SQL 기본

sqlite 생성

- sqlite3 데이터베이스이름.sqlite3

DDL (데이터 정의 언어)

CREATE

```
CREATE TABLE 테이블명 (칼럼명 옵션);

-- 부서
CREATE TABLE DEPT (
    deptno varchar2(4) PRIMARY KEY,
    deptname varchar2(20)
);

-- 직원
CREATE TABLE EMP (
    empno number(10),
    ename varchar2(20),
    sal number(10,2) default 0,
    deptno varchar2(4) NOT NULL,
    createdate date default SYSDATE,
    CONSTRAINT e_pk PRIMARY KEY (empno),
    CONSTRAINT d_fk FOREIGN KEY (deptno)
        REFERENCES dept(deptno)
        ON DELETE CASCADE
);
```

ALTER

```
-- 테이블명 변경
ALTER TABLE 테이블명
RENAME TO 새 테이블 명;

-- 칼럼 추가
ALTER TABLE 테이블명
ADD (칼럼);

-- 칼럼 변경
ALTER TABLE 테이블명
MODIFY (칼럼);

-- 칼럼 삭제
ALTER TABLE 테이블명
DROP COLUMN 칼럼명;
```

```
-- 컬럼명 변경
ALTER TABLE 테이블명
RENAME COLUMN 기존칼럼명 TO 새칼럼명;
```

DROP

```
DROP TABLE 테이블명 CASCADE CONSTRAINT; -- 해당 테이블을 참조한 슬레이브 테이블과 관련된 제약사항도 삭제
```

VIEW

> 가상의 테이블로 실제 데이터를 가지고 있지 않고 참조해서 원하는 칼럼만 조회
>
> 변경 불가능

```
-- 뷰 생성
CREATE VIEW 뷰이름 AS
SELECT * FROM 테이블명;

-- 뷰 제거
DROP VIEW 뷰이름;
```

DML (데이터 조작 언어)

```
SELECT * FROM 테이블명 WHERE 조건;

INSERT INTO 테이블명 VALUES 삽입할 레코드

UPDATE 테이블명 SET 수정할 데이터 WHERE 수정할 레코드;

DELETE FROM 테이블명 WHERE 삭제할 레코드;
```

INSERT

```
INSERT INTO 테이블명[(칼럼, 칼럼)] VALUES(칼럼내용, 칼럼내용);

-- NOLOGGING을 사용해서 로그파일 기록을 최소화시켜 성능을 향상시킨다.
ALTER TABLE 테이블명 NOLOGGING;
```

UPDATE

```
UPDATE 테이블명 SET 칼럼명=바꾼후레코드 [WHERE 칼럼명=기존레코드;]; -- WHERE 입력 안하면 모든 데이터가 수정된다.
```

DELETE

```
DELETE FROM 테이블명 [WHERE 칼럼명=레코드]; -- WHERE 입력 안하면 테이블 내의 모든 레코드 삭제

-- DELETE는 테이블 용량이 초기화되지 않는다.
-- TRUNCATE는 테이블 용량이 초기화된다.
TRUNCATE FROM 테이블명;
```

SELECT

> 실행순서 FROM - WHERE - GROUP BY - HAVING - SELECT - ORDER BY

```
-- 기본
SELECT 칼럼 FROM 테이블명 WHERE 칼럼명=레코드

-- 출력시 '님'이 붙어서 나옴
SELECT 칼럼||'님' FROM 테이블명;

-- ORDER BY
SELECT 칼럼 FROM 테이블명
[WHERE 조건]
[ORDER BY 칼럼 [ASC/DESC], 칼럼 [ASC/DESC]]; -- [오름차순/내림차순]
-- ORDER BY는 데이터를 너무 많이 사용한다.
-- 힌트를 사용해서 인덱스 기준 내림차순 가능
-- EX)
SELECT /*+ INDEX_DESC(deptno)*/ *
FROM DEPT deptno;

-- DISTINCT
SELECT [DISTINCT(중복없이)]칼럼 FROM 테이블명
[WHERE 조건];

-- ALIAS
SELECT 칼럼 AS "별명" FROM 테이블명 바꿀테이블명
[WHERE 바꾼테이블명.칼럼명=레코드];

-- WHERE
SELECT 칼럼 FROM 테이블명 WHERE 칼럼 LIKE '와일드카드';
SELECT 칼럼 FROM 테이블명 WHERE 칼럼 LIKE '%2_' -- %는 여러개가 가능, _는 정확히 저 갯수만큼만 가능;
SELECT 칼럼 FROM 테이블명 WHERE 칼럼 BETWEEN A AND B;
SELECT 칼럼 FROM 테이블명 WHERE 칼럼 IN 리스트;
SELECT 칼럼 FROM 테이블명 WHERE (칼럼1, 칼럼2) IN ((칼럼1요소1, 칼럼2요소1), (칼럼1요소2, 칼럼2요소2));
```

```
SELECT 칼럼 FROM 테이블명 WHERE 칼럼 IS NULL; -- NULL값 조회

-- NULL 함수
NVL(칼럼명, '변경될 값') -- NULL이면 다른 값으로 변경
NVL2(칼럼명, 'NULL이면 변경될 값', 'NULL이 아니면 변경될 값')
NULLIF(칼럼명1, 칼럼명2) -- 두개 값이 같으면 NULL 아니면 칼럼명1을 반환
COALESCE(칼럼명1, 칼럼명2, 칼럼명3 ...) -- NULL이 아닌 최소의 인자값을 반환

-- GROUP BY
SELECT 그룹화할칼럼1, 집계함수(칼럼2) FROM 테이블명
GROUP BY 칼럼1
HAVING 집계함수와 조건; -- WHERE도 사용 가능하나 WHERE는 개별 행의 조건을 따져서 조건에 맞지 않는 행은 아예 넣지를 않는다.
-- EX)
SELECT DEPTNO, SUM(SAL) FROM EMP
GROUP BY DEPTNO
HAVING SUM(SAL) > 10000;
-- 집계함수 종류
COUNT()
SUM()
AVG()
MAX()/MIN()
STDDEC() -- 표준편차
VARIAN() -- 분산

-- COUNT
SELECT COUNT(*) FROM 테이블명;    -- NULL 값을 포함한 행의 수
SELECT COUNT(칼럼) FROM 테이블명;  -- NULL 값을 제외한 행의 수

SELECT AVG(칼럼) FROM 테이블명;
SELECT SUM(칼럼) FROM 테이블명;
SELECT MAX(칼럼) FROM 테이블명;
SELECT MIN(칼럼) FROM 테이블명;

-- LIMIT
SELECT 칼럼 FROM 테이블명 LIMIT 숫자 OFFSET 숫자;
```

형변환

명시적 형변환

```
TO_NUMBER(문자열)

TO_CHAR(숫자 혹은 날짜, [FORMAT]) -- FORMAT의 문자로 변환

TO_DATE(문자열, [FORMAT])
```

암시적 형변환

```
-- EX)
SELECT *
FROM EMP
WHERE EMPNO='100';

-- EMPNO는 숫자형으로 지정해 놨기 때문에 TO_CHAR로 암시적 형변환을 해서 비교함
-- 하지만 암시적 형변환은 인덱스를 사용할 수는 없음

SELECT *
FROM EMP
WHERE EMPNO=TO_NUMBER('100')

-- 명시적 형변환으로 인덱스를 사용할 수 있음
```

내장형 함수

> 형변환 함수, 문자열 및 숫자형 함수, 날짜형 함수

DUAL 테이블

> 자동 생성 테이블
>
> 임시 테이블
>
> 모든 사용자가 사용 가능

문자열 함수

```
ASCII(문자)

CHAR(ASCII 코드값)

SUBSTR(문자열, M, N) -- M번째부터 N개

CONCAT(문자열1, 문자열2) -- 문자열 합치기

UPPER/LOWER(문자열)

LENGTH 혹은 LEN(문자열)

RTRIM/LTRIM(문자열, 지정 문자)

TRIM(문자열, 지정 문자) -- 양쪽에서 지정 문자 삭제
```

날짜형 함수

```
SYSDATE -- 오늘 날짜

EXTRACT(YEAR FROM SYSDATE)  -- 연도 가져오기

TO_CHAR(SYSDATE, 'YYYY/MM/DD')  -- 날짜 변환 포맷
```

숫자형 함수

```
ABS()

SIGN()

MOD(숫자1, 숫자2)  --나눗셈

CEIL/CEILING()

FLOOR()

ROUND(숫자, M)  -- 소수점 M자리에서 반올림

TRUNC(숫자, M)  -- 소수점 M자리에서 절삭

COALESCE(NULL, '2', '1') -- NULL이 아닌 인자값 반환 => '2' 반환
```

DECODE와 CASE

DECODE

> IF문을 구현할 수 있다.

```
SELECT DECODE(EMPNO, 1000, 'TRUE', 'FALSE') FROM EMP;
```

CASE

> IF-THEN-ELSE -END구현할 수 있다.

```
SELECT CASE
        WHEN EMPNO = 1000 THEN 'A'
        WHEN EMPNO = 1001 THEN 'B'
        ELSE 'C'
    END
FROM EMP;
```

ROWNUM과 ROWID과 WITH

ROWNUM

> MYSQL의 LIMIT와 같은것
>
> 조회되는 행 수를 제한 할 때

```
SELECT * FROM DEPT WHERE ROWNUM <= 15;

-- 페이지 조회 하려면 FROM 에 SELECT 넣는 서브쿼리를 사용해야 가능하다.

SELECT * FROM ( SELECT ROWNUM LIST, DEPTNAME FROM DEPT) WHERE LIST BETWEEN 2 AND
10;
```

ROWID

> 고유한 값

```
SELECT ROWID, DEPTNAME FROM DEPT;
```

WITH

> 변수 할당처럼 사용 가능

```
WITH W_EMP AS (SELECT * FROM EMP WHERE DEPTNO=30)
SELECT * FROM W_EMP;
```

DCL(Data Control Language)

GRANT

> 사용자에게 권한 부여

```
-- 구조
GRANT privileges ON object TO user;

-- privileges(권한) 종류
SELECT/INSERT/UPDATE/DELETE
REFERENCES -- 참조 제약조건 생성 권한
ALTER
INDEX -- 인덱스 생성 권한
ALL

-- WITH GRANT OPTION
WITH GRANT OPTION
-- 특정 사용자에게 권한을 부여할 수 있는 권한
-- A->B->C 일때 A가 B 권한 취소하면 C도 취소됨
```

```
WITH ADMIN OPTION
-- 모든 권한 부여
-- A->B->C 일때 A가 B 권한 취소하면 C는 취소 안됨
```

REVOKE

> 권한 취소

```
REVOKE privileges ON object FROM user;
```

TCL(Transaction Control Language)

COMMIT

> INSERT, UPDATE, DELETE 문으로 변경한 데이터를 데이터 베이스에 반영
>
> 완료시 LOCK이 해제된다.
>
> COMMIT 실행되면 하나의 트랜잭션 과정을 종료한다.

```
COMMIT;
```

*** AUTO COMMIT**

> DDL, DCL 사용 시 자동 COMMIT
>
> SET AUTOCOMMIT ON;을 실행하면 자동 커밋

ROLLBACK

> 데이터 변경 사항 모두 취소하고 트랜잭션 종료
>
> LOCK 해제되고 다른 사용자들도 데이터베이스 행조작 가능

```
ROLLBACK;
```

SAVEPOINT

> 트랜잭션을 작게 분할하는 것

```
SAVEPOINT [SAVEPOINT명]

-- ROLLBACK 시
ROLLBACK TO [SAVEPOINT명]
-- 그냥 ROLLBACK만 입력하면 SAVEPOINT 유무와 관계없이 모든 변경사항을 저장하지 않음.
```

SQL 활용

JOIN

EQUI(등가) 조인 (교집합)

> 해시 조인으로 조인한다.

EQUI 조인

> WHERE구에 "="을 사용하여 정의

```
--기본
SELECT * FROM 테이블1, 테이블2
WHERE 테이블1.칼럼1 = 테이블2.칼럼2

SELECT * FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
AND EMP.ENAME LIKE '임%'
ORDER BY ENAME;
```

INNER JOIN

> INNER JOIN으로 테이블 정의하고 ON구에 "="을 사용하여 정의

```
--기본
SELECT * FROM 테이블1 INNER JOIN 테이블2
ON 테이블1.칼럼1 = 테이블2.칼럼2

SELECT * FROM EMP INNER JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO
AND EMP.ENAME LIKE '임%'
ORDER BY ENAME;
```

INTERSECT

> 정확하게 교집합 되는 칼럼만 조회 시 사용

```
SELECT DEPTNO FROM EMP
INTERSECT
SELECT DEPTNO FROM DEPT;

-- DEPTNO만 출력
```

NON-EQUI(비등가) 조인

> "="외에 다른 것들을 사용 ("<", ">=" 등)

OUTER JOIN

> 교집합이 아닌 이외 칼럼을 추가로 합칠 때 사용

```
-- FULL OUTER JOIN
-- (+)로 표현 가능

SELECT * FROM DEPT, EMP
WHERE EMP.DEPTNO (+)= DEPT.DEPTNO;

-- LEFT/RIGHT OUTER JOIN
SELECT * FROM DEPT LEFT OUTER JOIN EMP
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

CROSS JOIN

> 조인구 없이 조인한다.
>
> 카테시안 곱이 발생 (그냥 두 개 행 갯수 곱한 만큼의 연산수라는 뜻)

```
SELECT * FROM EMP CROSS JOIN DEPT;

SELECT * FROM EMP, DEPT;
```

UNION

> 두 개의 테이블 하나로 합치는 것
>
> 두 개의 테이블에 칼럼 수 혹은 데이터 형식이 다르면 오류가 발생

UNION

> 중복을 제거하고 합친다.
>
> 정렬도 한다.

```
SELECT * FROM EMP
UNION
SELECT * FROM EMP;
```

UNION ALL

> 중복을 제거하거나 정렬하지 않는다.

```
SELECT * FROM EMP
UNION ALL
SELECT * FROM EMP;
```

MINUS (차집합)

```
SELECT DEPTNO FROM DEPT
MINUS
SELECT DEPTNO FROM EMP;

-- DEPT 테이블에 있는 것만 조회된다.
```

계층형 조회 (CONNECT BY)

> 트리 형태로 조회
>
> 역방향도 가능
>
> `START WITH`, `CONNECT BY PRIOR` 사용

```
CREATE TABLE EMP_TREE (
    EMPNO NUMBER(10) PRIMARY KEY,
    ENAME VARCHAR2(20),
    DEPTNO NUMBER(10),
    MGR NUMBER(10)
);

INSERT INTO EMP_TREE VALUES(1000, 'TEST1', 20, NULL);
INSERT INTO EMP_TREE VALUES(1001, 'TEST2', 30, 1000);
INSERT INTO EMP_TREE VALUES(1002, 'TEST3', 30, 1000);
INSERT INTO EMP_TREE VALUES(1003, 'TEST4', 20, 1000);
INSERT INTO EMP_TREE VALUES(1004, 'TEST5', 30, 1000);
INSERT INTO EMP_TREE VALUES(1005, 'TEST6', 30, 1001);
INSERT INTO EMP_TREE VALUES(1006, 'TEST7', 10, 1001);
INSERT INTO EMP_TREE VALUES(1007, 'TEST8', 20, 1002);
INSERT INTO EMP_TREE VALUES(1008, 'TEST9', 10, 1002);

-- 최대 깊이 조회
SELECT MAX(LEVEL)
FROM EMP_TREE
START WITH MGR IS NULL -- MGR이 NULL인 부분이 시작점
CONNECT BY PRIOR EMPNO = MGR; -- 조인 조건

-- LPAD : LEVEL마다 공백을 4배수씩 넣어준다.
SELECT LEVEL, LPAD(' ', 4*(LEVEL-1) ) || EMPNO, MGR, CONNECT_BY_ISLEAF
FROM EMP_TREE
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;
```

```
-- CONNECT BY 키워드
LEVEL -- 항목 깊이

CONNECT_BY_ROOT -- 최상위 값

CONNECT_BY_ISLEAF -- 최하위인지 아닌지

SYS_CONNECT_BY_PATH -- 전체 경로 표시

NOCYCLE -- 순환구조 발생지점까지만 전개

CONNECT_BY_ISCYCLE -- 순환구조 발생 지점 표시
```

서브 쿼리

> SELECT 문 내에 다시 SELECT 문 사용하는 SQL문

- 인라인 뷰 : FROM구에 SELECT
- 스칼라 서브쿼리 : SELECT문에 SELECT
- 서브쿼리 : WHERE구에 SELECT

단일 행 서브쿼리, 다중 행 서브쿼리

- 단일 행 서브쿼리
  - 결과가 반드시 한 행만 조회
  - 비교 연산자 =, <, <= 등 사용
- 다중 행 서브쿼리
  - 여러 개의 행 반환
  - 다중 행 비교 연산자
    - IN (SUBQUERY) : MAIN QUERY의 비교조건이 SUBQUERY의 결과 중 하나만 동일하면 참 (OR 조건)
    - ALL (SUBQUERY) : MAIN QUERY와 SUBQUERY가 모두 동일하면 참
    - ANY (SUBQUERY) : MAIN QUERY의 비교조건이 SUBQUERY의 결과 중 하나 이상 동일하면 참
    - EXISTS (SUBQUERY) : MAIN QUERY와 SUBQUERY의 결과가 하나라도 존재하면 참

스칼라 SUBQUERY

> 반드시 한 행과 한 칼럼만 반환하는 서브쿼리
>
> 여러 행이 반환되면 오류가 발생

연관(Correlated) SUBQUERY

> Subquery 내에서 Main Query 내의 칼럼을 사용하는 것

```
SELECT * FROM EMP A
WHERE A.DEPTNO =
( SELECT DEPTNO FROM DEPT B
WHERE B.DEPTNO=A.DEPTNO);
```

그룹 함수

ROLLUP

> 전체 합계를 구하는 함수

```
SELECT DECODE(DEPTNO, NULL, '전체합계', DEPTNO), SUM(EMPNO)
FROM EMP_TREE
GROUP BY ROLLUP(DEPTNO);

SELECT DEPTNO, MGR, SUM(EMPNO)
FROM EMP_TREE
GROUP BY ROLLUP(DEPTNO, MGR);
-- DEPTNO 별로 그룹 짓고 그 안에서 MGR별 그룹 합계 출력하고 해당 DEPTNO 그룹 합계 출력
-- 그리고 맨 아래 모든 총합 출력
```

GROUPING

> 어떤 속성으로 묶였는지 표시하기 위한 함수 (0 또는 1로 표현)

```
SELECT DEPTNO, GROUPING(DEPTNO), MGR, GROUPING(MGR), SUM(EMPNO)
FROM EMP_TREE
GROUP BY ROLLUP(DEPTNO, MGR);

-- GROUPING(MGR)이 1이면 해당 DEPTNO의 MGR 총합 표시하는 곳
-- GROUPING(DEPTNO)가 1이면 DEPTNO의 총합 표시하는 곳
```

GROUPING SET

> GROUP BY에 적힌 순서 상관 없이 해당하는 칼럼의 소계 구하는 함수

```
SELECT DEPTNO, MGR, SUM(EMPNO)
FROM EMP_TREE
GROUP BY GROUPING SETS (DEPTNO, MGR);
```

CUBE

> 조합할 수 있는 모든 경우의 수 조합된 결과

```
SELECT DEPTNO, MGR, SUM(EMPNO)
FROM EMP_TREE
GROUP BY CUBE(DEPTNO, MGR);
```

윈도우 함수

윈도우 함수

```
-- 기본
SELECT WINDOW_FUNCTION(ARGUMENTS)
    OVER (PARTITION BY 칼럼
         ORDER BY WINDOWING절)
FORM 테이블명;

-- 구조
ARGUMENTS -- WINDOW_FUNCTION(SUM, AVG 등)의 종류에 따라 들어가는 인수

PARTITION BY -- 전체 집합을 기준에 의해 소그룹으로 나눈다.

WINDOWING절 -- 행 기준의 범위 정한다. ROWS는 물리적 결과의 행 수, RANGE는 논리적인 값에 의한 범위

-- WINDOWING
ROWS -- 부분집합인 윈도우 크기를 물리적 단위로 행의 집합을 지정

RANGE -- 논리적인 주소에 의해 행 집합을 지정

BETWEEN~AND

UNBOUNDED PRECEDING -- 윈도우의 시작 위치가 첫번째 행

UNBOUNDED FOLLOWING -- 윈도우의 마지막 위치가 마지막 행

CURRENT ROW -- 시작위치나 마지막 위치가 현재 행


-- EX) 마지막 열에는 처음부터 끝까지 토탈 합만 쭉 나온다.
SELECT EMPNO, SAL,
    SUM(SAL) OVER(ORDER BY SAL
                  ROWS BETWEEN UNBOUNDED PRECEDING
                  AND UNBOUNDED FOLLOWING) TOTSAL
FROM EMP;

-- EX) 현재 행까지 누적 합 구하기
SELECT EMPNO, SAL,
    SUM(SAL) OVER(ORDER BY SAL
                  ROWS BETWEEN UNBOUNDED PRECEDING
                  AND CURRENT ROW) TOTSAL
FROM EMP;
```

순위 함수

> RANK 함수

```
-- 종류
RANK -- 동일한 순위는 동일한 값이 부여(2등 2명이면 3등 없고 4등부터)

DENSE_RANK -- 동일한 순위를 하나의 건수로 계산(2등 2명이어도 둘 다 2등하고 3등부터)

ROW_NUMBER -- 동일한 순위에 고유의 순위 부여(2등 2명이면 한명은 2등 한명은 3등)

-- EX)
SELECT ENAME, SAL,
    RANK() OVER(ORDER BY SAL DESC) ALL_RANK,
    DENSE_RANK() OVER(ORDER BY SAL DESC) DENSE_RANK,
    ROW_NUMBER() OVER(ORDER BY SAL DESC) ROW_NUM,
FROM EMP;
```

집계 함수(AGGREGATE)

```
-- 종류
SUM, AVG, COUNT, MAX/MIN
```

행 순서 관련 함수

```
-- 종류
FIRST_VALUE

LAST_VALUE

LAG -- 이전 행을 가지고 온다.

LEAD -- 특정 위치의 행을 가지고 온다. 기본값 1

-- EX) LAG
SELECT DEPTNO, ENAME, SAL,
LAG(SAL) OVER(ORDER BY SAL DESC) AS PRE_SAL
FROM EMP;

-- EX) LEAD 2번째 뒤에 있는 값 가져온다.
SELECT DEPTNO, ENAME, SAL,
LEAD(SAL, 2) OVER(ORDER BY SAL DESC) AS PRE_SAL
FROM EMP;
```

비율 관련 함수

```
-- 종류
CUME_DIST -- 파티션별 누적 백분율

PERCENT_RANK -- 파티션별 제일 먼저 나온 것 0 제일 늦게 나온 것 1로 순서별 백분율

NTILE -- 파티션별 인자로 들어온 순자만큼 등급을 나눈다.

RATIO_TO_REPORT -- 파티션별 전체 SUM(칼럼)에 대한 행 별 칼럼 값 백분율

-- EX)
SELECT DEPTNO, EMPNO,
NTILE(3) OVER(PARTITION BY DEPTNO ORDER BY EMPNO DESC) AS NTILE_EMPNO
FROM EMP_TREE;
```

테이블 파티션

RANGE PARTITION

> 값의 범위를 기준으로 나누어 저장하는 것

LIST PARTITION

> 특정 값을 기준으로 분할

HASH PARTITION

> 데이터베이스 관리 시스템이 내부적으로 해시 함수를 사용해서 테이블을 분할

COMPOSITE PARTITION

> 여러 개의 파티션 기법을 조합해서 사용하는 것

파티션 인덱스

- Global Index : 여러 개의 파티션에서 하나의 인덱스 사용 (Oracle은 지원 X)
- Local Index : 파티션 별로 각자의 인덱스 사용
- Prefixed Index : 파티션 키와 인덱스 키가 동일
- Non-Prefixed Index : 파티션 키와 인덱스 키가 다르다. (Oracle은 지원 X)

SQL 최적화의 원리

옵티마이저와 실행계획

옵티마이저

> 실행 계획을 세우는 소프트웨어

***Tip**

정렬 깔끔하게 하는법

```
.headers on
.mode column
SELECT * FROM flights;
```

csv파일 불러오기

```
.mode csv
.import 파일명(ex)users.csv 테이블명(ex)users_user
```