<1-1 데이터 모델링의 이해>

모델링 특징

추상화 : 현실세계, 다양한 현상 일정한 양식인 표기법에 의해 표현 (= 모형화, 가설적)
단순화 : 복잡한 현실세계를 약속된 규약 의해 제한된 표기법이나 언어로 표현
명확화 : 누구나 이해하기 쉽게 대상에 대한 애매모호함 제거

모델링의 세 가지 관점

데이터 관점: 업무가 어떤 데이터와 관련이 있는지, 데이터간 관계 무엇인지 (What, Data)
프로세스 관점: 업무가 실제하고 있는 일 무엇인지, 무엇을 해야하는지 (How, Process)
상관 관점 : 업무 처리하는 방법에 따라 데이터 어떻게 영향 받고 있는지 (Interaction)

데이터 모델링 정의

정보시스템 구축 위한 데이터 관점의 업무 분석기법, 현실세계 데이터에 대해 약속된 표기법에 의해 표현하는 과정, 데이터베이스 구축하기 위한 분석, 설계과정

데이터모델링 기능 : 명세화, 구조화, 문서화, 다양한 관점, 상세수준 표현

데이터 모델링의 유의점 : 중복 , 비유연성, 비일관성

데이터 모델링 중요성

파급효과가 크다(Leverage)
복잡한 정보 요구사항의 간결한 표현(Conciseness)
데이터 품질을 유지 (Data Quality)

데이터 모델링의 3단계 진행 : (추상적)개념적 -> 논리적 -> 물리적(구체적)
개념적 데이터 모델링(in계획분석단계): 추상화, 업무중심적, 포괄적, 전사적, EA수립시 사용
논리적 데이터 모델링(in분석단계): KEY, 속성, 관계 표현. 재사용성 높음 (정규화)
물리적 데이터 모델링(in설계단계): 실제 데이터베이스 이식할 수 있도록 성능,저장 등 물리적 성격 고려

데이터 독립성 필요 이유

유지보수 비용증가, 데이터 중복성, 복잡성 증가, 요구사항 대응 저하

ANSI / SPARC 3단계 구조

논리적 독립성(외부단계-개념적단계) : 개념적스키마 변경, 외부스키마 영향 없음. 논리적 구조 변경되어도 응용프로그램 영향 없음
물리적 독립성(개념적단계-내부적단계) : 내부스키마 변경, 외부/개념스키마 영향 없음. 저장장치의 구조변경은 응용프로그램과 개념스키마에 영향없음

좋은 데이터 모델의 요소 : 완정성, 중복배제, 업무규칙, 데이터 재사용, 의사소통, 통합성

데이터 모델링 세가지 요소 : 어떤 것(Thing), 성격(Attrivutes), 관계(Relationships)

엔터티의 개념 : 명사에 해당, 저장이 되기 위한 어떤 것(Thing)

엔터티의 특징 : 업무에서 필요한 정보, 식별이 가능해야 함, 인스턴스의 집합(두개이상), 업무 프로세스에 의해 이용, 속성 포함, 관계 존재

엔터티의 분류
유무형
-유형 : 물리적인 형태, 안정적 (사원, 물품, 강사)
-개념 : 물리적 형태 존재하지 않고 관리해야 할 개념적 정보 (조직, 보험상품)
-사건 : 업무 수행함에 따라 발생됨 (주문, 청구, 미납)
발생시점 (기본->중심-> 행위)
-기본 : 원래 존재하는 정보. 독립적으로 생성 가능. (사원, 부서, 고객)
-중심 : 기본엔터티로부터 발생되고 업무에 있어서 중심적인 역할 (계약, 접수)
-행위 : 두 개 이상의 부모엔터티로부터 발생되고 자주 내용 바뀌거나 데이터 양 증가(주문내역, 계약진행)

속성의 개념 : 업무에서 필요, 의미상 미분리, 인스턴스의 구성요소

인스턴스 - 속성 - 엔터티 의 관계
인스턴스 2개 이상 엔터티, 2개 이상 속성을 엔터티는 가짐, 1:1 (속성 : 속성값)

속성의 분류
기본속성(업무로부터 추출한 모든 속성_원래속성)
설계 속성 (코드성 속성, 1:1치환), 파생 속성(계산된 값)

도메인의 정의 : 각 속성이 가질 수 있는 값의 범위

관계 : 인스턴스의 관계가 페어링 -> 페어링의 집합이 관계(Relation)

관계 표기법 : 관계명(Membership), 관계차수(Cardinality), 관계선택사양(Optionality_필수관계, 선택관계)

관계 체크사항 : 연관규칙 존재하는지, 엔터티 사이 정보 조합이 발생되는지, 관계연결에 대한 규칙이 서술되어있는지, 관계연결 가능하게 하는 동사(Verb)있는지

식별자 분류
대표성 여부(주식별자/보조식별자), 스스로생성여부(내부식별자/외부식별자),
속성의 수 (단일식별자/복합식별자), 대체여부(본질식별자/인조식별자)

주식별자의 특징 : 유일성, 희소성, 불변성, 존재성

식별자/ 비식별자 관계 (상속여부가 Key Point)

식별자 관계 : 부모로부터 받은 식별자를 자신엔터티의 주식별자로 이용 (강한 연결관계, 실선표현)
비식별자 관계 : 부모로부터 속성 받았지만 일반속성으로 사용 (약한 연결관계, 점선 표현)

비식별자 관계 설정 고려사항 : 관계분석 -> 관계의 강/약 분석(약) -> 자식테이블 독립PK필요(필요) -> SQL 복잡도 증가 (비식별자)

식별자 관계설정 고려사항 : 강한관계, 주식별자 PK사용

<1-2 데이터 모델과 성능>

성능 데이터 모델링 고려사항 : 정규화 -> 용량산정 -> 트랙잰션 유형 파악 -> 반정규화 -> 이력모델 조정, PK/FK 조정, 슈퍼/서브타입 조정 -> 성능관점 데이터 모델 검증
(분석/설계 단계에서 성능 데이터 모델링하기)

함수적 종속성 : 데이터들이 어떤 기준값에 의해 종속되는 현상 (기준값을 결정자, 종속되는 값을 종속자) ex) 주민등록번호 - (이름,출생지,주소)

정규화 : 정규화 수행하면 데이터 처리(DML) 성능 향상, 조회는 향상 또는 저하 될수 있음

반정규화 장점 : 일반적으로 조회 성능향상이 됨
반정규화 절차 : 반정규화 대상 조사 -> 다른방법 검토 -> 반정규화 적용

반정규화 기법
-테이블 반정규화
테이블병합(1:1, 1:M, 슈퍼/서브타입)
테이블 분할(수직_칼럼단위,수평_로우단위)
테이블추가(중복,통계,이력,부분)
-칼럼 반정규화
중복칼럼추가, 파생칼럼추가, 이력테이블 칼럼 추가, PK에 의한 칼럼추가,응용시스템 오작동 위한 칼럼추가
-관계 반정규화 : 중복관계 추가, 테이블,칼럼 반정규화와 달리 데이터 무결성 영향 없음

대량 데이터
테이블의 수평, 수직 분할로 모델링 해야 한다
로우체이닝: 로우 길이가 너무 길어서 데이터 블록 하나에 데이터가 모두 저장되지 않고 두 개 이상의 블록에 걸쳐 하나의 로우가 저장되어 있는 형태
로우마이그레이션 : 데이터 블록에서 수정이 발생하면 수정된 데이터를 해당 데이터 블록에서 저장하지 못하고 다른 블록의 빈 공간을 찾아 저장하는 방식

파티셔닝
RANGE PARTITION(범위 나눠서. 년+월 이용하여 12개 파티션 테이블. 대상 테이블이 날짜 또는 숫자값으로 분리가 가능하고 각 영역별로 트랜잭션이 분리가능할 때 적용)

LIST PARTITION(특정값 지정, 고객_서울,고객_인천 지점별 파티션테이블)
HASH PARTITION (hash 조건 따라 해쉬 알고리즘 적용되어 테이블 분리)

슈퍼/서브 타입 모델
슈퍼타입 : 공통부분
서브엔터티 : 공통부분 상속받아 다른 엔터티와 차이가 있는 속성 대해 별도로 구분
변환기준? 데이터 양 & 트랜잭션 유형

슈퍼/서브타입 변환
One to One : 개별테이블, 확장성 우수, 조인성능 나쁨, I/O좋음, 관리 안좋음
Plus Type : 슈퍼서브타입테이블, 확장성 보통, 조인성능 나쁨, I/O좋음, 관리 안좋음
Single Type : 하나의 테이블, 확장성 나쁨, 조인성능 우수, I/O나쁨, 관리좋음

인덱스 특성을 고려한 PK/FK 데이터베이스 성능향상 이해
PK 컬럼 순서 중요, FK컬럼의 인덱스 컬럼화 필요

분산 데이터베이터 투명성
분할(단편화), 위치, 지역사상, 중복, 장애, 병행 투명성

분산 데이터베이스의 장단점
장점 : 지역자치성, 점증적 시스템 용량확장, 신뢰성, 가용성, 효용성, 융통성, 빠른응답, 통신비용 절감, 데이터가용성, 신뢰성, 시스템규모조절, 요구수용 증대
단점 : 비용, 오류잠재성 증대, 처리비용, 설계관리복잡성, 불규칙한 응답속도, 통제어려움, 데이터 무결성위협

분산 데이터베이스의 적용 기법
테이블 위치 분산 (테이블 구조 변하지 x, 각각의 테이블 위치가 다르게 지정)
테이블 분할 분산 (각각 테이블을 쪼개어 분산)
    수평 분할 - 로우단위, 수직 분할 - 칼럼단위
테이블 복제 분산 (동일한 테이블 다른 지역이나 서버에서 동시에 생성하여 관리)
    부분복제 - 통합된 건 본사, 각 지사별로 해당 로우
    광역복제 - 본사,지사 모두 동일한 데이터 가지고 있음
테이블 요약 분산
    분석요약 - 각 지사별로 요약, 본사에 통합
    통합요약 - 각 지사별로 존재하는 다른 내용이 정보요약, 본사에 통합

1차정규형 모든 도메인이 원자값만으로 구성되도록 하는 정규형. 중복제거
2차정규형 부분적함수종속 제거 (완전 함수 종속)
3차정규형 이행적 함수(X->Y, Y->Z, X->Z) 종속 제거

<2-1 SQL기본>

Table 의 구조 : 열(Column), 필드 (Field,Value), 행 (Row)
정규화 : 테이블 분할하여 데이터 정합성 확보하고, 불필요한 중복을 줄이는 프로세스
ERD 구성요소: 엔터티, 관계, 속성

데이터 유형
NUMERIC : 정수, 실수
CHARACTER(s)/CHAR(s) : 고정길이 VARCHAR2(s)/VARCHAR(s) : 가변길이

CHAR VS VARCHR
CHAR 문자열 비교 : 공백(BLANK)을 채워서 비교. 우선 짧은 쪽의 끝에 공백을 추가하여 2개의 데이터가 같은 길이가 되도록 한다. 그리고 앞에서부터 한 문자씩 비교한다. 끝의 공백만 다른 문자열은 같다고 판단한다.
VARCHAR 유형: 맨 처음부터 한 문자씩 비교하고 공백도 하나의 문자로 취급하므로 끝의 공백이 다르면 다른 문자로 판단한다.
ex) CHAR 유형 'AA' = 'AA ' VARCHAR 유형 'AA' <> 'AA '

DDL(데이터 정의어) : 데이터 구조를 정의하는데 사용되는 명령어
CREATE,ALTER,DROP,RENAME

테이블 생성 시 주의 사항 : 문자로 반드시 시작, A-Z, a-z, 0-9, _, $, #문자만허용

```
CREATE TABLE PLAYER(
PLAYER_ID CHAR(7) NOT NULL,
PLAYER_NAME VARCHAR(20) NOT NULL,
TEAM_ID CHAR(3) NOT NULL,
CONSTRAINT PLAYER_PK PRIMARY KEY (PLAYER_ID)
CONSTRAINT PLAYER_FK FOREIGN KEY (TEAM_ID) REFERENCES TEAM(TEAM_ID) );
```

테이블 구조 확인
Oracle : DESCRIBE 테이블명; SQL server : exec sp_help 'dbo.테이블명'

제약조건
기본키(PRIMARY KEY) : 테이블에 존재하는 각 행을 한 가지 의미로 특정할 수 있는 한 개 이상 칼럼. 기본키제약 = 고유키제약 & NOT NULL 제약
고유키(UNIQUE KEY) : 고유하게 식별하기 위한 고유키, NULL 값 가진 행 여러개 있어도 괜찮음
외부키(FOREIGN KEY) : 다른 테이블의 기본키로 사용되고 있는 관계를 연결하는 칼럼
NULL : 아직 정의되지 않은 미지의 값, 현재 데이터를 입력하지 못하는 경우
DEFAULT : 기본값을 사전에 설정. 데이터 지정하지 않는 경우 사전에 정의된 기본값 자동 입력됨

SELECT 문장을 통한 테이블 생성 사례
1) ORACLE CTAS : CREATE TABLE ~ AS SELECT ~

2) SQL SERVER SELECT * INTO TABLE1FROM TABLE2
※ 기존 테이블의 제약조건 중에 NOT NULL만 새로운 복제 테이블에 적용되고, 기본키, 고유키, 외래키 , CHECK 등의 다른 제약조건은 없어진다.

ALTER TABLE PLAYER ---------테이블변경
ADD (ADDRESS VARCHAR2(80)); ---------칼럼추가
ALTER TABLE PLAYER DROP COLUMN ADDRESS; ---------칼럼삭제
ALTER TABLE PLAYER MODIFY (ADDRESS VARCHAR2(80));-------Oracle 칼럼수정
ALTER TABLE PLAYER ALTER COLUMN ADDRESS VARCHAR2(80); SQL 칼럼수정
ALTER TABLE PLAYER RENAME COLUMN ADDRESS TO ADD----Oracle 칼럼이름 변경
ALTER TABLE PLAYER DROP CONSTRAINT 제약조건명;
ALTER TABLE PLAYER ADD CONSTRAINT 제약조건명 제약조건 (칼럼명);

RENAME 변경전 테이블명 TO 변경후 테이블명; ---oracle
sp_rename 변경전 테이블명, 변경후 테이블명; ---sql server

DROP TABLE PLAYER [CASCADE CONSTRAINT];
테이블과 관계 있었던 참조 제약조건도 삭제

TRUNCATE TABLE PLAYER ;
테이블 구조 유지한체 데이터만 전부 삭제

DML(데이터 조작어) : 자료들을 입력, 수정, 삭제, 조회
SELECT, INSERT, UPDATE, DELETE
실시간으로 테이블에 영향 미치지 않는다.
COMMIT이용해 TRANSACTION종료해야 실제 테이블에 반영(cf. DDL은 AUTO COMMIT임)

INSERT INTO PLAYER (PLAYER_ID, PLAYER_NAME) VALUES ('200207', '박지성');
UPDATE PLAYER SET POSITION = 'MF';
DELETE FROM PLAYER;
(cf. 테이블 전체 삭제하는 경우 , DELETE : 삭제된 로그 저장.
TRUNCATE : 삭제된 로그 없으므로 ROLLBACK 불가능, 시스템부하 적음)

SELECT PLAYER_ID [ALL/DISTINCT] FROM PLAYER;
ALL 이 Default 값. DISTINCT : 중복된 데이터 1건으로 처리해서 출력
SELECT 에서 좌측 정렬(문자 및 날짜 데이터), 우측 정렬(숫자 데이터) 임
WILDCARD. 해당 테이블의 모든 칼럼 정보 보고 싶을 때, *(애스터리스크) 이용
Alias 특징 : 컬럼명 바로 뒤에 위치, AS, as 키워드 사용가능, 이중 인용부호로 공백, 특수문자 포함 가능

합성 연산자 : || -> oracle, + = -> SQL Server ---- CONCAT(string1, string2)와 같음

TCL(트랜잭션 제어어) COMMIT, ROLLBACK
DML에 의해 조작된 결과를 작업단위(트랜잭션) 별로 제어하는 명령어
잠금(LOCKING)

트랜잭션의 특성
원자성 : 트랜잭션 정의된 연산들 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 한다. (all or nothing)
일관성 : 트랜잭션 실행 전 데이터베이스 내용 잘못되어 있지 않다면 트랜잭션 실행 이후에도 데이터베이스 내용 잘못 있으면 안된다.
고립성 : 트랜잭션 실행 도중 다른 트랜잭션의 영향 받아 잘못된 결과 만들어서는 안된다.
지속성 : 트랜잭션 성공적으로 수행되면 갱신한 데이터베이스 내용 영구적으로 저장된다.

COMMIT, ROLLBACK 효과
데이터 무결성 보장, 영구적인 변경 하기전 데이터 변경사항 확인 가능, 논리적 연관된 작업 그룹핑하여 처리가능
Commit : Oracle 은 Not Auto Commit, SQL Server 는 Auto Commit 이 Default 임

SQL Server트랜잭션 3가지방식
AUTO COMMIT : 명령어 성공적으로 수행 -> 자동으로 COMMIT, 오류 발생 ->ROLLBACK
암시적 트랜잭션 : Oracle과 같은 방식. 트랜잭션의 끝을 사용자가 명시적으로 COMMIT, ROLLBACK으로 처리
명시적 트랜잭션 : 트랜잭션 시작과 끝을 모두 사용자가 지정
BEGIN TRANSACTION(BEGIN TRAN)으로 트랜잭션시작

SAVEPOINT 의 이해가 필요 (Rollback 과 Savepoint 의 그림 이해)
SAVEPOINT 저장점명; ROLLBACK TO 저장점명; ---Oracle
SAVE TRANSACTION 저장점명; ROLLBACK TRANSACTION 저장점명; ---SQL Sever

연산자의 종류 (WHERE절에 사용되는..)
1) 비교 연산자 : =, >, >=, <, <=
2) SQL 연산자 : BETWEEN A AND B, IN(LIST), LIKE '비교문자열', IS NULL
3) 논리 연산자 : AND, OR, NOT
4) 부정연산자 :
!=, ^= , <> (같지 않다), NOT BETWEEN a AND b, NOT IN(list), IS NOT NULL

연산자 우선순위 : () -> NOT연산자 -> 비교,SQL비교연산자 ->AND ->OR

BETWEEN A AND B
A, B 모두를 포함하는 범위를 의미 (수학에서는 B 는 포함되지 않는 미만의 의미)
IN(LIST) 리스트 값 중 어느 하나라도 일치하면 된다.
LIKE '비교문자열' 비교문자열과 형태 일치하면 된다.
Like 시 사용하는 와일드 카드의 의미 : %(0개 이상 어떤 문자) ,_ (1개인 단일 문자 의미)

WHERE PLAYER_NAME LIKE '_A%' --- 선수 영문 이름 두 번째 문자가 A인 선수들 이름

IS NULL

SELECT PLAYER_NAME, POSITION WHERE POSITION IS NULL;

[표 Ⅱ-1-18] 문자 유형 비교 방법

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

내장함수

단일행 (문자형함수, 숫자형함수, 날짜형함수, 변환형함수, NULL관련함수)

다중행 (집계함수, 그룹함수, 윈도우함수)

단일행 함수의 종류 (문자형, 숫자형, 날짜형, 변환형, NULL관련 함수__ 날짜 별로 안중요)

1) 문자형 함수 : LOWER, UPPER, SUBSTR / SUBSTRING, LENGTH / LEN, LTRIM, RTRIM, TRIM, ASCII, CONCAT

2) 숫자형 함수 : ABS, MOD, ROUND, TRUNC, SIGN, CHR / CHAR, CEIL / CEILING, FLOOR, EXP, LOG, LN, POWER, SIN, COS, TAN

3)변환형 함수 : 암시적 데이터 유형 변환 (문자가 숫자로 변형됨)

4) NULL 관련 함수 : NVL / ISNULL, NULLIF, COALESCE

문자형 함수, 숫자형 함수 결과값을 묻는 경우가 많으므로 이해 필요

CEIL / CEILING() VS FLOOR()

1) CEIL / CEILING() : 숫자보다 크거나 같은 최소 정수를 리턴한다.

ex) CEIL(38.123) / CEILING(38.123) -> 39 CEILING(-38.123) -> 38

2) FLOOR() : 숫자보다 작거나 같은 최대 정수를 리턴한다.

ex) FLOOR(38.123) -> 38 FLOOR(-38.123) -> -39

※ CEILING은 천장, FLOOR은 바닥을 연상해서 외우면 좋을 듯 싶다.

단일행 Null 관련 함수

1) NVL(표현식1,표현식2) / ISNULL(표현식1,표현식2)

표현식1값이 NULL이면 표현식2값 출력한다. --- NVL(NULL판단대상,'NULL일 때 대체값')

2) NULLIF(표현식1,표현식2)

표현식1값이 표현식2와 같으면 NULL, 같지 않으면 표현식1값 출력한다

3) COALESCE(표현식1, 표현식2, ...)

임의의 개수 표현식으서 NULL이 아닌 최초의 표현식을 나타낸다. 모든 표현식이 NULL이라면 NULL값 리턴

(NULL과 관련된 함수가 아닌 것? 1.ISNULL 2.NULLIF 3.COALESCE 4.IS NOT NULL)

※ IS NULL, IS NOT NULL은 함수가 아니라 연산자이다.

[표 Ⅱ-1-26] 단일행 문자형 함수 사례

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

[표 Ⅱ-1-28] 단일행 숫자형 함수 사례

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

NVL / ISNULL 함수를 이용해 공집합을 9999로 바꾸기

SELECT NVL(MGR, 9999) MGR FROM EMP WHERE ENAME = 'JSC';

데이터를 찾을 수 없다.

-> NVL / ISNULL 함수는 NULL 값을 대상으로 다른 값으로 바꾸는 함수이지 공집합을 대상으로 하지 않는다.

SELECT NVL(MAX(MGR), 9999) MGR FROM EMP WHERE ENAME = 'JSC';

-> 집계함수를 인수로 한 NVL / ISNULL 함수를 이용해서 공집합인 경우에도 빈칸이 아닌 999로 출력하게 한다. 다른 함수와 달리 집계함수나 SCALAR SUBQUERY의 경우는 인수의 결과 값이 공집합인 경우에도 NULL을 출력한다.

Null 연산 : Quiz 로 풀어보기 (Null 연산 Quiz 를 통한 이해 (**))

집계함수 (SELECT , HAVING, ORDER BY절에 사용 할 수 있다)

COUNT(*) : NULL 값을 포함한 행의 수

COUNT(표현식) : 표현식의 값이 NULL 값인 것을 제외한 행의 수

SUM() : NULL 값을 제외한 합계

AVG() : NULL 값을 제외한 평균

MAX() : 최대값 출력 MIN() :최소값 출력

※ 조건절에 해당하는 데이터가 없을 때 COUNT(*)의 결과 값은 0

집계함수 통계정보 NULL값 가진 행 제외하고 수행

COL1	COL2	COL3	COL4
NULL	NULL	50	30
30	20	10	30
NULL	10	NULL	NULL

SUM(COL1) + SUM(COL2 + COL3) + SUM(COL4)의 값은?

답: 120

(NULL 연산 결과 값도 NULL. NULL + - * / 2 = NULL)

Group by 의 의미와 Having 의 의미, 그리고 각 예제를 이해하면 됨 (직접 해보기)

GROUP BY 절, HAVING절 특징

1) GROUP BY 절에서는 ALIAS명을 사용할 수 없다.

2) WHERE 절은 전체 데이터를 GROUP으로 나누기 전에 행들을 미리 제거시킨다.

3) HAVING 절은 일반적으로 GROUP BY 절 뒤에 위치한다.

4) GROUP BY, HAVING 절에는 SELECT 절에 정의되지 않은 컬럼은 사용 못함

5) 집계함수 WHERE절에 올 수 없다. GROUP BY 통해 소그룹별 기준 정한 후, SELECT절에서 집계함수 사용

※ ORDER BY 절을 SELECT 절에 정의되지 않은 컬럼 사용 가능

Order by 특징 기본적인 정렬순서는 오름차순 (ASC)이다. cf. 내림차순(DESC)

숫자 오름차순 - 가장 작은 값부터 출력

날짜 오름차순 - 가장 날짜값 빠른 값이 먼저 출력

Oracle - NULL 가장 큰 값으로 간주. 오름차순 -> 가장 마지막. 내림차순 -> 가장 먼저
SQL Server -NULL 가장 작은 값. 오름차순 -> 가장 먼저, 내림차순 -> 가장 마지막 위치
ORDER BY 절에서는 칼럼명, ALIAS명, 칼럼순서 같이 혼용해서 사용 가능

SELECT 문장 실행 순서
FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY
ex)
SELECT POSITION 포지션, ROUND(AVG(HEIGHT),2) 평균키
FROM PLAYER
GROUP BY POSITION
HAVING MAX(HEIGHT) >= 190
ORDER BY 1, 평균키 ;

ROWNUM WHERE 절에서 행의 개수를 제한하는 목적으로 사용
한 행만 가져오고 싶을 때
WHERE ROWNUM = 1; WHERE ROWNUM <= 1; WHERE ROWNUM < 2;
두건 이상의 N행을 가져오고 싶을 때
WHERE ROWNUM = N; (X) WHERE ROWNUM <= N; (O) WHERE ROWNUM < N+1; (O)

c.f) sql server
SELECT **TOP**(2) **WITH TIES** ENAME, SAL
(1위 한명, 공동2위가 2명있을 때
with ties 조건 추가하면 결과 3건 출력됨. with ties 없으면 결과 2건 출력됨)

Join : Equal Join , Non Equal Join -> 두개 이상의 테이블에서 컬럼을 가져오는 방법
EQUI JOIN(등가 조인) : where 절에 join 조건 "="연산자 사용해서 표현
ex) **WHERE** PLAYER.TEAM_ID = TEAM.TEAM_ID **AND** PLAYER.POSITION = 'GK';
ex) **FROM** PLAYER **INNER JOIN** TEAM **ON** PLAYER.TEAM_ID = TEAM.TEAM_ID
WHERE PLAYER.POSITION = 'GK';
INNER JOIN 참여하는 대상 테이블이 N개, 필요한 JOIN 조건은 N-1개

NON EQUI JOIN(비등가 조인) : BETWEEN, >, >=, <, <= 등의 연산자 사용해 JOIN
ex) WHERE E.SAL **BETWEEN** S.LOSAL **AND** S.HISAL;

일반 집합연산자와 현재의 SQL비교
UNIO연산(UNION 기능으로), INTERSECTION연산(INTERSECT 기능으로)
DIFFERENCE연산(EXCEPT,MINUS기능으로), PRODUCT연산(CROSS JOIN기능으로)

순수 관계 연산자와 현재의 SQL비교
Select (Where 절로 구현) , Project (Select 로 구현)
Natural Join(다양한 join기능으로구현), Divide (사용하지 않음)

Ansi SQL 의 Join : Inner Join (Natural Join, Using 조건절, On 조건절) , Cross Join, Outer Join

INNER JOIN : JOIN조건을 FROM절에서 정의하겠다. USING조건절, ON조건절 필수적
NATURAL JOIN : 두 테이블 간 동일한 이름을 값는 모든 컬럼에 대해 Equal Join 수행 (Using, ON 절 정의 불가, SQL Server 미지원)
JON에 사용된 컬럼들은 같은 데이터 유형이어야 하며, ALIAS나 테이블명과 같은 접두사를 붙일 수 없다.

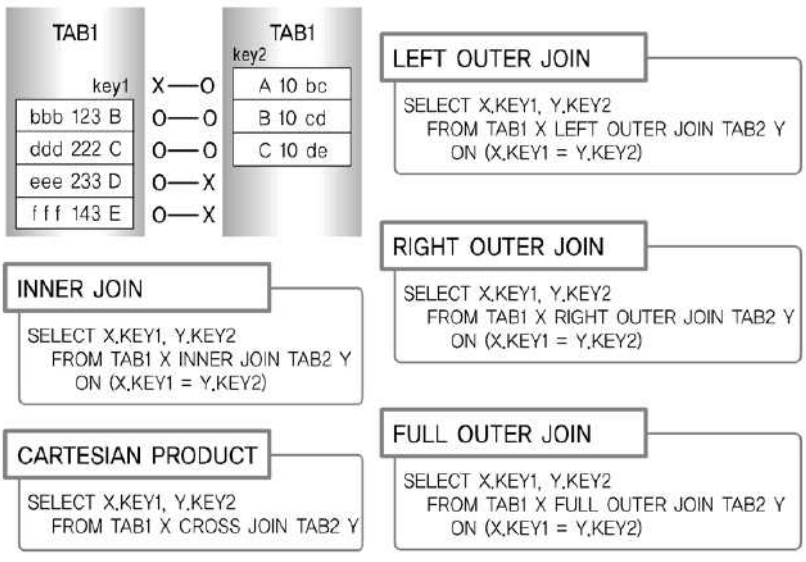
NATURAL JOIN은 JOIN에 사용된 같은 이름의 칼럼을 하나로 처리, INNER JOIN의 경우는 2개의 칼럼으로 표시한다.

USING 조건절
FROM 절에 USING 조건절을 이용하면 같은 이름을 가진 컬럼들 중에서 원하는 컬럼에 대해서만 선택적으로 EQUI JOIN을 할 수 있다.
SELECT * **FROM** DEPT **JOIN** DEPT_TEMP **USING** (DEPTNO);
※ JOIN 칼럼에 대해서는 ALIAS나 테이블 이름과 같은 접두사를 붙일 수 없다.

ON 조건절
두 테이블 간 특정 칼럼으로 Equal Join 수행, 칼럼명 다르더라도 JOIN조건 사용할 수 있는 장점 가짐 (ON 조건절 사용시 and 조건 추가 가능, where 절과 의미 구분 필요)

CROSS JOIN : 두 테이블의 Cartesian Product 임 (곱 조인) M*N건 조합 발생

OUTER JOIN : Left (Right) Outer Join, Full Outer Join
FROM EMP E **RIGHT JOIN** DEPT D



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

INNER JOIN : B-B, C-C (2건)
LEFT OUTER JOIN : B-B, C-C, D-NULL, E-NULL (4건)
RIGHT OUTER JOIN : NULL-A, B-B, C-C (3건)
FULL OUTER JOIN : NULL-A, B-B, C-C, D-NULL, E-NULL (5건)
CROSS JOIN : B-A, B-B, B-C, C-A, C-B, C-C, D-A, D-B, D-C, E-A, E-B, E-C
(3*4 12건)

집합 연산자 (JOIN사용하지 않고 연관된 데이터를 조회하는 방법)
Union all 은 제외한 모든 집합 연산자는 Sorting 을 수행함
UNION : 여러개의 SQL의 결과에 대한 합집합으로 결과에서 모든 중복된 행을 하나의 행으로 만든다.
UNION ALL : 중복된 행도 그대로 결과에 표시한다.
EXCEPT/MINUS (차집합) => NOT EXISTS, NOT IN 서브쿼리로 변경가능
INTERSECT (교집합) => EXISTS, IN 서브쿼리로 변경 가능

계층형 질의 계층형 간단하게 설명하기는 어려움. (문제의 범위나 이런걸로 봤을 때 일단 Skip 함)
1) START WITH 절은 계층구조 전개이 시작 위치를 지정하는 구문이다.
2) CONNECT BY 절은 다음에 전개될 자식 데이터를 지정하는 구문이다.
3) 루트 데이터는 LEVEL 1이다. (0이 아님)
4) PRIOR 자식 =부모 (부모->자식 방향으로 전개. 순방향 전개)
  PRIOR 부모 = 자식 (자식->부모 방향으로 전개. 역방향 전개)
ex)
SELECT LEVEL, LPAD(' '. 4*(LEVEL-1) || EMPNO 사원, MGR 관리자, CONNECT_BY_ISLEAF ISLEAF **FROM** EMP **START WITH** MGR IS NULL **CONNECT BY PRIOR** EMPNO=MGR;

셀프조인 : 동일 테이블 사이의 조인
SELECT WORKER.EMPNO 사원번호, WORKER.ENAME 사원명, MANAGER.ENAME 관리자명
FROM EMP WORKER, EMP MANAGER
WHERE WORKER.MGR = MANAGER.EMPNO;

서브쿼리 (하나의 SQL문안에 포함되어 있는 또 다른 SQL문)
1) 서브쿼리는 메인커리의 칼럼을 모두 사용할 수 있지만 메인쿼리는 서브쿼리의 칼럼을 사용할 수 없다. 질의 결과에 서브쿼리의 칼럼을 표시해야 한다면 조인방식으로 변환하거나 함수, 칼라 서브쿼리 등을 사용해야 한다.
2) 서브쿼리는 서브쿼리 레벨과 상관없이 항상 메인쿼리 레벨로 결과집합이 생성된다.
예를들어, 메인쿼리로 조직(1), 서브쿼리로 사원(M) 테이블을 사용하면 결과집합은 조직(1) 레벨이 된다.
3) 서브쿼리에서는 ORDER BY 절을 사용하지 못한다. ORDER BY 절은 SELECT 절에서 오직 한 개만 올 수 있기 때문에 ORDER BY 절은 메인쿼리의 마지막 문장에 위치해야 한다.

반환되는 데이터의 형태에 따른 서브쿼리 분류 : 단일 행 서브쿼리, 다중 행 서브쿼리, 다중칼럼

단일행 서브쿼리

서브쿼리가 단일행 비교 연산자 (=, <, <=, >, >=, <>)와 함께 사용할 때는 서브쿼리의 결과건수가 반드시 1건 이하이어야 한다.

다중행 서브쿼리

서브쿼리의 결과가 2건 이상 반환될 수 있다면 반드시 다중행 비교연산자 (IN, ALL, ANY, SOME)와 함께 사용해야 한다.

ex) IN (서브쿼리), ALL(서브쿼리), ANY(서브쿼리), EXISTS(서브쿼리)

다중칼럼 서브쿼리

서브쿼리의 결과로 여러개의 칼럼이 반환되어 메인쿼리의 조건과 동시에 비교되는 것을 의미

ex) WHERE (TEAM_ID, HEIGHT) IN (SELECT TEAM_ID, MIN(HEIGHT) FROM PLAYER GROUP BY TEAM_ID)

동작 방식에 따른 서브쿼리 분류

비연관 서브쿼리 (서브쿼리가 메인쿼리 칼럼을 가지고 있지 않는 형태)

연관 서브쿼리 (서브쿼리가 메인쿼리 칼럼을 가지고 있는 형태)

EXISTS서브쿼리는 항상 연관 서브쿼리로 사용된다. 또한 조건 만족하는 건이 여러건이더라도 조건을 만족하는 1건만 찾으면 추가적인 검색 진행하지 않는다.

그 밖의 위치에서 사용하는 서브쿼리

SELECT절에 서브쿼리 : 스칼라 서브쿼리 (한 행, 한 칼럼만을 반환하는 서브쿼리)

FROM절에 서브쿼리 : 인라인 뷰(Inline Veiw)

HAVING 절에 서브쿼리 사용, UPDATE문의 SET절에 서브쿼리 사용, INSERT문의 VALUES절에 서브쿼리 사용.

뷰(실제 데이터 가지고 있지 않는 가상테이블)의 장점 독립성, 편리성, 보안성

그룹함수

ROLLUP : 인수 계층구조. 인수순서 바뀌면 수행 결과도 바뀌게 된다. 칼럼 수=n ->N+1 LEVEL의 Subtotal이 생성. GROUP BY ROLLUP (DNAME, JOB);

ROLLUP이나 CUBE에 의한 소계가 계산된 결과에는 GROUPING(EXPR) = 1, 그렇지 않으면 GROUPING(EXPR)=0

DNAME	JOB	Total Empl	Total Sal
ACCOUNTING	CLERK	1	1300
ACCOUNTING	MANAGER	1	2450
ACCOUNTING	PRESIDENT	1	5000
ACCOUNTING		3	8750
RESEARCH	ANALYST	2	6000
RESEARCH	CLERK	2	1900
RESEARCH	MANAGER	1	2975
RESEARCH		5	10875
SALES	CLERK	1	950
SALES	MANAGER	1	2850
SALES	SALESMAN	4	5600
SALES		6	9400
		14	29025

CUBE : 결합 가능한 모든 값에 대하여 다차원 집계 생성. 인수들 간 평등한 관계. 순서 바뀌어도 상관 없음. 2의 N승 LEVEL의 Subtotal 생성.
GROUP BY CUBE (DNAME, JOB);

DNAME	JOB	Total Empl	Total Sal
All Departments	All Jobs	14	29025
All Departments	CLERK	4	4150
All Departments	ANALYST	2	6000
All Departments	MANAGER	3	8275
All Departments	SALESMAN	4	5600
All Departments	PRESIDENT	1	5000
SALES	All Jobs	6	9400
SALES	CLERK	1	950
SALES	MANAGER	1	2850
SALES	SALESMAN	4	5600
RESEARCH	All Jobs	5	10875
RESEARCH	CLERK	2	1900
RESEARCH	ANALYST	2	6000
RESEARCH	MANAGER	1	2975
ACCOUNTING	All Jobs	3	8750
ACCOUNTING	CLERK	1	1300
ACCOUNTING	MANAGER	1	2450
ACCOUNTING	PRESIDENT	1	5000

GROUPING SETS : 인수들에 대한 개별 집계. 평등한 관계 -> 인수의 순서 바뀌어도 결과 같다.
GROUP BY GROUPING SETS (DNAME, JOB);

```
SELECT DECODE(GROUPING(DNAME), 1,'ALL DEPARTMENTS', DNAME) AS DNAME,
       DECODE(GROUPING(JOB),    1,'ALL JOBS'  ,      JOB )  AS JOB,
       COUNT(*) "TOTAL EMPL", SUM(SAL) "TOTAL SAL"
FROM   EMP, DEPT
WHERE  DEPT.DEPTNO = EMP.DEPTNO
GROUP BY GROUPING SETS ( DNAME, JOB ) ;
```

DNAME	JOB	TOTAL EMPL	TOTAL SAL
RESEARCH	ALL JOBS	5	10875
SALES	ALL JOBS	6	9400
ACCOUNTING	ALL JOBS	3	8750
ALL DEPARTMENTS	MANAGER	3	8275
ALL DEPARTMENTS	ANALYST	2	6000
ALL DEPARTMENTS	PRESIDENT	1	5000
ALL DEPARTMENTS	SALESMAN	4	5600
ALL DEPARTMENTS	CLERK	4	4150

윈도우함수

그룹 내 순위(RANK) 관련 함수 : RANK, DENSE_RANK, ROW_NUMBER

RANK : 특정 항목(칼럼)에 대한 순위를 구하는 함수. 동일한 값에 대해서는 동일한 순위.

DENSE_RANK : 동일한 순위를 하나의 건수로 취급

ROW_NUMBER : 동일한 값이라도 고유한 순위를 부여

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

```
RANK        : 1 2 2 4
DENSE_RANK  : 1 2 2 3
ROW_NUMBER  : 1 2 3 4
```

SELECT DENSE_RANK() OVER (ORDER BY SAL DESC) DENSE_RANK

SELECT ROW_NUMBER() OVER (ORDER BY SAL DESC) ROW_NUMBER

그룹 내 집계(AGGREGATE) 관련 함수 : SUM ,MAX, MIN, AVG, COUNT

(c.f SQL Server의 경우 집계 함수는 뒤에서 설명할 OVER 절의 내의 ORDER BY 지원 하지 않음)

SELECT MGR, ENAME, SAL, **SUM(SAL) OVER** (**PARTITION BY** MGR **ORDER BY** SAL **RANGE UNBOUNDED PRECEDING**) AS MGR_SUM **FROM** EMP ;

RANGE UNBOUNDED PRECEDING : 현재 행 기준으로 파티션 내의 첫 번째 행까지의 범위 지정
동일 순위의 모든 값에 대한 총합계로 누적됨

SELECT MGR, ENAME, HIREDATE, SAL, **ROUND(AVG(SAL) OVER (PARTITION BY** MGR **ORDER BY** HIREDATE **ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING**)) AS MGR_AVG **FROM** EMP ;

ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING : 현재 행을 기준으로 파티션 내에서 앞의 한건, 현재 행, 뒤의 한 건을 범위로 지정 . ROWS 는 현재 행의 앞뒤 ROWS 를 말함

SELECT MAX(SAL) OVER (PARTITION BY MGR) --MAX 대신 MIN,COUNT,SUM,AVG 등 그룹내 집계함수사용

RANGE BETWEEN 50 PRECEDING AND 150 FOLLOWING : 현재 행(Row)의 급여값을 기준으로 급여가 -50 에서 +150 의 범위 내에 포함된 모든 행이 대상. RANGE 는 현재 행의 데이터 값을 기준으로 앞뒤 데이터 값의 범위를 표시

그룹 내 행 순서 관련 함수 : FIRST_VALUE, LAST_VALUE, LAG, LEAD 함수

(ORACLE 에서만 지원)

FIRST_VALUE : 파티션별 윈도우에서 가장 먼저 나온 값을 구한다. 다른 함수와 달리 공동 등수를 인정하지 않고 처음 나온 행만을 처리함.

SELECT DEPTNO, ENAME, SAL,M **FIRST_VALUE(ENAME) OVER (PARTITION BY** DEPTNO **ORDER BY** SAL DESC **ROWS UNBOUNDED PRECEDING**) AS DEPT_RICH
FROM EMP ;

ROWS UNBOUNDED PRECEDING : 현재 행 기준으로 파티션 내의 첫 번째 행까지의 범위를 지정

```
    DEPTNO ENAME             SAL DEPT_RICH
---------- ---------- ---------- ----------
        10 KING             5000 KING
        10 CLARK            2450 KING
        10 MILLER           1300 KING
-----------------------------------------------------
        20 SCOTT            3000 SCOTT << DEPTNO = 20
        20 FORD             3000 SCOTT << DEPTNO = 20
        20 JONES            2975 SCOTT
        20 ADAMS            1100 SCOTT
        20 SMITH             800 SCOTT
```

LAST_VALUE : 파티션별 윈도우에서 가장 나중에 나온 값을 구한다. 공동 등수를 인정하지 않음.
SELECT DEPTNO, ENAME, SAL, LAST_VALUE (ENAME) OVER (PARTITION BY DEPTNO ORDER BY SAL DESC ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS DEPT_POOR
FROM EMP ;
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING : 현재 행을 포함해서 파티션 내의 마지막 행까지의 범위를 지정

```
   DEPTNO ENAME             SAL DEPT_POOR
--------- ---------- ---------- ----------
        10 KING             5000 MILLER
        10 CLARK            2450 MILLER
        10 MILLER           1300 MILLER
-------------------------------------------
        20 SCOTT            3000 SMITH
        20 FORD             3000 SMITH
        20 JONES            2975 SMITH
        20 ADAMS            1100 SMITH
        20 SMITH             800 SMITH
```

LAG : 파티션별 윈도우에서 이전 몇 번째 행의 값을 가져 올 수 있다.
ex) 2 행 앞의 SALARY 가져오기, 가져올 값이 없으면 0으로 처리
SELECT ENAME, HIREDATE, SAL, LAG(SAL, 2, 0) OVER (ORDER BY HIREDATE) AS PREV_SAL FROM EMP WHERE JOB ='SALESMAN';

```
ENAME      HIREDATE                SAL   PREV_SAL
---------- --------------- ---------- ----------
ALLEN      20-FEB-81             1600          0
WARD       22-FEB-81             1250          0
TURNER     08-SEP-81             1500       1600
MARTIN     28-SEP-81             1250       1250
```

LEAD : 파티션별 윈도우에서 이후 몇 번째 행의 값을 가져 올 수 있다.
ex) 입사일자가 빠른 기준으로 정렬하고, 바로 다음에 입사한 인력의 입사일자를 함께 출력한다.
SELECT ENAME, HIREDATE, LEAD(HIREDATE,1) OVER (ORDER BY HIREDATE) AS "NEXTHIRED" FROM EMP ;

```
ENAME      HIREDATE        NEXTHIRED
---------- --------------- -----------
SMITH      17-DEC-80       20-FEB-81
ALLEN      20-FEB-81       22-FEB-81
WARD       22-FEB-81       02-APR-81
JONES      02-APR-81       01-MAY-81
BLAKE      01-MAY-81       09-JUN-81
```

그룹 내 비율 관련 함수 : CUME_DIST, PERCENT_RANK, RATIO_TO_REPORT, NTILE

칼럼 값에 대한 백분율 -> RATIO_TO_REPORT

파티션 내 전체 SUM(칼럼) 값에 대한 행별 칼럼 값의 백분율을 소수점으로 구함

```
SQL> SELECT ENAME, SAL,
           ROUND(RATIO_TO_REPORT(SAL) OVER (),2) AS R_R,
           SUM(SAL) Over ( Order by JOB  ) AS "Total"
     FROM EMP
     WHERE JOB='SALESMAN' ;

ENAME             SAL        R_R      Total
---------- ---------- ---------- ----------
ALLEN            1600        .29       5600 << ( 1600/5600 )
WARD             1250        .22       5600 << ( 1250/5600 )
MARTIN           1250        .22       5600 << ( 1250/5600 )
TURNER           1500        .27       5600 << ( 1500/5600 )
```

행의 순서에 대한 (0부터 1사이 값) 백분율 -> PERCENT_RANK

파티션별 윈도우에서 제일 먼저 나오는 것을 0으로, 제일 늦게 나오는 것으로 1로하여 값이 아닌 행의 순서별 백분율을 구함

```
SQL> SELECT DEPTNO, ENAME, SAL,
           PERCENT_RANK() OVER ( PARTITION BY DEPTNO ORDER BY SAL DESC ) AS P_R
    FROM EMP ;

    DEPTNO ENAME             SAL        P_R
---------- ---------- ---------- ----------
        10 KING             5000          0
        10 CLARK            2450         .5
        10 MILLER           1300          1  <<
------------------------------------------------------------------
        20 SCOTT            3000          0
        20 FORD             3000          0
        20 JONES            2975         .5
        20 ADAMS            1100        .75
        20 SMITH             800          1  <<
```

1/(파티션)전체 건수로 표현하는 백분율 -> CUME_DIST

파티션별 윈도우에서 전체건수에 현재 행보다 작거나 같은 건수에 대한 누적백분율 구함

```
SQL> SELECT DEPTNO, ENAME, SAL,
           CUME_DIST() OVER ( PARTITION BY DEPTNO ORDER BY SAL ) AS A
    FROM   EMP ;

    DEPTNO ENAME             SAL          A
---------- ---------- ---------- ----------
        10 MILLER           1300 .333333333
        10 CLARK            2450 .666666667
        10 KING             5000          1
        20 SMITH             800         .2
        20 ADAMS            1100         .4
        20 JONES            2975         .6
        20 SCOTT            3000          1
        20 FORD             3000          1
        30 JAMES             950 .166666667
        30 MARTIN           1250         .5
        30 WARD             1250         .5
        30 TURNER           1500 .666666667
        30 ALLEN            1600 .833333333
        30 BLAKE            2850          1
```

파티션별 전체 건수를 ARGUMENT 값으로 N 등분한 결과 구함 -> NTILE

```
SQL> SQL> SELECT ENAME, SAL, NTILE(4) OVER (ORDER BY SAL DESC ) AS QUAR_TILE
     FROM EMP ;

ENAME             SAL  QUAR_TILE
---------- ---------- ----------
KING             5000          1
FORD             3000          1
SCOTT            3000          1
JONES            2975          1
-----------------------------------------------------------------
BLAKE            2850          2
CLARK            2450          2
ALLEN            1600          2
TURNER           1500          2
-----------------------------------------------------------------
MILLER           1300          3
WARD             1250          3
MARTIN           1250          3
-----------------------------------------------------------------
ADAMS            1100          4
JAMES             950          4
SMITH             800          4
-----------------------------------------------------------------
```

DCL(DATA CONTROL LANGUAGE) :유저를 생성하고 권한을 제어할 수 있는 명령어
대부분의 데이터베이스는 데이터 보호와 보안을 위해서 유저와 권한을 관리함
개별 오브젝트에 대한 작업을 위해서는 오브젝트 권한 부여 필요

ROLE : 유저와 권한들 사이에서 중개 역할
ROLE 을 생성하고, ROLE 에 각종 권한들을 부여한 후, ROLE 을 다른 ROLE 혹은 유저에 부여
ROLE 을 만들어 사용하는 것이 권한을 직접 부여하는 것보다 빠르고 안전하게 관리 가능

절자형SQL 이용하여 SQL문의 연속적인 실행이나 조건에 따른 분기처리를 이용하여 특정 기능을 수행하는 저장모듈인 PROCEDURE, TRIGGER, USER DEFINED FUNCTION을 만들 수 있다.
T-SQL : SQL Server를 제어하기 위한 언어

USER DEFINED FUNCTION : PROCEDURE처럼 절차형 SQL을 로직과 함께 데이터베이스 내에 저장해 놓은 명령문의 집합을 의미. PROCEDURE과 달리 RETURN 사용해서 하나의 값을 반드시 되돌려주어야 한다.

TRIGGER : 특정한 테이블에 INSERT,UPDATE,DELETE 와 같은 DML 문이 수행되었을 때, 데이터베이스에서 자동으로 동작하도록 작성한 프로그램
사용자가 직접 호출하여 사용하는 것이 아니고 데이터베이스에서 자동적으로 수행하게 됨
-이벤트 발생 대상 : 테이블, 뷰, 데이터베이스
-발생 범위 : 전체트랜잭션 작업, 각 행에 대해서 발생

[표 Ⅱ-2-19] 프로시저와 트리거의 차이점

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

옵티마이저

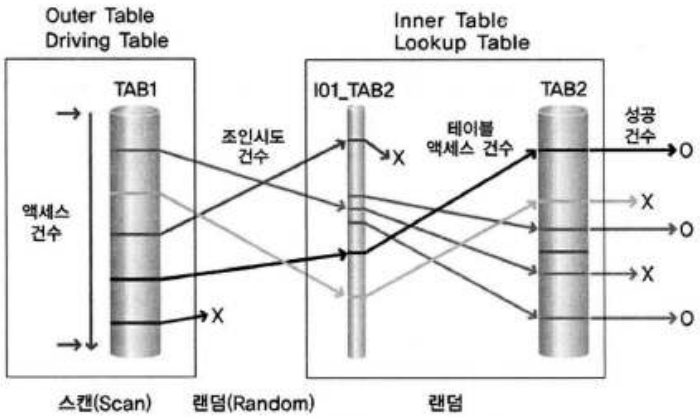
규칙기반 옵티마이저 : 우선순위가 높은 규칙이 적은 일량으로 해당작업을 수행하는 방법

비용기반 옵티마이저 : SQL문을 처리하는데 필요한 비용이 가장 적은 실행계획을 선택하는 방식

실행계획 : SQL에서 요구한 사항을 처리하기 위한 절차와 방법을 의미

실행계획을 구성하는 요소에는 조인순서(Join Order), 조인기법(Join Method), 엑세스 기법(Access Method), 최적화 정보(Optimization Information), 연산(Operation) 등이 있다.

SQL처리흐름도 : 실행계획을 시각화한 도표



[그림 Ⅱ-3-5] SQL 처리 흐름도

- TABL1을 outer Table 또는 Driving table 이라하고 TAB2를 Inner TAble 또는 Lookup table이라고 한다.
- 조인순서 TAB1->TAB2
- TAB1 : 풀스캔 TAB2는 인덱스스캔
- 조인방법 NL Join
- TAB1에대한 엑세스는 SCAN, TAB2는 랜덤엑세스(대량의 데이터를 랜덤엑세스하면 많은 I/O 발생으로 성능에 좋지않다.)

ORACLE의 INDEX 액세스 기법의 종류

INDEX UNIQU SCAN, INDEX RANGE SCAN, INDEX RANGE SCAN DESCNDING,
INDEX FULL SCAN, FAST FULL INDEX SCAN, INDEX SKIP SCAN