |

UPDATE문에서 사용하기

|  |
| --- |
| update TEMP\_STD set style\_id = (  **case when** style\_nm = 'H1' **then**  '002'  **when** style\_nm = 'H2' **then** '003'  when style\_nm = 'H3' then '004'  when style\_nm = is null then '001'  else 'error'  end  ) |

|  |
| --- |
| --Update Join  UPDATE t  SET  t.status\_cd = (  CASE WHEN a.confirm\_yn ='N' THEN 'S903' --미정  WHEN a.confirm\_yn is null THEN 'S901' -- 승인  ELSE 'S902' -- 거절  END  )  FROM TEMP\_STD t  INNER JOIN  ( select chapter\_id, sntc\_std\_id, lang\_cd, sntc\_std\_\_nm, confirm\_yn from TEMP\_STD  ) a  ON t.chapter\_id = a.chapter\_id and t.sntc\_std\_id = a.sntc\_std\_id and t.lang\_cd = a.lang\_cd |

IF 조건 처리

IF [ELSE]

|  |
| --- |
| **IF** Boolean\_expression  {sql\_statement | statement\_block}  [ **ELSE**  {sql\_statement | statement\_block} ]  **;**  Boolean\_expression은 TRUE 또는 FALSE값을 반환하는 식이며 SELECT문이라면 괄호(‘(‘, ‘)’)로 묶어야 한다.  statement\_block의 경우 두개 이상의 문장을 실행해야 하기에 BEGIN END로 문장을 그룹핑해야 한다. |

IF COUNT

|  |
| --- |
| DECLARE @AvgWeight decimal(8,2), @BikeCount int  **IF** **(**SELECT **COUNT**(\*) FROM Product WHERE Name LIKE 'Touring-3000%'**)** > 5  BEGIN  SET @BikeCount = (SELECT COUNT(\*) FROM Product WHERE Name LIKE 'Touring-3000%');  SET @AvgWeight = (SELECT AVG(Weight) FROM Product WHERE Name LIKE 'Touring-3000%');  PRINT 'There are ' + CAST(@BikeCount AS varchar(3)) + ' Touring-3000 bikes.'  PRINT 'The average weight of the top 5 Touring-3000 bikes is ' + CAST(@AvgWeight AS varchar(8)) + '.';  END  **ELSE**  BEGIN  SET @AvgWeight = (SELECT AVG(Weight) FROM Product  WHERE Name LIKE 'Touring-3000%' );  PRINT 'Average weight of the Touring-3000 bikes is ' + CAST(@AvgWeight AS varchar(8)) + '.' ;  END  **;**  GO |

IF EXISTS()

유의사항:

IF EXISTS (Select count(\*) From ABC)와 같이 Select문에 count함수를 사용하면 조건에 해당하는 결과가 없어도 **0(zero)를 리턴하므로 항상 참(True)가 실행된다**. 따라서 IF EXISTS를 사용하여 존재하는 지를 알려면 **COUNT함수를 사용하지 않아야 한다.**

IF EXISTS (SELECT ABC.col1 From ABC)

|  |
| --- |
| DECLARE @intYNResource **smallint**;  DECLARE @nmTaskWorker **varchar**(**80**);    SET @nmTaskWorker = '';    IF LEN(@nmTaskWorker) > **1**  BEGIN  IF EXISTS (SELECT id\_emp FROM TBL\_COMPANY\_EMP WHERE nm\_emp\_name = @nmTaskWorker)  print 'EMP-->' + @nmTaskWorker ;  ELSE  BEGIN  IF EXISTS (SELECT id\_tbl\_resource\_list FROM TBL\_RESOURCE\_LIST WHERE nm\_name\_venorperson = @nmTaskWorker)  print 'RES --> ' + @nmTaskWorker ;  ELSE print 'New Vendor'  END  END  ELSE print 'null'; |

IF NOT EXISTS()

|  |
| --- |
| DECLARE @nmTaskWorker **varchar**(**80**) ; --작업자명이 없는 경우가 많다.  DECLARE @intYNResource **smallint**;    DECLARE Worker\_Cursor **CURSOR** FOR  SELECT j.nm\_taskworker\_name  FROM BM\_JOBS j  INNER JOIN PROJECTS\_LIST p -- INNER JOIN  ON j.nm\_projects\_code = p.nm\_projects\_code  ;  OPEN Worker\_Cursor;  FETCH NEXT FROM Worker\_Cursor  INTO @nmTaskWorker ;  IF **@@FETCH\_STATUS** <> **0**  PRINT ' <NONE> NO FETCH ROWS '  WHILE **@@FETCH\_STATUS** = **0**  BEGIN  IF LEN(@nmTaskWorker) > **1**  BEGIN --벤더와 직원 테이블에 없는 데이타 찾기  IF NOT EXISTS (SELECT id\_tbl\_resource\_list FROM TBL\_RESOURCE\_LIST WHERE nm\_name\_venorperson = @nmTaskWorker)  BEGIN  IF NOT EXISTS (SELECT id\_emp FROM TBL\_COMPANY\_EMP WHERE nm\_emp\_name = @nmTaskWorker)  BEGIN  PRINT 'Unregistered vendor or freelancer:'+ @nmTaskWorker ;  END  END  END    FETCH NEXT FROM Worker\_Cursor INTO @nmTaskWorker ;  END  CLOSE Worker\_Cursor;  DEALLOCATE Worker\_Cursor; |

|  |
| --- |
| DECLARE @resourceID **int**  DECLARE @searchText **varchar**(MAX)  SET @searchText = '%test%'  -- 삭제할 리소스를 이름으로 찾기  SET @resourceID = (  SELECT id\_tbl\_resource\_list FROM TBL\_RESOURCE\_LIST WHERE nm\_name\_venorperson LIKE @searchText  )  -- 연락처 정보가 존재하는 지  IF **EXISTS** ( SELECT \* FROM RESOURCE\_CONTACTS WHERE id\_tbl\_resource\_list = @resourceID )  DELETE FROM RESOURCE\_CONTACTS WHERE id\_tbl\_resource\_list = @resourceID  ELSE  SELECT \* FROM RESOURCE\_CONTACTS WHERE at\_contact\_mail LIKE @searchText |

Break/Continue/Return/Goto로 탈출하기

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BREAK** | **CONTINUE** | **RETURN** | **GOTO** |
|  | IF 또는 WHILE 루프에서  바로 탈출 | 이후 문장은 실행하지 않음.  WHILE로 돌아가 계속 루프 | 프로시져, 배치, 블록에서  조건없이 바로 탈출. | 지정한 라벨로 점프 |
|  | 조건 또는 루프에서 탈출 | WHILE 계속 진행 | 프로그램 종료 | 점프 |

BREAK

IF 또는 WHILE 내부 루프에서 바로 탈출 할 때 사용하는 키워드 : BREAK

|  |
| --- |
| -- Uses AdventureWorks  WHILE ((SELECT AVG(ListPrice) FROM dbo.DimProduct) < $300)  BEGIN  UPDATE DimProduct  SET ListPrice = ListPrice \* 2;  IF ((SELECT MAX(ListPrice) FROM dbo.DimProduct) > $500)  BREAK;  END |

CONTINUE

WHILE 루프에서 **CONTINUE 이후의 문장은 실행하지 않고** 무시하면서 **WHILE루프를 다시** 시작한다.

RETURN

Exits unconditionally from a query or procedure. RETURN is immediate and complete and can be used at any point to exit from a procedure, batch, or statement block.

Statements that follow RETURN are not executed.

|  |
| --- |
| RETURN [ integer\_expression ] |

|  |
| --- |
| WHILE **@@FETCH\_STATUS** = **0**  BEGIN  SET @intEmpCount = (SELECT COUNT(id\_emp) FROM TBL\_COMPANY\_EMP WHERE nm\_emp\_name = @nmPMname) ;    IF ( @intEmpCount > **1** )  BEGIN  -- -- 여러개 존재  PRINT 'Error Occurred: PM name is more than one ==> ' + @nmPMname ;  **RETURN** -- 이후 문장 처리하지 않고 배치프로그램 나옴  END  ;    IF ( @intEmpCount = **0** )  -- PM명 존재하지 않음...  PRINT 'Error Occurred: PM name is not existed==> ' + @nmPMname ;  ELSE -- PM명 존재  BEGIN  SET @idEmpID = (SELECT id\_emp FROM TBL\_COMPANY\_EMP WHERE nm\_emp\_name = @nmPMname );  END  ;  FETCH NEXT FROM Project\_Cursor  INTO @nmProjCode, @nmPMname ;  END |

GOTO

제어의 흐름을 해당 라벨로 점프하여 수행토록 한다.

라벨 선언은 라벨명 뒤에 콜론( : )을 붙여 선언한다.

{**LABEL\_NAME**} **:**

제어를 특정 라벨로 이동하게 하려면 GOTO 명령어에 라벨명을 적는다.

GOTO { **LABEL\_NAME** } ;

|  |
| --- |
| DECLARE @Counter **int**;  SET @Counter = **1**;  WHILE @Counter < **10**  BEGIN  SELECT @Counter  SET @Counter = @Counter + **1**  IF @Counter = **4** GOTO Branch\_One --Jumps to the first branch.  IF @Counter = **5** GOTO Branch\_Two --This will never execute.  END  Branch\_One :  SELECT 'Jumping To Branch One.'  -- **GOTO** Branch\_Three; --This will prevent Branch\_Two from executing.  Branch\_Two :  SELECT 'Jumping To Branch Two.'  **Branch\_Three:**  SELECT 'Jumping To Branch Three.'; |

1

2

3

Jumping To Branch One.

Jumping To Branch Two.

Jumping To Branch Three.

커멘트 처리

한줄 주석처리는 –( hyphens ) 를 사용하며, 여러 줄에 걸쳐서 주석할 때는 /\* \*/(splash star)을 사용한다.

-- region name --코멘트 입력

-- endregion

Backslash 긴 스트링을 2줄 이상으로 기록

\ breaks a long string constant into two or more lines for readability.

|  |
| --- |
| SELECT 'abc\  def' AS ColumnResult; |

BEGIN… END

Encloses a series of Transact-SQL statements so that a group of Transact-SQL statements can be executed. BEGIN and END are control-of-flow language keywords.

|  |
| --- |
| BEGIN  {sql\_statement | statement\_block}  END |

|  |
| --- |
| BEGIN TRANSACTION;  GO  IF @@TRANCOUNT = 0  **BEGIN**  SELECT FirstName, MiddleName  FROM Person.Person WHERE LastName = 'Adams';  ROLLBACK TRANSACTION;  PRINT N'Rolling back the transaction two times would cause an error.';  **END;**  ROLLBACK TRANSACTION;  PRINT N'Rolled back the transaction.'; |

THROW

Raises an exception and transfers execution to a CATCH block of a TRY…CATCH construct in SQL Server

TRY…CATCH

WAITFOR

DBCC(데이터베이스 콘솔 명령어)

|  |  |
| --- | --- |
| 데이터베이스 | 콘솔 명령 문은 다음과 같은 범주로 분류할 수 있습니다. |
| 유지 관리 | 데이터베이스, 인덱스 또는 파일 그룹에 대한 유지 관리 태스크 |
| 기타 | 추적 플래그 설정이나 메모리에서 DLL 제거 같은 기타 태스크입니다. |
| 알림 | 다양한 정보를 수집하고 표시하는 태스크입니다. |
| 유효성 | 검사 데이터베이스, 테이블, 인덱스, 카탈로그, 파일 그룹 또는 데이터베이스 페이지 할당에 대한 유효성 검사 작업입니다. |

BACKUP DATABASE : DB백업

|  |
| --- |
| BACKUP **DATABASE** [DEVINFRAMUI] TO  DISK = N'D:\Backup\DEVINFRAMUI-FullBackup.20180223.bak' WITH  NOFORMAT,  NOINIT,  NAME = N'DEVINFRAMUI-Full Database Backup',  SKIP,  NOREWIND,  NOUNLOAD,  STATS = **10**  GO |

DBCC CHECKIDENT: Identity 리셋 등

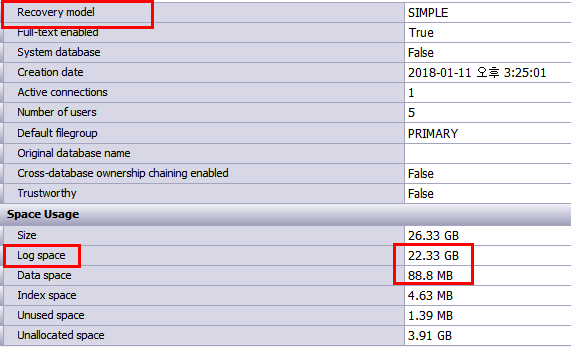
Identity 리셋

|  |
| --- |
| DBCC CHECKIDENT  (  table\_name  [, { NORESEED | { RESEED [, new\_reseed\_value ]}}]  )  [ WITH NO\_INFOMSGS ]  NORESEED 현재 ID 값을 변경하지 않도록 지정합니다  RESEED 현재 ID 값을 변경하도록 지정합니다  new\_reseed\_value ID 열의 현재 값으로 사용할 새 값입니다.  WITH NO\_INFOMSGS 모든 정보 메시지를 표시하지 않습니다. |

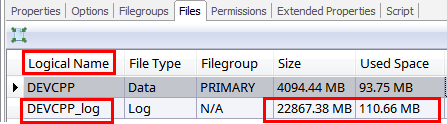
|  |
| --- |
| DELETE FROM [Test\_Table];  --Then the first row will get the identity = 1  DBCC CHECKIDENT (‘[Test\_Table]’, RESEED, **0**) |

DBCC SHRINKFILE: 로그 사이즈 줄이기

급격하게 비정상적으로 커진 로그 사이즈로 인해 전체 DB사이즈(40G)에 영향을 주고 있다. 로그 크기를 줄인다



논리적 파일명



실제파일명



|  |
| --- |
| DBCC SHRINKFILE  (  { file\_name | file\_id } --논리적 파일명을 기입하세요  { [ , EMPTYFILE ]  | [ [ , target\_size ] – 파일사이즈는 M 바이트 단위임  [ , { NOTRUNCATE | TRUNCATEONLY } ] ] --  }  )  [ WITH NO\_INFOMSGS ] |

|  |
| --- |
| USE DEVBMS;  DBCC SHRINKFILE  (  DEVCPP\_log  , **200**  ); |

파일명 또는 파일ID를 조회하여 사용하려면 다음 쿼리를 사용하세요.

|  |
| --- |
| SELECT file\_id, name  FROM sys.database\_files;  GO  DBCC SHRINKFILE (**2**, TRUNCATEONLY); |

##ORACLE

오라클은 ID 자동 증가가 없다?? SEQUENCE

오라클은 테이블내 컬럼ID를 자동으로 증가시키지 않는다. 그래서 해당 컬럼에 사용할 시퀀스(SEQUENCE)를 생성한 후, 데이터를 입력할 때 시컨스를 자동 증가하는 방식으로 적용해야 한다.

시퀀스 조회

|  |
| --- |
| SELECT sequence\_name, min\_value, max\_value, increment\_by, last\_number from **USER\_SEQUENCES** order by sequence\_name |

시퀀스 생성

|  |
| --- |
| CREATE **SEQUENCE** *HSP2021\_S* START WITH 1 INCREMENT BY 1 MAXVALUE 10000 CYCLE NOCACHE; |

문범(Syntax)

|  |
| --- |
| CREATE SEQUENCE *sequence\_name*  [INCREMENT BY n1] -- 생략하면 1로 지정됨  [START WITH n2] -- 생략하면 1로 지정됨  [MAXVALUE n3 or NOMAXVALUE] -- NOMAXVALUE: 최대 10의 27승, 내림 -1 까지  [MINVALUE n4 or NOMINVALUE] -- NOMINVALUE: 오름 1 까지, 내림 -10의 26승까지  [CYCLE or **NOCYCLE**] -- NOCYCLE: 최대값에 도달해도 다시 생성하지 않는다.[DEFAULT]  [**CACHE** or NOCACHE]; -- NOCACHE: 메모리에 할당하여 사용하지 않음 |

시퀀스를 이용한 테이블 입력

|  |
| --- |
| INSERT INTO HSP2021 ( UNIT\_ID, CUST\_ID, CUST\_DEPT\_ID, BILL\_FORM, UNIT\_CODE1, UNIT\_CODE2, BILL\_ITEM,CODE\_YEAR )  SELECT HSP2021**\_S.NEXTVAL**, CUST\_ID, CUST\_DEPT\_ID, BILL\_FORM, UNIT\_CODE1, UNIT\_CODE2, BILL\_ITEM, '2017' FROM HSP2021 WHERE CUST\_ID = 36 AND CUST\_DEPT\_ID = 34 AND BILL\_FORM = 'GL\_MU' AND bill\_code\_year = '2015' ; |

시퀀스 관련 오류

|  |
| --- |
| **SQL 오류 : ORA-02287 시퀀스 번호는 이 위치에 사용할 수 없습니다” "sequence number not allowed here"** |

**ORDER BY (또는 Group by)**로 SELECT하는 경우 시퀀스를 얻지 못한다. 따라서 ORDER BY절을 지우고 SQL문을 실행하면 해결된다.

시퀀스 삭제

|  |
| --- |
| **DROP SEQUENCE** seq\_autoID |

시퀀스의 초기값 변경

|  |
| --- |
| ALTER SEQUENCE *seq\_autoID* START WITH 91770 INCREMENT BY 1 |