

## ❖ 데이터베이스 용어 정리

### Database



- 무결성(데이터 정확성)이 보장된 데이터를 모아놓은 집합/저장소
- 정확한 데이터를 저장하기 위해 분류체계(속성, 특성 ex(표의 열 이름))를 기준으로 데이터를 저장
- 단순히 데이터를 저장하기 위한 용도로 사용되는게 아니라, 저장된 데이터를 가공해서 업무에 도움이 될 수 있는 신뢰성 있는 정보로 활용하기 위해 사용

### DBMS



- 데이터베이스를 관리하기 위한 중간 역할을 수행하는 소프트웨어 또는 시스템
- 사용자와 데이터베이스를 연계하여 포괄적인 관리시스템
- Oracle, SqlServer 등이 존재함

### Table



- 데이터를 저장하기 위한 저장 단위로 표와 같이 열과 행으로 구성, 표의 제목
- 이상적인 테이블은 한 개의 컬럼에 한 개의 PK가 설치되어 있는 구조

### Column



- 테이블에 데이터를 저장하기 위한 분류(표의 열 이름)
- 한 개의 테이블은 여러 개의 column들로 구성이 되어 있음
- 데이터 무결성을 보장하기 위해서 column들은 구조를 가짐( Datatype, Data의 크기/길이)

# ❖ 데이터베이스 용어 정리

## Primary Key



- 데이터 무결성을 보장하기 위한 제약조건의 일부
- 테이블의 한 개의 column에 반드시 설치되어 있어야 함
- 테이블을 대표하는 Column에 설치하며, Not null + Unique 특성을 가짐
- 예외적으로 복합PK를 지정할 수 있음 ( 여러 개의 컬럼에 한 개의 PK를 설치)

## Foreign key



- 다른 테이블의 컬럼의 데이터를 참조할 때 사용
- 관계형 데이터베이스 내에서 테이블과 테이블 간의 관계를 설정할 때 사용하며 종속적인 삭제를 방지
- 참조 column은 반드시 PK, UK가 지정되어 있어야 함

## 트랜잭션



- 데이터베이스 내에 발생하는 작업의 단위를 의미하며, 데이터를 저장하기 위한 일괄처리의 기준이 됨
- DML, DDL, DCL언어에 의해 트랜잭션이 발생
- DML 문장 전체가 하나의 트랜잭션, DDL, DCL언어는 한 문장이 하나의 트랜잭션으로 처리
- 트랜잭션이 실행되면 ,Commit과 Rollback 중 하나가 발생

## Commit



- 트랜잭션이 정상적으로 수행이 되었을 때, 트랜잭션의 전체 내용을 Database에 물리적으로 영구히 저장

## Rollback



- 트랜잭션이 비정상적으로 수행이 되었을때 commit이 되면 데이터무결성이 문제가 발생 함
- 잘못된 트랜잭션의 내용이 저장되지 않도록, 트랜잭션의 전체 내용을 취소할때 사용

# ❖ 데이터베이스 용어 정리

SQL



- 데이터베이스와 통신하기 위한 유일한 수단
- 모든 행위는 SQL언어를 통해 진행, Select, DML, DDL, DCL, TCL 언어로 구성

Select



- 테이블의 저장된 데이터를 검색하기 위한 명령문
- 검색을 하는 이유는 DB에 저장된 데이터를 가져와서 업무에 도움이 될 수 있는 정보로 활용하기 위함

DML



- 데이터베이스에 저장된 데이터에 대한 작업을 수행하는 명령문으로 insert, update, delete로 구성
- 트랜잭션에 가장 많은 영향을 주는 언어로 DB를 운영하기위한 언어임
- DB의 Data는 현재시점만을 가르켜야 함. Data가 현재시점을 반영 할 수 있도록 삽입, 변경, 삭제를 통해 DB를 운영하는 언어

DDL



- DB내에서는 물리적인 구조를 가지고 있는 애들을 객체(table, view, sequence, index)라고 표시함
- 객체들의 구조를 정의하는 언어로 create, alter, drop 명령어를 사용

DCL



- DB의 보안 정책 중 하나가 권한이 있는 user들이 불법적인 행위를 하지 못하도록 방지하는 것
- DB내부에서는 사용자 관리를 ID/PW, 사용자에게 부여하는 권한으로 사용자를 관리함
- grant(권한부여), revoke(권한회수) 명령어를 통해 사용자 관리 ( DBA가 관리 )

# ❖ SQL SELECT문 – 전체 문법

SELECT    DISTINCT    \*    COLUMN    ALIAS



- 보고자 하는 데이터를 소유한 컬럼이름을 명시
- 명시된 컬럼의 데이터가 결과로 출력
- 출력된 결과의 중복값을 제거(Distinct)
- 결과 출력시 컬럼이름을 변경하고자 할때 alias 기능을 사용

FROM



- Select 리스트절에 명시한 컬럼의 소유주인 테이블 이름을 명시, 명시된 테이블로부터 데이터를 가져옴

WHERE



- 테이블로부터 데이터를 가져올때 조건을 두어 조건에 맞는 행에 대해서만 검색하고자 할때 사용
- 행에 대한 조건식(컬럼 연산자 값)을 명시
- Alias 명 사용 불가, 그룹함수 사용불가

GROUP BY



- Select 리스트절에 그룹함수가 사용되었을때, Select 리스트절에 명시된 그룹함수가 사용되지 않은 컬럼을 기준으로 그룹화 수행을 위해 사용

HAVING



- Group by절이 사용되어 소그룹이 만들어졌을때, 조건에 맞는 그룹에 대해 검색하고자 할때 사용, group by절과 같이사용

ORDER BY



- 결과로 출력하고자 하는 데이터를 정렬하기 위해서 사용

## ❖ SQL SELECT문 – 연산자

단일 비교연산자 ( =, <, >, != 등)



- 한 개의 값을 비교하고자 할 때 사용
- 반드시 값과 비교대상인 컬럼의 데이터 타입이 같아야 함

Between A and B



- 사이, 범위 값을 검색하고자 할 때 사용
- 이상 이하의 값을 찾을 때 사용, 모든 데이터 타입에 사용가능

IN



- 동시에 똑같은 값 여러 개를 찾고자 할 때 사용
- 모든 데이터타입에 사용 가능

Like



- 전체가 아닌 일부 값만을 알고 있을 때 사용
- %(문장 전체), \_(문자하나)를 사용하여 검색

IS NULL



- Null값(없는 값)을 찾을때 사용

- Where 및 Having 절은 반드시 조건식이 명시되어야 함. 조건식은 컬럼 연산자 값( salary>1000, avg(salary)>10000))으로 구성됨
- 조건을 추가할때도 반드시 새로운 조건식이 명시되어야 하며, And와 Or 연산자를 사용하여 조건을 추가해야 함

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시 ( JOIN )

- 여러 개의 테이블에서 데이터를 검색하고 할 때 사용
- From 절에 여러 개의 테이블이 명시 됨
- 조인이 수행이 되면, 조인 대상 테이블이 가지고 있는 행을 하나의 행으로 조합해야 함
- 하나의 행을 조합 할때 무결성이 보장된(정확한) 행으로 조합하기 위해 조인조건을 사용하여 하나의 행으로 조합
- 조인이 수행이 되면 N-1에 해당하는 조인조건을 반드시 Where절에 명시해야 함
- 조인문장 내에 조인 대상 테이블들이 보유한 컬럼 중 똑같은 이름의 컬럼을 사용할 시에는 반드시 소유주인 테이블이름을 접두어로 붙여야 함  
( Table Fullname, ,From절의 Alias 사용)

## ❖ SQL SELECT문 – 조인 종류

### 등가 조인 ( Equi-Join )



- 조인 대상 테이블에 같은 데이터가 존재 할때 수행하는 조인
- PK-FK로 관계가 설정된 테이블에서 많이 사용
- = 연산자를 사용 ( d.department\_id=e.department\_id)

### 비등가 조인 ( Non Equi-Join )



- 조인 대상 테이블에 같은 데이터가 존재하지 않을때 수행하는 조인
- Between A And B와 같은 다른 비교연산자를 사용
- ( e.salary between l.losalary and l.highsalary)

### 포괄 조인 ( Outer-Join )



- 조인 대상 테이블 중 같은 데이터가 존재하고 어느한쪽의 테이블에 데이터가 더 많을때 사용하는 것으로 등가조인의 확장판
- (+)연산자를 부족한 쪽 테이블에 사용하여 수행하며 (+)연산자는 부족한 쪽 테이블에 null값을 갖는 행을 삽입하여 수행

### 자체 조인 ( Self-Join )



- 한 개의 테이블을 From절의 Alias 기능을 사용하여 여러 개의 테이블인것처럼 만들어 수행하는 조인 ( from employees e, employees m )
- 자체 조인 내에서 등가, 비등가, 포괄조인이 발생
- 서브쿼리로 대체 가능

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

```
SQL> desc employees
Name
-----
EMPLOYEE_ID (PK)
FIRST_NAME
LAST_NAME
EMAIL
PHONE_NUMBER
HIRE_DATE
JOB_ID
SALARY
COMMISSION_PCT
MANAGER_ID
DEPARTMENT_ID (FK)
```

```
SQL> desc departments
Name
-----
DEPARTMENT_ID (PK)
DEPARTMENT_NAME
MANAGER_ID
LOCATION_ID (FK)
```

```
SQL> desc locations
Name
-----
LOCATION_ID (PK)
STREET_ADDRESS
POSTAL_CODE
CITY
STATE_PROVINCE
```

e.department\_id = d.department\_id and d.location\_id = l.location\_id



## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

- 등가 조인(EQUI JOIN)

모든 사원의 번호, 부서 번호, 부서이름을 출력하시오

```
SELECT  e.employee_id, e.department_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id;
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

급여가 15000 이상인 사원의 이름과 급여, 그 사원이 근무하는 부서이름을 출력하시오.

```
SELECT  e.last_name, e.salary, d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
AND     e.salary >= 15000
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

연봉이 150000 이상인 사원의 이름과 연봉, 그 사원이 근무하는 부서이름과 부서가 위치한 지역번호를 출력하시오. 단 연봉은 AnnSal로 출력하시오

```
SELECT  e.last_name, e.salary*12 "Annsal", d.department_name, d.location_id
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
AND     e.salary*12 >= 150000
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

업무에 MAN이 포함이 되며, 급여가 2500이상인 사원에 대하여 사원번호, 사원 이름, 업무, 급여, 부서명을 출력하고, 급여를 많이 받는 순서대로 출력하시오

```
SELECT  e.employee_id, e.last_name, e.job_id, e.salary, d.department_name
FROM    employees e, departments d
WHERE   e.department_id=d.department_id
AND     job_id = 'MK_MAN' and e.salary >=2500
ORDER BY e.salary desc;
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

Toronto에 근무하는 사원의 이름과 급여, 부서이름과 도시명을 출력하시오

```
SELECT  e.last_name, e.salary, d.department_name, l.city
FROM    employees e, departments d, locations l
WHERE   e.department_id=d.department_id
AND     d.location_id = l.location_id
AND     l.city = 'Toronto';
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

- 비등가 조인(NON-EQUI JOIN)

회사에 근무하는 사원의 이름과 급여와 사원별 급여등급을 출력하시오

```
SELECT  e.last_name, e.salary, j.grade
FROM    employees e, salgrade j
WHERE   e.salary BETWEEN j.losal AND j.hisal;
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

회사에 근무하는 직원중 급여 등급이 4인 직원의 이름과 급여와 직원별 급여등급을 내림차순으로 정렬하여 출력하시오

```
SELECT      e.last_name, e.salary, j.grade
FROM        employees e, salgrade j
WHERE       e.salary BETWEEN j.losal AND  j.hisal
AND         j.grade = 4
ORDER BY    e.salary DESC;
```

## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

- 포괄 조인(OUTER JOIN)

회사에 근무하는 모든사원의 이름과 그 사원이 근무하는 부서번호 및 부서이름을 출력하시오. 단 부서를 배정받지 못한 사원도 추가하시오

```
SELECT  e.last_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id(+);
```



## ❖ SQL SELECT문 – 여러 테이블의 데이터 표시(JOIN)

- 자체 조인(SELF JOIN)

회사에 근무하는 사원의 이름과 그 사원을 관리하는 관리자 이름을 출력하시오

```
SELECT  worker.last_name, manager.last_name  
FROM    employees worker, employees manager  
WHERE   worker.manager_id = manager.employee_id;
```

## ❖ SQL SELECT문 – 서브쿼리

- Select 문장 내에 또라는 Select문장이 포함된 쿼리를 의미
- Group by절을 제외한 Select 절 전체에 사용 가능
- Where절과 having절에서 사용되는 서브쿼리는 조건식의 값으로 사용하며, 특정 값을 모를때 자주 사용  
( where salary > ( 서브쿼리), having avg(salary) > (서브쿼리) )
- 서브쿼리는 괄호안에 삽입
- Where, having절에서 사용되는 서브쿼리는 연산자 오른쪽에 위치
- Where, having절에서 사용되는 서브쿼리에서는 order by 절 사용 자제
- 서브쿼리의 결과가 하나냐 다수이냐에 따라 연산자 종류를 다르게 사용

## ❖ SQL SELECT문 – 서브 쿼리

- 단일 행 서브 쿼리



- 다중 행 서브 쿼리



## ❖ SQL SELECT문 – 서브쿼리 연산자

단일행 서브쿼리 연산자



- 서브쿼리의 결과가 한 개
- = 와 같은 단일연산자를 사용

다중행 서브쿼리 연산자



- In
- 서브쿼리의 결과가 다수일때 다수의 값과 동일한 값을 검색

- > all : 서브쿼리의 결과 전체보다 큰값을 찾을때 사용
- < all : 서브쿼리의 결과 전체보다 작은값을 찾을때 사용

- > any : 서브쿼리의 결과 중 최솟값보다 큰 값을 검색
- < any : 서브쿼리의 결과 중 최댓값보다 작은 값을 찾을때 사용

All과 any연산자는 서브쿼리의 복수개의 결과보다 크거나 작은값을 구하기위해 사용

## ❖ SQL SELECT문 – 서브쿼리 수행 가이드

1. 내가 모르고 있는 값이 무엇인지를 찾자. ( Abel의 급여 )

2. 모르고 있는 특정값을 서브쿼리를 통해 찾자.

- 서브쿼리의 Select list절에 어떤 컬럼을 통해 모르는 특정 값을 찾을지를 결정 ( select salary...)

3. 서브쿼리가 넘겨주는 결과값을 메인쿼리의 어떤 컬럼이 받을것인지를 결정하자

- where salary > ( select salary)

- 반드시 서브쿼리에서 넘겨주는 값과 메인쿼리의 받는 값의 데이터 타입이 같아야 함

- 대부분 서브쿼리와 메인쿼리의 컬럼이름이 같음

4. 서브쿼리의 값과 메인쿼리의 값을 비교하기 위한 연산자를 결정하자

- 서브쿼리의 결과가 하나일 경우는 단일 연산자를, 복수개의 값일 경우에는 in, any, all과 같은 복수연산자를

## ❖ SQL SELECT문 – 서브 쿼리

- 141번 사원과 동일한 업무를 수행하는 사원의 이름과 업무를 출력하시오

```
SELECT      last_name, job_id
FROM        employees
WHERE       job_id = ( SELECT job_id
                        FROM   employees
                        WHERE  employee_id = 141) ;
```

## ❖ SQL SELECT문 – 서브 쿼리

- Ernst와 동일한 부서에 근무하는 직원중 급여가 5000보단 큰 사원의 이름과 급여를 출력하시오

```
SELECT      last_name, salary
FROM        employees
WHERE       department_id = ( SELECT  department_id
                              FROM    employees
                              WHERE   last_name = 'Ernst')
AND         salary > 5000;
```

## ❖ SQL SELECT문 - 서브 쿼리

- 이름에 t를 포함하고 있는 사원과 같은 부서에 근무하는 사원의 이름과 사원번호와 부서번호를 출력하시오

[illegible]



## ❖ SQL SELECT문 – 서브 쿼리

- 최저급여를 받는 사원보다 더 많은 급여를 받는 사원의 이름과 급여를 출력하시오

```
SELECT    last_name, salary
FROM      employees
WHERE     salary > ( SELECT  MIN(salary)
                     FROM    employees) ;
```

## ❖ SQL SELECT문 – 서브 쿼리

- 50번부서의 평균급여보다 더 많은 급여를 받는 사원의 이름과 급여와 부서번호를 출력하시오

```
SELECT      last_name, salary, department_id
FROM        employees
WHERE       salary > ( SELECT  avg(salary)
                        FROM    employees
                        where    department_id=50) ;
```



## ❖ SQL SELECT문 – 서브 쿼리

- IT\_PROG 직업을 가진 사원의 최대급여보다 더 많은 급여를 받는 사원의 번호와 이름과 급여를 출력하시오

```
SELECT  employee_id, last_name, salary
FROM    employees
WHERE   salary > ALL ( SELECT  salary
                        FROM    employees
                        WHERE   job_id = 'IT_PROG')
AND     job_id <> 'IT_PROG';
```

## ❖ SQL SELECT문 – 데이터 그룹

- 그룹함수
  - 그룹별로 한 개 또는 여러 개의 값을 받아들여 함수의 연산을 통해 한 개의 결과값을 출력
  - AVG, SUM -> 숫자 데이터를 보유한 컬럼에만 사용
  - MAX, MIN, COUNT -> 모든 데이터타입에 사용 가능
  - 사용법 : 그룹함수(column) - > avg(salary)
  - AVG, SUM, MAX, MIN, COUNT - > 컬럼을 인수로 받아들일 때는 NULL값 무시
  - COUNT(\*) 사용시 NULL값 인정 - > 행단위 처리
  - NULL값 처리시 NVL함수 중첩사용 -> avg(NVL(commission\_pct, 0))
  - DISTINCT 옵션을 이용하여 중복제거 후 처리 - > count(distinct department\_id)

## ❖ SQL SELECT문 – 데이터 그룹

- GROUP BY

- Select 리스트절에 그룹함수가 사용되었을때, Select 리스트절에 명시된 그룹함수가 사용되지 않은 컬럼을 기준으로 그룹화 수행을 위해 사용
- ```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

## ❖ SQL SELECT문 – 데이터 그룹

100번 부서를 제외한 각 부서별 평균급여가 7,000이상인 부서를 출력하시오

```
SELECT  DEPARTMENT_ID, AVG(SALARY)
FROM    EMPLOYEES
WHERE   DEPARTMENT_ID!=100
GROUP BY DEPARTMENT_ID
HAVING  AVG(SALARY)>=7000;
```

## ❖ SQL SELECT문 – 데이터 그룹

동일한 직업을 가진 직원들의 총 수를 출력하시오

```
SELECT  JOB_ID, COUNT(EMPLOYEE_ID)
FROM    EMPLOYEES
GROUP BY JOB_ID;
```



## ❖ SQL SELECT문 – 데이터 그룹

직원이 4명 이상인 부서의 부서번호와 인원을 출력하시오

```
SELECT  DEPARTMENT_ID, COUNT(EMPLOYEE_ID)
FROM    EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING  COUNT(EMPLOYEE_ID) >= 4;
```

## ❖ SQL SELECT문 – 데이터 그룹

매니저로 근무하는 직원들의 총 수를 출력하시오

```
SELECT COUNT(DISTINCT MANAGER_ID)
FROM EMPLOYEES;
```

# ❖ SQL DML문

## DML



- 데이터베이스에 저장된 데이터에 대한 작업을 수행하는 명령문으로 insert, update, delete로 구성
- 트랜잭션에 가장 많은 영향을 주는 언어로 DB를 운영하기위한 언어임
- DB의 Data는 현재시점만을 가르켜야 함. Data가 현재시점을 반영 할 수 있도록 삽입, 변경, 삭제를 통해 DB를 운영하는 언어

```
Insert into Table_name  
Values ( value, value );
```

```
Update table_name  
Set column = value  
Where column = value;
```

```
Delete From table_name  
Where column=value;
```

# ❖ SQL DDL문

DDL



- DB내에서는 물리적인 구조를 가지고 있는 애들을 객체(table, view, sequence, index)라고 표시함
- 객체들의 구조를 정의하는 언어로 create, alter, drop 명령어를 사용

```
Create Table Table_name  
( Column Datatype(length),  
  Constraint 제약조건이름 제약조건유형(column));
```

```
Create Table dept  
( Deptid number(3),  
  Dname varchar(2) not null,  
  Constraint dept_deptid_pk Primary key(deptid) );
```

## ❖ SQL DDL문

| 데이터 유형                        | 설명                        |
|-------------------------------|---------------------------|
| VARCHAR2( <i>size</i> )       | 가변 길이 문자 데이터              |
| CHAR( <i>size</i> )           | 고정 길이 문자 데이터              |
| NUMBER( <i>p</i> , <i>s</i> ) | 가변 길이 숫자 데이터              |
| DATE                          | 날짜 및 시간 값                 |
| LONG                          | 최대 2GB의 가변 길이 문자 데이터      |
| CLOB                          | 최대 4GB의 문자 데이터            |
| RAW 및 LONG RAW                | 원시 이진 데이터                 |
| BLOB                          | 최대 4GB의 이진 데이터            |
| BFILE                         | 외부 파일에 저장된 이진 데이터(최대 4GB) |
| ROWID                         | 테이블에서 행의 고유 주소를 나타내는 64진수 |

```
ALTER TABLE table  
ADD          (column datatype [DEFAULT expr]  
              [, column datatype]...);
```

```
ALTER TABLE table  
MODIFY       (column datatype [DEFAULT expr]  
              [, column datatype]...);
```

```
ALTER TABLE table  
DROP        (column);
```

```
ALTER TABLE table  
ADD          (column datatype [DEFAULT expr]  
              [, column datatype]...);
```

```
ALTER TABLE table  
MODIFY       (column datatype [DEFAULT expr]  
              [, column datatype]...);
```

```
ALTER TABLE table  
DROP        (column);
```


## ❖ SQL VIEW

- 데이터 액세스를 제한하기 위해
- 복잡한 질의를 쉽게 작성하기 위해
- 데이터 독립성을 제공하기 위해
- 동일한 데이터로부터 다양한 결과를 얻기 위해

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 80;
```



## ❖ SQL TOP-N 분석



```
SELECT ROWNUM as RANK, last_name, salary
FROM (SELECT last_name, salary FROM employees
      ORDER BY salary DESC)
WHERE ROWNUM <= 3;
```

## ❖ SQL 시퀀스

```
CREATE SEQUENCE dept_deptid_seq  
        INCREMENT BY 10  
        START WITH 120  
        MAXVALUE 9999  
        NOCACHE  
        NOCYCLE;
```

```
-  
INSERT INTO departments(department_id,  
        department_name, location_id)  
VALUES    (dept_deptid_seq.NEXTVAL,  
        'Support', 2500);
```

## ❖ SQL DCL

```
CREATE USER user  
IDENTIFIED BY password;
```

```
CREATE USER scott  
IDENTIFIED BY tiger;  
User created.
```

```
GRANT create session, create table,  
      create sequence, create view  
TO    scott;  
Grant succeeded.
```

```
REVOKE select, insert  
ON      departments  
FROM    scott;  
Revoke succeeded.
```