

1. 기본적인 **SQL** 문장

목적

- **SQL SELECT** 문장의 성능을 나열
- 기본적인 **SELECT** 문장을 실행
- **SQL** 문장과 **SQL*Plus** 명령어 사이의 차이점을 구분

SQL SELECT 문장의 성능

- **SELECT** 문장의 데이터 베이스로 부터 정보를 검색
 - **Selection** : 질의에 대해 리턴하고자 하는 테이블의 행을 선택하기 위해 사용
 - **Projection** : 질의에 대해 리턴하고자 하는 테이블의 열을 선택하기 위해 사용
 - **Join** : 공유 테이블의 열에 대해 링크를 생성하여 다른 테이블에 저장되어 있는 데이터를 함께 가져오기 위해 사용

기본적인 **SELECT** 문장

```
SELECT [DISTINCT] {*, column [alias],...}  
FROM table;
```

- **SELECT** 절 : 디스플레이 시킬 열을 나열
- **FROM** 절 : **SELECT** 절에 나열된 열을 포함하는 테이블 명시
- 구문 형식에서
 - **SELECT** : 하나 이상의 열을 나열
 - **DISTINCT** : 중복을 제거
 - * : 모든 열을 선택
 - **Column** : 명명된 열을 선택
 - **Alias** : 선택된 열을 다른 이름으로 변경
 - **FROM table** : 열을 포함하는 테이블을 명시

SQL 문장의 작성

- **SQL** 문장은 대소문자를 구별하지 않음
- **SQL** 문장은 한 줄 이상일수 있음
- 키워드는 단축하거나 줄로 나눌 수 없다
- 절은 대개 줄로 나누어 씀
- 탭과 줄 넣기(indent)는 읽기 쉽게 하기 위해 사용
- 일반적으로 키워드는 대문자로 입력
- **SQL*plus**에서 **SQL** 문장은 프롬프트에 입력되며, 이후의 라인은 번호가 붙음. 이것을 **SQL buffer**라 부름 오직 한 문장이 한 번에 버퍼에 있을 수 있음

모든 열 선택

```
SQL > SELECT *  
2 > FROM dept;
```

- **SQL** 키워드에 *을 포함하여 테이블의 열 데이터 모두를 디스플레이 할 수 있음
- **SELECT** 키워드 이후에 모든 열을 나열하여 테이블의 모든 열을 디스플레이 할 수 있음

특정 열 선택

```
SQL > SELECT deptno, loc  
2 > FROM dept;
```

- 열 이름을 콤마로 구분하여 명시함으로써 테이블의 특정 열을 디스플레이 할 수 있음
- **SELECT** 절에서 원하는 열의 순서대로 결과에 나타나기 원하면 보고자 하는 열을 순서대로 명시

디폴트 열 헤딩

- 디폴트 자리맞춤
 - 좌측 : 날짜와 문자 데이터
 - 우측 : 숫자 데이터
- 디폴트 디스플레이
 - 대문자

산술 표현식

- 산술 연산자를 사용하여 **NUMBER**와 **DATE** 데이터 상의 표현식을 만듦

Operator	Description
+	더하기
-	빼기
*	곱하기
/	나누기

FROM 절을 제외한 SQL 문장의 절에서 산술 연산자 사용가능

산술 연산자 사용

```
SQL > SELECT ename, sal, sal+300  
2 > FROM emp;
```

- 위의 예는 직원의 급여를 **300**달러 증가시키기 위해 덧셈연산자를 사용하고 결과에 **SAL+300** 열을 디스플레이
- 열 **sal+300**은 **dept** 테이블의 새로운 열이 아님

연산자 우선순위

$*$ $/$ $+$ $-$

- 곱하기와 나누기는 더하기와 빼기보다 우선순위가 높다
- 같은 우선순위의 연산자는 좌측에서 우측으로 계산됨
- 괄호는 강제로 계산의 우선순위를 바꾸거나 문장을 명료하게 하기 위해 사용

Null 값 정의

- **null**은 이용할 수 없거나, 지정되지 않았거나, 알 수 없거나 또는 적용할 수 없는 값임
- **Null**은 숫자 **0** 이나 공백과는 다름
- 열이 **NOT NULL**로 정의 되지 않거나, 열이 생성될 때 **PRIMARY KEY**로 정의되지 않았다면, 어떤 데이터형의 열은 **Null** 값을 포함할 수 있음
- 산술 표현식에 있는 어떤 값이 **Null**값이면 결과는 **null**이다
 - 예를 들어 **0**으로 나누기를 시도하면 에러가 발생하지만 **null**로 숫자를 나누면 결과는 **null** 또는 **unknown**임

열 별칭(alias) 정의

- 열 헤딩 이름을 변경
- 계산에 유용
- 열 이름 바로 뒤에 둬. 열 이름과 별칭 사이에 키워드 **AS**를 넣기도 함
- 공백이나 특수문자 또는 대문자가 있으면 이중 인용부호 (“ ”)가 필요함

열 별칭 사용

```
SQL > SELECT ename AS name, sal salary  
2 > FROM emp;
```

```
SQL > SELECT ename "Name", sal*12 "Annual Salary"  
2 > FROM emp;
```

- **AS** 키워드는 열별칭 이름 앞에 사용
- 질의의 결과는 열 헤딩을 대문자로 나타냄

연결연산자

```
SQL > SELECT ename || job AS "Employees"  
2 > FROM emp;
```

- 연결연산자(||)를 사용하여 문자 표현식을 생성하기 위해 다른 열, 산술 표현식 또는 상수 값에 열을 연결 할 수 있음
- 연산자 양쪽에 있는 열은 단일 결과 열을 만들기 위해 조합됨

리터럴(Literal) 문자 스트링

```
SQL > SELECT ename || '||'is a'|| '||job AS "Employee Details"  
2 > FROM emp;
```

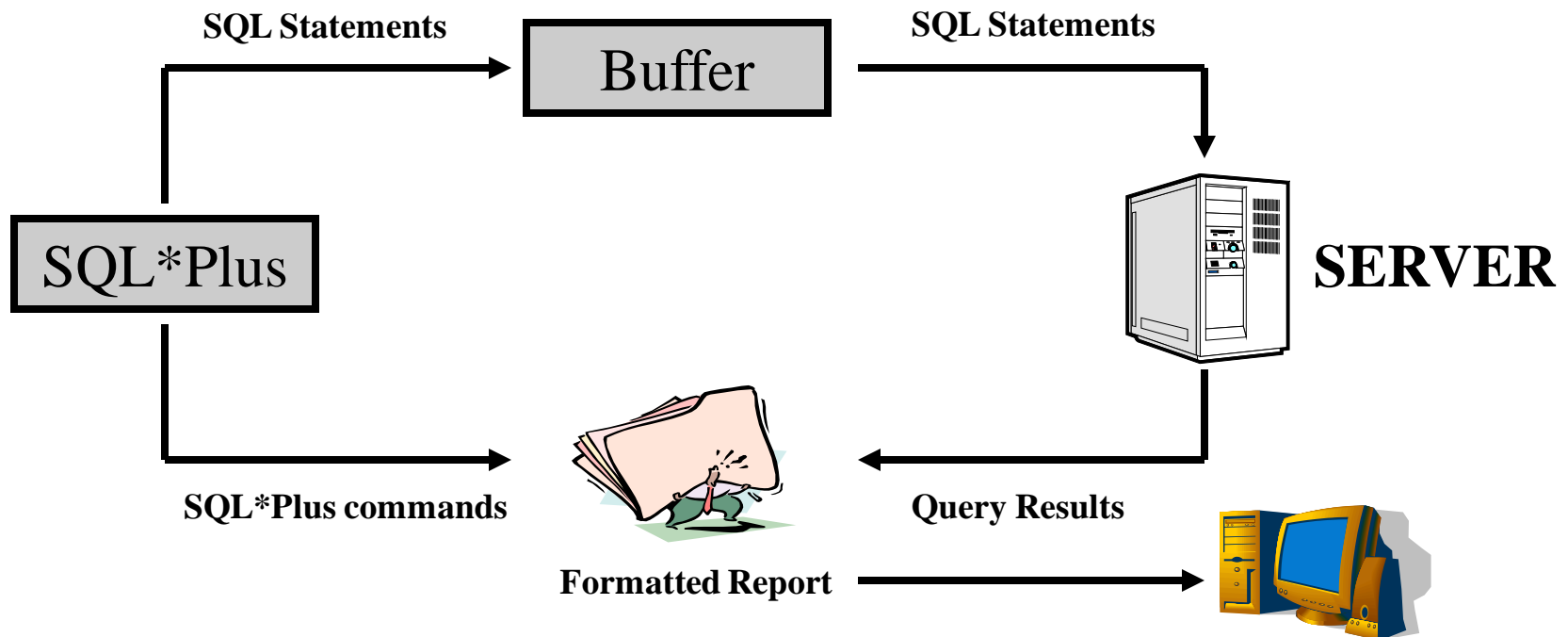
- 리터럴은 열 이름이나 열 별칭이 아닌, **SELECT** 목록에 포함되어 있는 문자, 표현식 또는 숫자임
- 리턴되는 각각의 행에 대하여 출력됨
- 리터럴 스트링은 질의 결과에 포함될 수 있으며 **SELECT** 목록에서 열과 같이 취급됨
- 날짜와 문자 리터럴은 단일 인용부호(' ') 내에 있어야 함

중복행과 중복행의 제거

- 질의의 디폴트 디스플레이는 중복되는 행을 포함하는 모든 행
- **SELECT** 절에서 **DISTINCT** 키워드를 사용하여 중복되는 행을 제거
- **DISTINCT** 키워드 뒤에 여러 개의 열을 명시할 수 있음
- **DISTINCT** 키워드는 선택된 모든 열에 영향을 미치고, 결과는 모든 열의 **distinct**한 조합을 나타냄

```
SQL > SELECT DISTINCT deptno  
2 > FROM emp;
```

SQL과 SQL*Plus의 상호작용



- **SQL**문장 입력 시 **SQL buffer**라는 메모리 한 부분에 저장되어 새로운 문장을 입력할 때 까지 유지됨
- **SQL*Plus**는 **SQL** 문장의 실행을 위해 문장을 인식하고 서버에 **SQL**문장을 보내는 오라클 툴이며 고유의 명령어도 가짐

SQL과 SQL*Plus의 특징

- **SQL의 특징**

- 프로그래밍 경험이 적거나 전혀 없는 사용자도 사용할 수 있다,
- 비 절차적 언어임
- 시스템을 생성하고 유지 관리하기 위해 요구되는 시간을 절약함
- 영어와 비슷한 언어임

- **SQL*Plus의 특징**

- 파일로 부터 **SQL**입력을 받음
- **SQL**문장을 수정하기 위한 라인 편집을 제공함
- 환경적인 설정을 제어함
- 질의 결과를 기본적인 리포트 형태로 포맷
- 로컬과 원격 데이터베이스를 액세스함

SQL 문장과 SQL*Plus 명령 비교

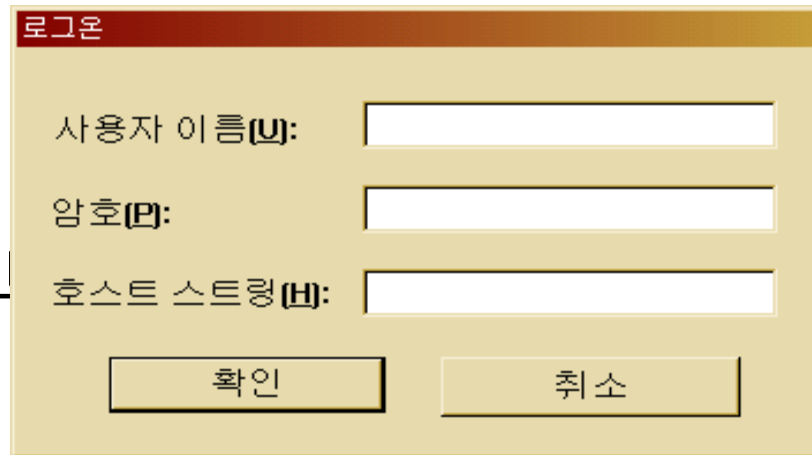
SQL	SQL*Plus
데이터를 액세스 하기위해 오라클 서버와 통신	SQL문장을 인식하고 그들의 서버로 전송
ANSI 표준 SQL을 기본으로 함	SQL 문장을 실행시키기 위한 오라클 소유의 인터페이스
데이터 베이스의 데이터와 테이블을 조작	데이터베이스 값을 조작할 수 있음
하나이상의 라인이 SQL버퍼에 입력됨	한번에 한 라인씩 입력함. SQL버퍼에 저장되지 않음
연속된 문자가 없음	명령어가 한 라인보다 길어지면 연속 문자인 대쉬를 이용
단축될 수 없음	단축 될 수 있음
명령을 즉시 실행하기 위해 종료 문자를 사용	종료문자가 필요 없음.명령어는 즉시 실행됨
어떤 포맷팅을 수행하기 위해 함수를 사용	데이터를 포맷하기 위해 명령어를 사용

SQL*Plus 개요

- **SQL*Plus**로 로그인
- 테이블 구조를 기술
- **SQL** 문장을 편집
- **SQL*Plus**로 부터 **SQL**을 실행
- **SQL** 문장을 파일로 저장하고 **SQL** 문장을 파일에 추가합니다.
- 저장된 파일을 실행
- 편집을 하기 위해서 파일로부터 버퍼로 명령어를 로드함

SQL*Plus 로그인

- 윈도우 환경에서



A screenshot of the SQL*Plus login window. The window has a title bar with the text '로그온' (Login). Inside the window, there are three input fields with labels: '사용자 이름(U):' (Username), '암호(P):' (Password), and '호스트 스트링(H):' (Host String). Below the input fields are two buttons: '확인' (OK) and '취소' (Cancel).

- 명령어 라인

```
sqlplus [username[/password[@database]]]
```

테이블 구조 디스플레이

- 테이블 구조를 디스플레이 하려면 **SQL*Plus DESCRIBE** 명령어를 사용

```
DESC[RIBE] tablename
```

```
SQL> DESCRIBE dept
```

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR(14)
LOC		VARCHAR(13)

- Null? : 열이 데이터를 포함하는 지의 여부를 나타냄
- Type : 열의 데이터 형을 디스플레이

SQL*Plus 편집 명령어

- **SQL*Plus** 명령어는 한번에 한 라인에 입력되며 **SQL**버퍼에 저장않됨.

명령어	설명
A[PEND] text	텍스트를 현재 라인의 끝에 추가함
C[HANGE] / old / new	현재 라인의 old 텍스트를 new 텍스트로 변경함
C[HANGE] / text /	현재 라인의 text를 삭제함
CL[EAR] BUFF[ER]	SQL 버퍼의 모든 라인을 삭제함
DEL	현재 라인을 삭제함
I[NPUT]	무한대의 라인을 삽입
I[NPUT] text	Text로 이루어진 한 라인을 삽입
L[IST]	SQL 버퍼의 모든 라인을 나열
L[IST] n	한 라인(n으로 명시된)을 나열함
L[IST] m n	일정범위의 라인(m n으로 명시된)을 나열함
R[UN]	버퍼의 SQL문장을 디스플레이 하고 실행함
N	현재 라인으로 만들기 위해서 라인을 명시
N text	라인 n을 text 로 대체
O text	첫째 라인 앞에 한 라인을 삽입

SQL*Plus 파일 명령어

명령어	설명
SAV[E] filename[.ext] [REP[LACE]APP[END]]	SQL 버퍼의 현재 내용을 파일로 저장함. 기존의 파일에 추가하려면 APPEND를 사용. 기존파일에 덮어쓰려면 REPLACE를 사용
GET filename[.ext]	이전에 저장된 파일의 내용을 SQL버퍼에 기록함
STA[RT] filename[.ext]	이전에 저장된 명령파일을 실행
@filename	이전에 저장된 명령파일을 실행
ED[IT]	편집기를 호출하고 afiedt.buf라는 파일로 버퍼내용을 저장
ED[IT]filename[.ext]	저장된 파일의 내용을 편집하기 위해서 편집기를 호출
SPO[OL] [filename[.ext]] OFF[OUT]	질의 결과를 파일에 저장 OFF는 스푼 파일을 닫음. OUT는 스푼 파일을 닫고 시스템 프린터 결과를 전송
EXIT	SQL*Plus를 종료함

요약

```
SELECT [DISTINCT] {*, column [alias],...}  
FROM table;
```

- 다음을 위해서 **SQL*Plus**를 사용함
 - **SQL**문장을 실행
 - **SQL**문장을 편집