

The background features a row of 15 cubes of varying heights, with the tallest cube in the center. A large circle is centered over the text area. The text is contained within a rectangular frame that is partially covered by the circle.

# Chapter 05

## SQL 언어

데이터베이스 응용

## □ SQL 언어

### ● SQL(Structured Query Language)

- \* 1974년 IBM 연구소에서 발표된 SEQUEL(Structured English QUery Language)에서 유래
- \* 특징
  - ☑ SQL은 비절차적인
  - ☑ 대화식 언어로 사용 가능
  - ☑ 다른 종류의 범용 프로그래밍 언어로 작성된 프로그램에 내장(embed)시킨 형태로도 사용 가능
  - ☑ 각각의 튜플 단위가 아니라 튜플들을 집합 단위로 처리

SQL의 특징

관계대수와 관계해석을 기초로 한 고급 데이터 언어
이해하기 쉬운 형태로 표현
대화식 질의어로 사용 가능
데이터 정의, 데이터 조작, 제어 기능 제공
C, C++, Java 등의 언어에 삽입
레코드 집합 단위로 처리
비절차적 언어

## □ SQL 언어

### ● SQL의 명령어

- ※ **DDL (Data Definition Language) : 데이터베이스 및 테이블의 구조를 정의하거나 변경**

SQL문	내 용
CREATE	데이터베이스 및 객체 생성
DROP	데이터베이스 및 객체 삭제
ALTER	기존에 존재하는 데이터베이스 객체를 변경

- ※ **DML (Data Manipulation Language) : 데이터의 삽입, 삭제, 검색과 수정 등을 처리**

SQL문	내 용
INSERT	데이터베이스 객체에 데이터를 입력
DELETE	데이터베이스 객체에 데이터를 삭제
UPDATE	기존에 존재하는 데이터베이스 객체안의 데이터 수정
SELECT	데이터베이스 객체로부터 데이터를 검색

## □ *SQL 언어*

### ※ DCL (Data Control Language) : 데이터베이스 사용자의 권한을 제어

SQL문	내 용
GRANT	데이터베이스 객체에 권한 부여
REVOKE	이미 부여된 데이터베이스 객체의 권한 취소

## □ SQL 언어

### ● 데이터 정의어(Data Definition Language)

#### \* 특징

- ☑ 데이터를 정의하는 데 사용
- ☑ 데이터베이스를 생성하거나 데이터베이스 객체들을 생성하고, 변경/제거하기 위해 사용하는 명령문들
- ☑ CREATE, ALTER, DROP

#### \* 데이터베이스 구조 생성하기

```
CREATE DATABASE 데이터베이스_이름 ;
```

#### \* 도메인 정의하기

```
CREATE DOMAIN 도메인_이름 데이터 타입  
[ 기본값 선언 ]  
[ 도메인_제약조건_목록 ] ;
```

## □ SQL 언어

### ● Oracle 10g에서 지원하는 데이터 타입 (1/2)

DATA TYPE	설 명
SMALLINT	소수점을 포함하지 않는 -99999 ~ 999999까지의 정수
INTEGER, INT	-99999999999 ~ 99999999999까지의 정수
DECIMAL(p, s), NUMBER(p, s)	전체 p(최대 38) 자리 중 소수점 이하 s(음수 가능)자리, (ex) NUMBER(*,-2)
LONG	가변길이 공간에 문자 데이터 저장 (2G bytes 또는 $2^{31}-1$ bytes)
RAW(size)	가변길이의 raw 이진 데이터 저장 (2,000 bytes 까지)
LONG RAW	2G bytes까지 가변길이의 raw 이진 데이터 저장 공간
BINARY_FLOAT	32-bit 부동소수점수, 5 bytes 공간
BINARY_DOUBLE	64-bit 부동소수점수, 9 bytes 공간
DATE	날짜 데이터

## □ SQL 언어

### ● Oracle 10g에서 지원하는 데이터 타입 (2/2)

DATA TYPE	설 명
CHAR(n)	n bytes(또는 문자수) 고정길이 문자열(1 ~ 2000 bytes) CHAR(10 CHAR) : DB의 문자셋에 따른 10개 문자수
NCHAR(n)	해당 DB의 문자셋에 맞는 n개 문자의 고정길이 문자열(최대 2,000 bytes)
VARCHAR(n), VARCHAR2(n)	n bytes(또는 문자수) 가변길이 문자열(1 ~ 4000 bytes)
NVARCHAR(n), NVARCHAR2(n)	해당 DB의 문자셋에 맞는 n개 문자의 가변길이 문자열(1 ~ 4000 bytes)
CLOB, NCLOB	Character Large Object의 약자. 최대 128 TB의 크기
BLOB	Binary Large Object의 약자. 최대 128 TB의 크기
BFILE	O/S 파일시스템에 저장된 이진 데이터에 대한 위치값을 저장

## □ SQL 언어

### \* 테이블 생성하기

```
CREATE TABLE 테이블_이름
( 열_이름_1 데이터타입 [ NOT NULL][DEFAULT 값],
  ...
  열_이름_n 데이터타입 [ NOT NULL][DEFAULT 값]
  [ PRIMARY KEY ( 열_이름_1, [ 열_이름_2, ..., 열_이름_n ],
  [ UNIQUE ( 열_이름_1, [ 열_이름_2, ..., 열_이름_n ],
  [ FOREIGN KEY ( 열_이름_1, [ 열_이름_2, ..., 열_이름_n ],
    REFERENCES 기본테이블 [ (열_이름_1, ... , 열_이름_n) ]
    [ ON DELETE 옵션 ]
    [ ON UPDATE 옵션 ],
  [ CONSTRAINT 제약조건_이름 ] [CHECK ( 조건식 ) ] ),
  [ TABLESPACE 테이블스페이스_이름 ] ) ;
```



## □ SQL 언어

### ● 테이블 생성시 제한 사항과 고려할 점

- \* 테이블 이름과 컬럼은 항상 알파벳 문자로 시작해야 하며 A~Z까지의 문자, 0~9까지의 숫자, 그리고 \$, #, \_ (Under Bar)를 사용할 수 있다. 그러나 공백은 사용할 수 없다.
  - ☑ 테이블 또는 컬럼 이름에 한글 사용 가능 : 다른 응용 프로그램과 호환성 조심
- \* 테이블의 컬럼 이름은 30자를 초과할 수 없고, 예약어를 사용할 수 없다.
- \* 한 테이블 안에서 컬럼 이름은 같을 수 없으며 다른 테이블에서의 컬럼 이름과는 같을 수 있다.
- \* 각 컬럼의 데이터 타입은 꼭 지정되어야 한다.
- \* 각 컬럼들은 콤마", "로 구분되고, Create 명령문은 세미콜론"; "으로 끝난다.

```
CREATE TABLE EMP
( emp_no    SMALLINT    NOT NULL,
  emp_name  CHAR(20)
    CONSTRAINT emp_name_nn NOT NULL,
  salary    INTEGER
    CONSTRAINT emp_salary_nn NOT NULL,
  PRIMARY KEY ( emp_no ),
  CONSTRAINT emp_salary_min CHECK (salary > 0),
  CONSTRAINT emp_email_uk  UNIQUE (email)
);
```

## □ SQL 언어

### \* 테이블 변경

☑ ALTER TABLE 문을 사용하여 변경

☑ 기능

- ✍ 새로운 컬럼 추가
- ✍ 현재 존재하는 컬럼에 대한 기본값을 새로 정의
- ✍ 현재 존재하는 컬럼에 대한 기본값 삭제
- ✍ 존재하는 컬럼 삭제
- ✍ 기본 테이블에 대한 새로운 무결성 제약 조건 명시
- ✍ 기본 테이블에 존재하는 무결성 제약 조건 삭제, 동작, 비동작

```
ALTER TABLE 테이블_이름  
ADD 속성_이름 속성_데이터타입 ;
```

컬럼 추가

```
ALTER TABLE Movie  
DISABLE CONSTRAINT Movie_FK ;
```

외래키 제약 끄기

```
ALTER TABLE EMP  
DROP COLUMN EMP_ADDRESS CASCADE CONSTRAINTS ;
```

컬럼 삭제

## □ SQL 언어

### ● 테이블 삭제

- ✱ `drop table members;` // 외래키 제약 때문에 삭제 불가
- ✱ `drop table members cascade constraints;` // 제약 조건 무시하고 삭제

## □ SQL 언어

### ● 데이터 조작용어(Data Manipulation Language)

※ SQL 명령어 : SELECT, INSERT, DELETE, UPDATE

※ 데이터 삽입(INSERT)

```
INSERT INTO 테이블_이름( 컬럼_이름_1, ... , 컬럼_이름_n )  
VALUES ( 컬럼값_1, ... , 컬럼값_n ) ;
```

```
INSERT INTO 테이블_이름( 컬럼_이름_1, ... , 컬럼_이름_n )  
SELECT 문 ;
```

☑ 레코드의 직접 삽입

```
INSERT INTO EMP( EMP_NO, EMP_NAME, SALARY, DEPT_CODE )  
VALUES ( 100, '홍길동', 1000000 , 111 ) ;
```

## □ *SQL 언어*

### ☑ **부속 질의문을 이용한 레코드 삽입**

```
INSERT INTO COMPUTER( STD_NO, SDT_NAME, GRADE )  
  SELECT STD_NO, SDT_NAME, GRADE  
    FROM STUDENT  
   WHERE DEPT_NAME = '컴퓨터공학부' ;
```

## □ SQL 언어

### \* 행 데이터 변경하기

```
UPDATE 테이블_이름  
    SET 컬럼_이름 = 컬럼_값 [, 컬럼_이름 = 컬럼_값, .. ]  
    [ WHERE 조건식 ] ;
```

#### ☑ 직접 컬럼의 값 변경하기

```
UPDATE EMP  
    SET DEPT_CODE = '333'  
    WHERE EMP_NO = 200 ;
```

#### ☑ 부속 질의를 사용하여 변경하기

```
UPDATE EMP  
    SET SALARY = SALARY + 10000  
    WHERE DEPT_NO IN  
        ( SELECT DEPT_NO  
          FROM DEPT  
          WHERE DEPT_NAME = 'MANAGER' OR DEPT_NAME = 'SALE' ) ;
```

## □ SQL 언어

### \* 행 삭제하기

```
DELETE FROM 테이블_이름  
[ WHERE 조건식 ] ;
```

☑ 조건식에 명시된 조건을 만족하는 레코드 삭제

```
DELETE FROM EMP  
WHERE DEPT_NO = 333 ;
```

### \* TRUNCATE : DELETE와 동일하나 rollback이 불가능 함

```
TRUNCATE TABLE Movie ;
```

## □ SQL 언어

### \* 데이터 검색하기

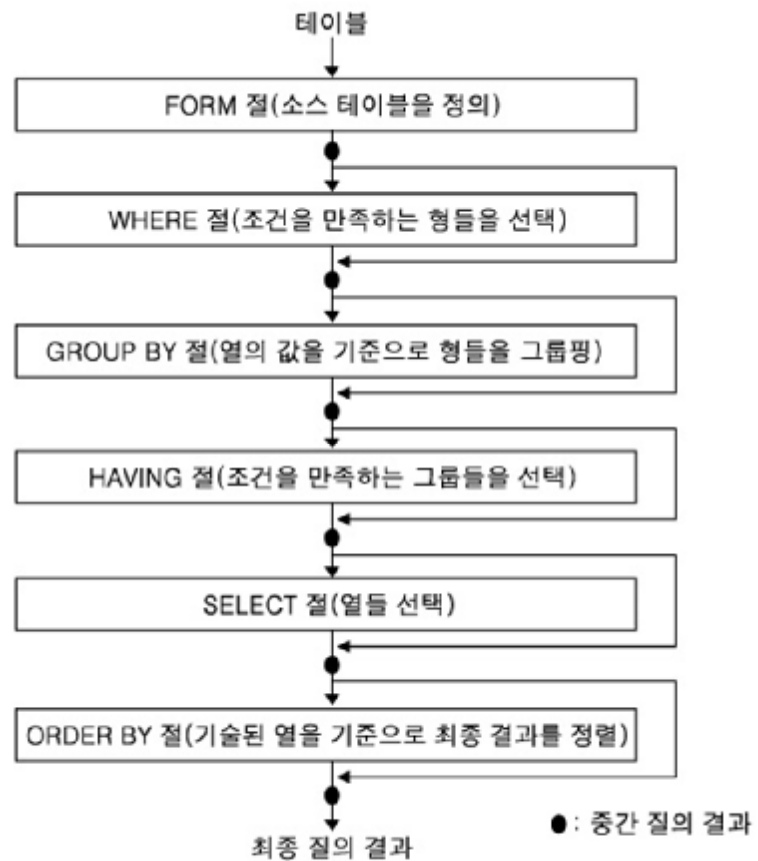
#### ☑ SELECT 문의 구조

종류	기능
SELECT 절	SELECT 문에 의해 검색될 데이터 항목, 즉, 열 이름들을 기술한다.
FROM 절	데이터를 검색할 테이블(소스 테이블)들을 기술한다.
WHERE 절	질의 결과에 포함될 행들이 만족해야 할 조건을 기술한다.
GROUP BY 절	그룹 질의를 기술할 때 사용. 소스 테이블의 각 행에 대해 질의 결과를 각각 생성하는 것이 아니고, 특정 열에 동일한 값을 갖는 행들을 그룹화한 후 각 그룹에 대해 한 개 행씩 질의 결과를 생성한다.
HAVING 절	GROUP BY 절에 의해 구성된 그룹들에 대해 적용할 조건을 기술한다.
ORDER BY 절	특정 열의 값을 기준으로 질의 결과를 정렬할 때 사용한다.



## □ SQL 언어

### ☑ SELECT 문의 수행 과정



## □ SQL 언어

### ☑ 단순 질의

#### ✍ 테이블의 전체 열 검색하기

```
SELECT * FROM EMP ;
```

#### ✍ 중복하는 행 제거하기(DISTINCT)

질의 : 대리점이 개설된 모든 도시 이름을  
검색하시오.

```
SELECT City_name FROM Shop ;
```

(질의 결과)

city\_name

-----

서울

서울

서울

광주

광주

대전

대전



질의 : 대리점이 개설된 모든 도시 이름을  
검색하시오.

```
SELECT DISTINCT City_name  
FROM Shop ;
```

(질의 결과)

city\_name

-----

서울

광주

대전

## □ SQL 언어

### ● 검색 조건 사용하기

- ※ 연산자들을 사용함으로써 특정 조건을 만족하는 데이터에 대한 검색을 효과적으로 수행
- ※ 연산자의 종류

종류	연산자	설명
비교 연산자	=, <>, <, <=, >, >=	값의 크기를 비교하여 질의 검색
범위 연산자	BETWEEN	검사값이 기술된 두 값 사이에 속하는지를 검사하여 질의 처리
리스트 연산자	IN	검사값이 주어진 값의 리스트에 속하는지 여부를 검사
패턴매칭 연산자	LIKE	문자열의 일부가 같은 데이터를 검색할 때 사용
논리연산자	AND, OR, NOT	AND, OR, NOT을 사용하면 단순 탐색 조건을 결합하여 질의 처리

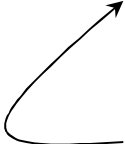
## □ SQL 언어

### ● 질의 결과 정렬하기

- \* SELECT 문의 ORDER BY 절을 사용하여 질의 결과를 정렬
- \* ORDER BY 절은 키워드 ORDER BY와 정렬할 컬럼과 정렬 순서를 정의하여 명시
- \* ORDER BY 절을 명시하지 않으면 ASC가 기본값.

```
SELECT 검색할_컬럼_이름  
FROM 검색할_테이블_이름  
[ WHERE 조건식 ]  
[ ORDER BY 컬럼_이름 [ ASC | DESC ] ] ;
```

```
SELECT *  
FROM STUDENT  
ORDER BY DEPT_CODE ASC, DEPT_NAME DESC ;
```



STUDENT 테이블의 DEPT\_CODE에 대해  
오름차순 정렬을 수행하고 다시 DEPT\_NAME 컬럼에  
대해 2차 정렬을 수행

## □ SQL 언어

### ● 문자열 비교

- \* 문자열 비교는 사전식(lexicographic) 순서 (예: 'at' < 'bar') 에 기반을 두고 있다.
- \* s LIKE p : 간단한 패턴 부합(match)을 기반으로 한 문자열 비교
  - ☑ s는 문자열이며 p는 패턴이다.
  - ☑ NOT LIKE도 가능
  - ☑ 패턴: 특수 문자인 %와 \_을 선택적으로 가진 문자열
    - ✍ % : 0 이상의 길이를 가진 임의의 문자열
    - ✍ \_ : 임의의 한 문자

[예] 제목이 "Star"로 시작하는 모든 영화를 찾아라.

```
SELECT title
FROM Movie
WHERE title LIKE 'Star _ _ _ _'; /* 또는, 'Star%' */
```

## □ SQL 언어

[예] 제목에 소유격('s)을 갖는 모든 영화를 찾아라.

```
SELECT title
```

```
FROM Movie
```

```
WHERE title LIKE '%''s%';
```

```
/* {Logan's Run, Alice's Restaurant}등의 결과가 나옴 */
```

연속되는 두개의 어포스트로피는 하나의 어포스트로피를 나타낸다.

## ● LIKE 수식에서 이스케이프(escape) 문자

\* 키워드 ESCAPE와 사용하고자 하는 이스케이프 문자

[예] s LIKE 'x%%x%' ESCAPE 'x'

/\* %로 시작해서 %로 끝나는 모든 문자열 \*/

## □ SQL 언어

### ● SQL 실습

- \* 100~150분 사이의 상영시간을 갖는 영화 제목, 개봉년도, 영화사 이름은?
  - ☑ 영화사 이름으로 정렬
- \* 95 또는 124분의 상영시간을 갖는 영화 제목과 개봉년도는?
- \* 영화 제목에 'and'가 들어간 영화 제목과 개봉년도는?
- \* 'california'에 사는 영화배우 이름과 주소는 ?

## □ SQL 언어

### ● 요약(aggregation) 질의

- \* 요약 정보에 대한 질의를 지원하기 위해서 6개의 그룹 함수와 GROUP BY 절 사용
- \* 그룹 함수란
  - ☑ 데이터베이스에서 검색된 데이터를 요약하기 위해 사용하는 함수로 다음과 같은 요약 정보 생성

함수 종류	설명
합계 계산 함수(SUM)	단일 열에 속하는 값들의 합계
평균 계산 함수(AVG)	단일 열에 속하는 값들의 평균
최소값/최대값 계산 함수(MIN / MAX)	한개 열에 속하는 값들 중에서 각각 가장 작은 값과 가장 큰 값
값의 개수 계산 함수(COUNT)	열에 저장된 데이터 값들의 개수를 계산
행의 개수 계산 함수(COUNT (*))	"행의 개수"를 계산하는 함수



## □ SQL 언어

### ☑ 예

그룹 함수를 사용한  
연산식을 포함하거나,  
하나의 연산식에  
다수의 그룹 함수  
포함 가능

질의 : 서울에 위치한 대리점은 모두 몇 개인가?

```
SELECT COUNT (*)  
FROM SHOP  
WHERE City_name = '서울'
```

(질의 결과)

```
COUNT(*)  
-----  
3
```

```
SELECT AVG(( Total / cnt_num ) * 100)  
FROM Salesmans  
WHERE StoreCode = 12
```

(질의 결과)

```
AVG(( Total / cnt_num) * 100)  
-----  
89.162543
```

## □ SQL 언어

### ● 그룹 질의(Group by 절)

- \* 요약 질의 결과를 그룹별로 분류하여 요약하고자 하는 경우, GROUP BY 절을 사용하여 질의 결과 요약 가능
- \* GROUP BY 절을 포함하는 질의문은 소스 테이블로부터 데이터 값을 기준으로 행들을 그룹핑하고, 각 행 그룹에 대해 결과로서 한 개의 요약 행 생성.
- \* 예 : 지역별 평균값 생성

```
SELECT City_name, AVG( prices )  
FROM SHOP  
GROUP BY City_name ;
```

(질의 결과)

City_name	AVG(prices)
대전	90000
서울	345000
청주	575000

## □ SQL 언어

### ● Having 절을 사용한 검색

※ 행 집단을 선택하기 위해 사용

※ 예 : 판매원들의 판매량을 대리점별로 그룹지어 합계를 계산하고 판매 금액이 1000000 이상인 결과만을 검색

```
SELECT Store_Code, SUM( prices )  
FROM Salesmans  
GROUP BY Store_Code  
HAVING SUM( prices ) > 1000000 ;
```

(질의 결과)

Store_Code	SUM(Sales)
A_11	9100000
A_12	3500000
B_14	4360000

## □ SQL 언어

### ● SQL 실습

- \* 각 영화사가 제작한 영화의 수와 평균 상영시간은?
- \* 1970년 이전에 처음으로 영화를 제작한 적이 있는 영화사와 제작 영화 편수는?
- \* 같은 주소를 갖는 제작자들의 수와 그 주소는?

### ● ROLLUP과 CUBE의 활용

- \* ROLLUP : 각 그룹에 대한 통계값 계산

```
SELECT studioname, year, incolor, TO_CHAR(AVG(length), '099'), COUNT(*)  
FROM movie  
GROUP BY ROLLUP(studioname, year, incolor);
```

- \* CUBE : 각 그룹에 대해 가능한 모든 조합에 대한 그룹별 통계값 계산

```
SELECT studioname, year, incolor, TO_CHAR(AVG(length), '099'), COUNT(*)  
FROM movie  
GROUP BY CUBE(studioname, year, incolor);
```

## □ SQL 언어

STUDIONAME	YEAR	INC	평균상영	COUNT(*)
fox	1939	t	222	1
fox	1939		222	1
fox	1969	t	110	1
fox	1969		110	1
fox	1977	t	124	1
fox	1977		124	1
fox	1986	t	137	1
fox	1986		137	1
fox	1994	t	115	2
fox	1994		115	2
fox	1996	t	137	1
fox	1996		137	1
fox			137	7
mgm	2000	t	107	1
mgm	2000		107	1
mgm	1973	t	129	1
mgm	1973		129	1
mgm	1980	t	222	1
mgm	1980		222	1
mgm	1985	t	150	1
mgm	1985		150	1
mgm	1994	t	099	1
mgm	1994		099	1
mgm	1995	t	107	2

ROLLUP

warner bros	1997	t	136	1
warner bros	1997		136	1
warner bros			127	2
new york film	1970	t	088	1
new york film	1970		088	1
new york film			088	1
neue const film	1993	t	140	1
neue const film	1993		140	1
neue const film			140	1
touchstone pictures	1996	t	124	1
touchstone pictures	1996		124	1
touchstone pictures			124	1
			135	28

## □ SQL 언어

**CUBE**

STUDIONAME	YEAR	INC	평균상영	COUNT(*)
mgm	1994		099	1
mgm	1994	t	099	1
mgm	1995		107	2
mgm	1995	t	107	2
disney			169	3
disney		t	169	3
disney	1990		119	1
disney	1990	t	119	1
disney	1991		274	1
disney	1991	t	274	1
disney	2002		113	1
disney	2002	t	113	1
paramount			131	6
paramount		t	131	6
paramount	1970		099	1
paramount	1970	t	099	1
paramount	1981		115	1
paramount	1981	t	115	1
paramount	1983		232	1
paramount	1983	t	232	1
paramount	1984		118	1
paramount	1984	t	118	1
paramount	1989		127	1
paramount	1989	t	127	1

## □ SQL 언어

### ● 질의들의 합집합, 교집합, 차집합

- \* SQL은 관계 대수의 합집합, 교집합, 차집합 연산을 제공한다.
- \* 사용되는 키워드는 UNION, INTERSECT, EXCEPT(MINUS) 이다.

[예1] 재산이 \$10,000,000보다 많고 영화 임원인  
모든 여자 스타들의 이름과 주소를 찾아라.

```
(SELECT name, address
FROM MovieStar
WHERE gender = 'F' )
INTERSECT
(SELECT name, address
FROM MovieExec
WHERE netWorth > 10000000);
```

[예2] 영화제작자가 아닌 스타  
들의 이름과 주소는 ?

```
SELECT name, address
FROM MovieStar
MINUS
SELECT name, address
FROM MovieExec;
```

Oracle에서는 괄호 불필요



## □ SQL 언어

### ● ANSI 조인

- \* SQL 국제표준에 따른 조인들 : CROSS JOIN, INNER JOIN, NATURAL JOIN

### ● CROSS JOIN

- \* cartesian product 실행
- \* SELECT \* FROM movie CROSS JOIN movieexec;  
☑ SELECT \* FROM movie, movieexec;

### ● INNER JOIN

- \* 일반적인 세타 조인  
SELECT e.name, s.name  
FROM movieexec e INNER JOIN moviestar s ON e.address = s.address;  
  
SELECT e.name, s.name  
FROM movieexec e, moviestar s  
WHERE e.address = s.address;

## □ SQL 언어

### ● INNER JOIN에서 USING의 사용

- \* 조인할 애트리뷰트의 지정

```
SELECT e.name, s.name, address
```

```
FROM movieexec e INNER JOIN moviestar s USING (address);
```

### ● NATURAL JOIN

- \* 같은 이름의 애트리뷰트들을 기준으로 조인

```
SELECT name, address
```

```
FROM movieexec NATURAL JOIN moviestar;
```

- \* 특정 애트리뷰트를 기준으로 조인 (JOIN~ON 사용)

```
SELECT s.name, address
```

```
FROM studio s JOIN moviestar USING (address);
```

### ● OUTER JOIN

- \* 조인되지 않는 컬럼값들도 NULL로 결과에 포함

```
SELECT e.name, s.name, address
```

```
FROM movieexec e FULL OUTER JOIN moviestar s USING (address);
```

**ON e.address = s.address**



## □ SQL 언어

- 부질의는 릴레이션을 결과로 생성하는 수식이다.

### ● 스칼라 값을 생성하는 부질의

- \* select-from-where 문에 의해 하나의 값만 생성될 때 그 문장은 하나의 상수처럼 사용될 수 있다.

[예] Star Wars의 제작자를 찾아라.

```
SELECT  name
FROM    MovieExec
WHERE   cert# = (SELECT producerC#
                  FROM Movie
                  WHERE title = 'Star Wars');
```

- subquery : 임의의 튜플들을 생성
- 한 개 이상의 값이 생성된다면?

## □ SQL 언어

### ● 릴레이션이 비교 대상이 되는 조건

- SQL에는 부울 값을 결과로 생성하는 연산자들이 있다.

R은 릴레이션을 s는 스칼라 값을 나타낸다고 하자.

\*EXISTS R : R에 튜플이 하나라도 존재하면(iff) 참  $\leftrightarrow$  NOT EXISTS R

\*s IN R : s가 R에 있는 값 중 어느 하나와 일치하면(iff) 참  $\leftrightarrow$  NOT IN R

\*s > ALL R : s가 단항 릴레이션 R의 모든 값보다 크면(iff) 참  $\leftrightarrow$  s <= ANY R

☑ “>” 연산자 대신 다른 비교 연산자가 사용될 수 있다.

☑ s <> ALL R  $\equiv$  s NOT IN R

\*s > ANY R : s가 단항(unary) 릴레이션 R의 값 중 적어도 하나보다 크면(iff) 참  
 $\leftrightarrow$  s <= ALL R

☑ “>” 연산자 대신에 다른 비교 연산자도 사용될 수 있다.

☑ s = ANY R  $\equiv$  s IN R

☑ NOT s > ANY R : s는 R에 있는 모든 값 보다 작거나 같다. 즉, R에서 최소값

\*EXISTS와 IN은 프레디케이트인 반면 ALL과 ANY(또는 SOME)은 정량자(quantifier)

## □ SQL 언어

[예] Movie(title, year, length, inColor, studioName, producerC#)  
StarsIn(movieTitle, movieYear, starName)  
MovieExec(name, adress, cert#, netWorth)

- Harrison Ford가 출연한 영화의 제작자를 찾아라.

```
SELECT name
FROM   MovieExec
WHERE  cert# IN
      (SELECT producerC#
       FROM   Movie
       WHERE  (title, year) IN
             (SELECT movieTitle, movieYear
              FROM   StarsIn
              WHERE  starName = 'Harrison Ford'));
```

Harrison Ford가  
출연한 영화들

Harrison Ford가  
출연한 영화의 제작자들

## ● EXISTS의 사용

```
SELECT    name
FROM      MovieExec
WHERE     EXISTS
          (SELECT  *
           FROM    Movie
           WHERE   cert# = producerC# AND
                  (title, year) in (SELECT  movieTitle, movieYear
                                   FROM      StarsIn
                                   WHERE     starName = 'Harrison Ford'))
);
```

## □ SQL 언어

### ● SQL 실습 과제

- ① ‘paramount’ 영화사에서 제작한 영화 제목과 개봉년도, 출연한 배우이름과 배우의 주소는 ?
- ② 상영시간이 100분 이상인 영화를 제작한 제작자 이름과 주소는?
- ③ 영화 제목, 개봉년도, 영화사 이름, 영화사 사장, 제작자 이름은?
- ④ 각 영화사가 각 해에 제작한 영화의 수와 평균 상영시간은?
- ⑤ 1980년 이전에 첫 영화에 출연했던 배우 이름과 이 배우가 출연했던 영화 편수, 평균 상영시간은?
- ⑥ “new york”에 사는 배우가 출연했던 영화 제목과 제작자 이름은?
- ⑦ 4000000이상의 재산을 가진 각 제작자별 제작한 영화의 평균 상영시간과 영화 편수는?

## □ SQL 언어

### ● 뷰

#### \* 정의

- ☑ 하나 이상의 기본 테이블로부터 유도되어 만들어진 가상 테이블
- ☑ 기본 테이블이 물리적으로 구현되어 실제로 데이터를 저장하는 데 비해 뷰는 물리적으로 구현되지 않고, 뷰의 정의만 시스템 내에 저장해 두었다가 필요 시 실행 시간에 테이블 생성.
- ☑ 기본적으로 기본 테이블로부터 유도되지만 정의된 뷰가 다른 뷰에 의해 사용될 수도 있다.

#### \* 특징

- ☑ 뷰가 정의된 기본 테이블이 제거되면, 뷰의 사용이 불가능하게 됨.
- ☑ 외부 스키마는 뷰와 기본 테이블의 정의로 구성.
- ☑ 뷰에 대한 검색은 기본 테이블과 거의 동일하게 사용되지만 삽입, 삭제, 갱신은 제약을 받는다.
- ☑ 뷰는 CREATE 문에 의해 정의되며, SYSVIEWS에 저장된다.
- ☑ 뷰에 대한 정의는 ALTER 문을 이용하여 변경할 수 없다.
- ☑ 뷰는 DROP 문을 사용하여 제거할 수 있다.



## □ SQL 언어

### \* 장단점

장점	단점
논리적 독립성 제공	독자적인 인덱스를 가질 수 없다.
데이터 접근 제어로 보안 가능	정의를 변경할 수 없다.
사용자의 데이터 관리를 간단하게 함	삽입, 삭제, 갱신 연산에 많은 제약이 따른다.
하나의 테이블로 여러 개의 상이한 뷰를 정의	

## □ SQL 언어

### ● 뷰 생성하기 \* 문법

```
CREATE VIEW 뷰_이름 [ ( 컬럼_이름_1, ... , 컬럼_이름_n ) ]  
    AS SELECT_문  
    [ WITH CHECK OPTION ] ;
```

☑ 예 : 학생(STUDENT) 테이블로부터 컴퓨터공학부 학생들만으로 뷰(STD\_VIEW)를 구성

```
CREATE VIEW STD_VIEW ( STD_NO, STD_NAME, GRADE )  
    AS SELECT STD_NO, STD_NAME, GRADE  
        FROM STUDENT  
        WHERE DEPT_NAME = '컴퓨터공학부'  
    WITH CHECK OPTION ;
```

또는

```
CREATE VIEW STD_VIEW  
    AS SELECT STD_NO, STD_NAME, GRADE  
        FROM STUDENT  
        WHERE DEPT_NAME = '컴퓨터공학부'  
    WITH CHECK OPTION ;
```

## □ SQL 언어

- CREATE TABLE 문을 이용하여 정의된 릴레이션은 실제로 DB 내에 존재하는 반면 뷰(view)는 물리적으로 존재하지 않는다.

### ● 뷰의 선언

```
CREATE VIEW <뷰 이름> AS <뷰 정의>
```

\* <뷰 정의>는 하나의 질의다.

```
CREATE VIEW ParamountMovie AS  
  
    SELECT title, year  
  
    FROM Movie  
  
    WHERE studioName = 'Paramount';
```

- ☞ 기본(base) 테이블(기본 릴레이션) : 실제로 튜플이 저장된 테이블
- ☞ 뷰/가상(virtual) 테이블 : 실제로 튜플이 존재하지는 않고 기본 테이블 또는 다른 뷰를 기반으로 SQL 질의 형태로 정의(defenition)가 저장된 가상 테이블

## □ SQL 언어

### ● 뷰에 대한 질의

- \* 뷰에 대한 질의가 주어지면 해당 튜플들을 기본 테이블로 부터 가져 옴
- \* 뷰에 대한 질의를 기본 테이블에 대한 질의로 변환

```
SELECT title  
FROM ParamountMovie  
WHERE year = 1979;
```

뷰

```
SELECT title  
FROM Movie  
WHERE studioName = 'Paramount' AND year = 1979;
```

기본 테이블

질의 변경  
(query modification)

## □ SQL 언어

### ● "WITH CHECK OPTION" 절

- ☑ 이 옵션절은 뷰에 대한 갱신이나 삽입 연산 실행 시 뷰를 정의할 때 사용한 조건을 위배하는 경우, 실행하지 않도록 하기 위해 사용.

### ● 뷰 변경

- \* 뷰에 대한 정의는 기본 테이블에서처럼 ALTER 문을 사용하여 변경할 수 없음

### ● 뷰 삭제

- \* DROP 문을 사용하여 뷰 제거

```
DROP VIEW 뷰_이름 { RESTRICT | CASCADE } ;
```

- \* RESTRICT 옵션

- ☑ 다른 곳에서 참조하고 있지 않는 한 데이터베이스에서 삭제 가능

- \* CASCADE 옵션

- ☑ 해당 뷰뿐만 아니라 이 뷰가 사용된 다른 모든 뷰나 제약 조건(CONSTRAINT)이 함께 삭제.

## □ SQL 언어

● 다음의 뷰를 생성하여라.

1. 1,000,000이상의 netWorth를 가지는 제작자가 제작한 영화의 제목, 상영년도, 제작자 이름으로 구성된 MovieProdName
2. 주소에 'uk'가 있는 배우가 출연한 영화의 제목, 상영년도, 배우 이름, 배우 주소로 구성된 UK\_MovieStar
3. 각 영화사에 대해서 제작 영화의 수, 최근 제작한 영화의 개봉년도, 평균상영시간으로 구성된 Movie\_Studio. 단, 각 영화사가 제작한 영화의 평균상영시간이 100분 이상이고 처음 개봉한 영화의 년도가 1980년 이후이어야 한다.

## □ SQL 언어

### ● Materialized View

- \* 일반 view와 달리 view의 정의에 사용된 SQL 문장의 질의 결과를 명시적으로 DB에 저장한다.
- \* materialized view가 생성된 후 base table가 변경되어도 옵션에 따라서 materialized view에 영향을 주지 않기도 한다.
  - ☑ refresh 옵션을 이용해 명시적으로 base table의 변경을 반영하게 할 수 있음
- \* read-only, updatable, writeable 등으로 분류하여 materialized view의 갱신이 가능함

### ● 사용 예

```
CREATE MATERIALIZED VIEW all_customers
  REFRESH START WITH ROUND(SYSDATE + 1) + 11/24
  NEXT NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24
  AS
      SELECT * FROM sh.customers@remote
      UNION
      SELECT * FROM sh.customers@local;
```

## □ SQL 언어

### 뷰의 갱신 가능성

[예] 뷰 **ParamountMovie**에 다음과 같은 튜플을 삽입한다고 하자.

```
INSERT INTO ParamountMovie
VALUES ('Star Trek', 1979)
```

**studioName** 애트리뷰트가 뷰의 애트리뷰트에 포함되어 있지 않으므로, 뷰에 삽입되는 튜플이 **Movie**에 반영될 때 **studioName** 값에 **NULL**이 들어간다. 이 튜플은 **ParamountMovie**의 조건을 만족하지 않는다. 즉, **ParamountMovie** 뷰에 나타나지 않는다. 따라서 뷰 **ParamountMovie**는 다음과 같이 수정되어야 한다:

```
CREATE VIEW ParamountMovie AS
SELECT studioName, title, year
FROM    Movie
WHERE   studioName = 'Paramount';
```



## □ SQL 언어

**Movie(title, year, length, inColor, studioName, producerC#)**

**MovieExec(name, address, cert#, netWorth)**

```
CREATE VIEW MovieProd AS  
  
    SELECT title, name  
  
    FROM Movie, MovieExec  
  
    WHERE producerC# = cert#;
```

- 다음과 같은 튜플을 뷰 MovieProd에 삽입하려 한다고 하자:

```
('Greatest Show on Earth', 'cecil B. DeMille')
```

- ✱ **Movie와 MovieExec의 키는 NULL이면 안 된다.**
- ✱ **조인이 이루어지는 애트리뷰트들은 NULL이면 안 된다.**
  - ☑ 두 NULL 값은 동일하지 않음에 주목하라.

# Oracle의 Object Type 생성

```
CREATE OR REPLACE TYPE person_typ AS OBJECT (  
    idno      NUMBER,  
    name      VARCHAR2(30),  
    phone     VARCHAR2(20))  
    NOT FINAL;           // subtype을 가질 수 있게 정의 – default는 FINAL  
  
/  
  
CREATE TYPE student_typ UNDER person_typ (  
    dept_id NUMBER,  
    major VARCHAR2(30))  
    NOT FINAL;  
  
/  
  
CREATE TYPE part_time_student_typ UNDER student_typ (  
    number_hours NUMBER);
```

## Oracle의 Object Table 생성

```
CREATE TABLE contacts (  
    contact      person_typ,  
    contact_date DATE  
);
```

## Oracle의 Object 삽입

```
INSERT INTO contacts
```

```
VALUES (person_typ (12, 'Bob Jones', '650-555-0130'), '2003-02-01' );
```

```
/
```

```
INSERT INTO contacts
```

```
VALUES (student_typ(51, 'Joe Lane', '1-650-555-0178', 12, 'HISTORY'),  
        '2003-06-23' );
```

```
/
```

```
INSERT INTO contacts
```

```
VALUES (part_time_student_typ(52, 'Kim Patel', '1-650-555-0190', 14,  
        'PHYSICS', 20), '2003-06-29' );
```

## Object Table에 대한 검색

```
SELECT contact_date,  
       TREAT(contact as person_typ).name,  
       TREAT(contact as student_typ).major  
FROM contacts  
WHERE  
       TREAT(contact as part_time_student_typ).number_hours > 2;
```

## Structured Type 생성 예 (1)

```
CREATE OR REPLACE TYPE Member AS OBJECT (  
    Name NVARCHAR2 (10) ,  
    Address NVARCHAR2 (30) ,  
    Birthdate DATE ) NOT FINAL ;
```

```
CREATE OR REPLACE TYPE RoomType AS OBJECT (  
    Building CHAR (6 CHAR) ,  
    Floor NUMBER (2) ,  
    RoomNo NUMBER (3) ) NOT FINAL ;
```

## Structured Type 생성 예 (2)

```
CREATE OR REPLACE TYPE Professor  
  UNDER Member (  
    Major VARCHAR2 (10 CHAR) ,  
    Office RoomType  
  ) NOT FINAL ;
```

```
CREATE OR REPLACE TYPE Student  
  UNDER Member (  
    Grade NUMBER (1) ,  
    SubMajor NVARCHAR2 (10)  
  ) NOT FINAL ;
```

## Structured Type 생성 예 (3)

```
CREATE TABLE Departments (
    Name NVARCHAR2 (10) NOT NULL ,
    Office RoomType );
CREATE TABLE Professors (
    ID NUMBER (5) NOT NULL ,
    Info Member ,
    DeptName NVARCHAR2 (10) );
CREATE TABLE Students (
    StudentNo NUMBER (8) NOT NULL ,
    Info Member ,
    DeptName NVARCHAR2 (10) NOT NULL );
```



## □ SQL 언어

# 튜플 삽입

-- Insert Departments

```
insert into departments values ('CSE', RoomType('Eng', 6, 608));
```

```
insert into departments values ('EE', RoomType('Eng', 3, 301));
```

```
insert into departments values ('CITY', RoomType('Eng', 5, 503));
```

-- Insert Students

```
insert into students values(2001, Student('홍길동', '부산시 남구 대연3동', '1990-01-02', 3, 'EE'), 'CSE');
```

```
insert into students values(2002, Student('강남길', '부산시 수영구', '1991-06-12', 2, 'CITY'), 'CSE');
```

```
insert into students values(2003, Student('김용건', '부산시 사하구', '1991-11-22', 4, 'CITY'), 'EE');
```

-- Insert Professors

```
insert into professors values(1001, Professor('양성재', '부산시 남구', '1959-12-20', '전산학',  
RoomType('Eng', 6, 609)), 'CSE');
```

```
insert into professors values(1002, Professor('황전산', '부산시 해운대구 우동', '1968-02-10', '전산학',  
RoomType('Eng', 6, 619)), 'CSE');
```

## SELECT문을 이용한 검색

```
select studentno, treat(info as student).name  
from students;
```

```
select ID, DeptName, treat(info as professor).name  
from professors;
```

```
select treat(info as student).name, studentNo,  
       treat(office as roomtype).building  
from students, departments d  
where deptname = 'CSE' and d.name = deptname;
```

```
select d.name, treat(d.office as roomtype).Floor  
from departments d;
```

## NESTED TABLE을 이용한 애트리뷰트 생성

```
CREATE TYPE phone_ty AS OBJECT (  
    name VARCHAR2(20),  
    seq  INTEGER,  
    no   CHAR(11) );  
  
/  
CREATE OR REPLACE TYPE phone_tab AS TABLE OF phone_ty;  
  
/  
CREATE TABLE person (  
    name VARCHAR2(20) PRIMARY KEY,  
    birthdate DATE,  
    phone_list phone_tab)  
    NESTED TABLE phone_list STORE AS p_table;
```

## □ SQL 언어

# 튜플 삽입 예

```
insert into person values ('박동길', '1966-06-16', phone_tab(phone_ty('mobile', 1, '01065645145')));
insert into table (select phone_list from person where name = '박동길')
    values (phone_ty('office', 2, '0516635145'));
insert into table (select phone_list from person where name = '박동길')
    values (phone_ty('home', 3, '0546635140'));
insert into person values ('강남길', '1953-10-06', phone_tab() );
insert into table (select phone_list from person where name = '강남길')
    values (phone_ty('office', 1, '0516635145'));
insert into table (select phone_list from person where name = '강남길')
    values (phone_ty('home', 2, '0546635140'));
insert into person values ('홍길동', '1966-06-16',
    phone_tab(    phone_ty('mobile', 1, '01065645145'),
                  phone_ty('mobile', 2, '11065645145'),
                  phone_ty('mobile', 3, '21065645145'),
                  phone_ty('office', 4, '31065645145')
    ) );
```

## SELECT문을 이용한 검색

```
select p.name, count(l.no)
from person p, table(p.phone_list) l
group by p.name;
```

```
select p.name, p.birthdate, l.no
from person p, table(select phone_list from person where name = '강남길') l
where p.name = '강남길';
```

```
select p.name, p.birthdate, l.no
from person p, table(p.phone_list) l
where p.name = '강남길';
```

### ■ 함수란? (내장함수)

- 오라클에서 제공하는 다양한 기능을 수행하는 객체
- 함수 사용 장소
  - SELECT 리스트
  - WHERE 절
  - START WITH 절
  - HAVING 절
  - INSERT 문의 INTO 절
  - UPDATE 문의 SET 절

## ■ 함수의 종류

- 숫자형 함수
- 문자형 함수
- 날짜형 함수
- NULL 관련 함수
- 변환 함수
- 집단화 함수
- 객체참조 함수
- MODEL 함수
- DECODE 와 CASE

## ■ 숫자형 함수 (1/2)

- ABS(n) : 절대값 리턴
- SIGN(n) : 양수(1) or 음수(-1) 인지 반환, 0은 0 반환.
- ROUND(n, i)
  - \* n을 소수점 이하 i+1 번째에서 반올림
  - \* i가 0 또는 음수인 경우 소수점 좌측으로 i번째에서 반올림
  - \* ROUND(15.193, 1) = 15.2, ROUND(15.193, -1) = 20
- TRUNC(m, n) : m을 소수점 이하 n번째에서 잘라냄
  - \* TRUNC(15.79, 1) = 15.7, TRUNC(15.79, -1) = 10
- CEIL(n) : n과 같거나 큰 가장 작은 정수 반환
  - \* CEIL(15.7) = 16
- FLOOR(n) : n보다 작거나 가장 큰 정수 반환
  - \* FLOOR(15.7) = 15



## ■ 숫자형 함수 (2/2)

- MOD(m, n) : m/n의 나머지 값 (C언어의 % 연산자)
- REMAINDER(n2, n1) : MOD와 같으나 숫자 타입뿐 아니라 숫자가 아닌 타입도 매개변수로 올 수 있음
- POWER(m, n) : m의 n 제곱 계산
- SQRT(n) : Square root를 계산
- 기타 삼각함수 및 수학적함수 : SIN(), TAN(), COS()

## □ SQL 언어

### ■ 문자형 함수 (1/3)

- 문자형 데이터 타입(CHAR, VARCHAR2, NCAHR ..) 대상 연산 함수
- CONCAT(char1, char2), || : 문자열 연결
- INITCAP(char) : 문자열 char를 각 단어의 첫 문자를 대문자로 나머지 문자들은 소문자로 변환 함 : Capitalize
  - \* INITCAP('the soap') = The Soap
- LOWER(char), UPPER(char) : 각각 소문자 또는 대문자로 구성된 문자열로 변환 함
- LPAD(expr1, n [,expr2]) : expr1의 왼쪽에 expr2를 총길이 n이 되도록 채움
  - \* LPAD('ABCD E', 15, '+-') = '+-+--+--+ABCD E'
  - \* LPAD('ABCD E', 15) = '        ABCD E'
- RPAD(expr1, n [,expr2]) : expr1의 오른쪽에 exprt2를 총길이 n이 되도록 채움
- LTRIM(char [,set]) : char에서 set으로 지정된 문자를 왼쪽에서 제거
  - \* LTRIM('xyXxyWord', 'xy') = 'XxyWord', LTRIM('        AbcWird') = 'AbcWird'
- RTRIM(char [,set]) : char에서 set으로 지정된 문자를 오른쪽에서 제거

## □ SQL 언어

### ■ 문자형 함수 (2/3)

- SUBSTR(char, pos, len) : pos 위치에서 len 길이 만큼의 문자열 일부를 반환
  - \* SUBSTR('ABCDEFGH', 3, 4) = 'CDEF', SUBSTR('ABCDEFGH', -5, 4) = 'CDEF'
- SUBSTRB(char, pos, len) : pos와 len을 바이트 크기로 가정하여 SUBSTR의 기능을 수행
- REPLACE(char, src\_str, rep\_str) : 문자열 char에서 문자열 src\_str을 모두 문자열 rep\_str로 변환
  - \* REPLACE('JACK and JUE', 'J', 'BL') = 'BLACK and BLUE'
- TRANSLATE(expr, frm\_str, to\_str) : 문자열 frm\_str의 각 문자들을 문자열 to\_str로 모두 변환
  - \* TRANSLATE('SQL\*Plus User's/Guide', ' \*/', '\_\_\_') = 'SQL\_Plus\_Users\_Guide'

## □ SQL 언어

### ■ 문자형 함수 (3/3)

- TRIM([LEADING, TRAILING, BOTH] [trim\_chr FROM] trim\_src) : 문자열 trim\_src에서 문자열 trim\_chr를 앞(LEADING), 뒤(TRAILING) 혹은 둘 다(BOTH)에서 제거

✱ TRIM(LEADING '0' FROM '0000ABCDEF') = 'ABCDEF'

✱ TRIM(BOTH ' ' ABCDE ' ') = 'ABCDE'

- ASCII(char) : 문자 char의 ASCII 코드 값 반환
- INSTR(string, src\_str [, pos [, occur]]) : 문자열 string에서 문자열 src\_str을 검색하여 위치를 반환 함. pos는 string 내에서 찾기 시작할 위치, occur는 src\_str이 여러 개 있는 경우 검색될 순번.

✱ INSTR('CORPORATE FLOOR', 'OR', 3, 2) = 14

- LENGTH(char) : 문자열 길이 반환

### ■ 날짜형 함수 (1/2)

- SYSDATE : 시스템 현재 날짜 및 시간
- CURRENT\_DATE : TIME\_ZONE 기준 현재 날짜
- SYSTIMESTAMP : 시스템의 날짜를 실수로 반환
- ADD\_MONTHS(date, integer) : 날짜형 데이터인 date에 integer 값 만큼의 달(month)을 추가
- MONTHS\_BETWEEN(date1, date2) : 날짜형 데이터인 date1과 date2 사이의 달(month) 수를 반환
  - \* MONTHS\_BETWEEN(TO\_DATE('02-02-1995', 'MM-DD-YYYY'), TO\_DATE('01-01-1995', 'MM-DD-YYYY')) = 1.03225806
- NEXT\_DAY(date, char) : 날짜형 데이터인 date 이후에 요일값은 char 첫번째 날짜를 반환
  - \* NEXT\_DAY('02-FEB-2001', 'TUESDAY') = '06-FEB-2001'
- LAST\_DAY(date) : 날짜형 데이터인 date가 나타내는 달(month)의 마지막 날
  - \* LAST\_DAY('02-MAY-2001') = '31-MAY-2001'

## ■ 날짜형 함수 (2/2)

- ROUND(date, fmt) : 날짜형 데이터인 date에 대해서 fmt 형식에 따라서 올림 함
  - \* ROUND('27-OCT-2000', 'YEAR') = '01-JAN-2001'
- TRUNC(date, fmt) : 날짜형 데이터인 date에 대해서 fmt 형식에 따라서 내림 함
- EXTRACT([YEAR] [MONTH] [DAY] [HOUR] [MINUTE] [SECOND] FROM datetime) : 날짜형 데이터인 datetime로부터 특정 날짜 단위값을 추출
  - \* EXTRACT(YEAR FROM DATE '2011-03-16') : 2011
- DBTIMEZONE : DB의 timezone 값 반환 (우리나라는 +9)
- SESSIONTIMEZONE : 현재 session의 timezone 값 반환

## □ SQL 언어

### ■ NULL 관련 함수

- NVL(expr1, expr2) : expr1이 NULL이면 expr2를, expr1이 NULL이 아니면 expr1을 반환
  - \* NVL(TO\_CHAR(length), '해당 없음'), SELECT id, NVL(phoneno, name) ...
- NVL2(expr1, expr2, expr3) : expr1이 NULL이면 expr2를, expr1이 NULL이 아니면 expr3를 반환
- NULLIF(expr1, expr2) : expr1과 expr2가 같으면 NULL을 반환. 다르면 expr1을 반환
- COALESCE(expr1, expr2, ...) : expr1부터 첫번째 NULL이 아닌  $expr_i$ 를 반환
- LNNVL(condition)
  - \* condition을 체크하여 결과가 FALSE, UNKNOWN이면 TRUE, 결과가 TRUE이면 FALSE를 반환
  - \* 매개변수로 조건식이 들어감. 따라서 SELECT 절에서 사용 불가

## □ SQL 언어

### ● NULL에 대한 연산

①  $x$  나  $+$ 와 같은 산술 연산자에 NULL을 사용 : 결과는 NULL

☑  $x$ 의 값이 NULL이면,  $x + 3$  은 NULL이다.

②  $=$  이나  $>$ 와 같은 비교 연산자에 NULL을 사용 : 결과는 UNKNOWN

☑  $x$ 의 값이 NULL이면,  $x > 3$  은 UNKNOWN이다.

\*  $x$ 의 값이 NULL인 경우  $x = \text{NULL}$  ?

☑ SQL 문법에 어긋남.

☑  $x \text{ is NULL}$ ,  $x \text{ is NOT NULL}$ 을 사용

\*  $x$ 의 값이 NULL인 경우  $x * 0$  ?,  $x - x$  ?

☑ 모두 결과는 NULL



## □ SQL 언어

### ● 진리값 UNKNOWN

- \* TRUE(1), UNKNOWN(1/2), FALSE(0)으로 이해
- \*  $x \text{ AND } y : \min(x,y)$ ,  $x \text{ OR } y : \max(x,y)$ , NOT  $x : 1-x$

x	y	x AND y	x OR y	NOT x
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

[예] SELECT \* FROM Movie

WHERE length <= 120 OR length > 120; /\* length IS NOT NULL \*/

▶ length가 NULL이 아닌 튜플을 찾아라 !!!

## □ SQL 언어

### ■ 변환함수

- TO\_CHAR(char)
- TO\_CHAR(datetime)
- TO\_CHAR(number) : TO\_CHAR(length, '999')
- TO\_NUMBER(expr, fmt) : TO\_NUMBER(ss\_no)+10
- TO\_BINARY\_DOUBLE, TO\_BINARY\_FLOAT
- TO\_DATE(char, fmt)
- TO\_TIMESTAMP(char, fmt) : TO\_TIMESTAMP('10-SEP-0214:10:10.123000', 'DD-MON-RRHH24:MI:SS.FF')
- TO\_TIMESTAMP\_TZ(char, fmt)