2-1 윈도우 함수의 개념

- 윈도우 함수(window function)
  - 테이블의 행과 행 사이 관계를 쉽게 정의하기 위해 MySQL에서 제공하는 함수
  - OVER 절이 들어간 함수

- 윈도우 함수와 함께 사용되는 집계 함수
  - AVG( ), COUNT( ), MAX( ), MIN( ), STDDEV( ), SUM( ), VARIANCE( ) 등

- 윈도우 함수와 함께 사용되는 비집계 함수
  - CUME_DIST( ), DENSE_RANK( ), FIRST_VALUE( ), LAG( ), LAST_VALUE( ), LEAD( ), NTH_VALUE( ), NTILE( ), PERCENT_RANK( ), RANK( ), ROW_NUMBER( ) 등

2-2 순위 함수

- 순위 함수
  - 결과에 순번 또는 순위(등수)를 매기는 함수
  - 비집계 함수 중에서 RANK( ), NTILE( ), DENSE_RANK( ), ROW_NUMBER( ) 등이 해당

```
<순위함수이름>() OVER(
   [PARTITION BY <partition_by_list>]
   ORDER BY <order_by_list>)
```

[실습 6-3] 순위 함수 사용하기

교재 201~204p 참고

1 cookDB 초기화하기

1-1 C:₩SQL₩cookDB.sql 파일 실행

1-2 열린 쿼리 창을 모두 닫고 새 쿼리 열기

2 키가 큰 순으로 정렬하기

2-1 회원 테이블(userTBL)에서 키가 큰 순으로 순위 매기기

```
USE cookDB;
SELECT ROW_NUMBER() OVER(ORDER BY height DESC)
"키큰순위", userName, addr, height
   FROM userTBL;
```

키큰순위	userName	addr	height
1	박수홍	서울	183
2	강호동	경북	182
3	이휘재	경기	180
4	남희석	충남	180
5	유재석	서울	178
6	김용만	서울	177
7	신동엽	경기	176
8	김제동	경남	173
9	김국진	서울	171
10	이경규	경남	170

[실습 6-3] 순위 함수 사용하기

교재 201~204p 참고

2-2 키가 같은 경우 이름의 가나다순으로 정렬

```
SELECT ROW_NUMBER() OVER(ORDER BY height DESC,
userName ASC) "키큰순위", userName, addr, height
   FROM userTBL;
```

키큰순위	userName	addr	height
1	박수홍	서울	183
2	강호동	경북	182
3	남희석	충남	180
4	이휘재	경기	180
5	유재석	서울	178
6	김용만	서울	177
7	신동엽	경기	176
8	김제동	경남	173
9	김국진	서울	171
10	이경규	경남	170

2-3 각 지역별로 순위 매기기

```
SELECT addr, ROW_NUMBER() OVER(PARTITION BY addr
ORDER BY height DESC, userName ASC) "지역 별키큰순위",
userName, height
   FROM userTBL;
```

addr	지역별키큰순위	userName	height
경기	1	이휘재	180
경기	2	신동엽	176
경남	1	김제동	173
경남	2	이경규	170
경북	1	강호동	182
서울	1	박수홍	183
서울	2	유재석	178
서울	3	김용만	177
서울	4	김국진	171
충남	1	남희석	180

[실습 6-3] 순위 함수 사용하기

교재 201~204p 참고

2-4 키가 같을 경우 동일한 등수로 처리

```
SELECT DENSE_RANK() OVER(ORDER BY height DESC) "키큰순위", userName, addr, height
   FROM userTBL;
```

	키큰순위	userName	addr	height
▶	1	박수홍	서울	183
	2	강호동	경북	182
	3	이휘재	경기	180
	3	남희석	충남	180
	4	유재석	서울	178
	5	김용만	서울	177
	6	신동엽	경기	176
	7	김제동	경남	173
	8	김국진	서울	171
	9	이경규	경남	170

2-5 3등 다음에 4등을 빼고 5등이 나오게 하려면 RANK( ) 함수 사용

```
SELECT RANK() OVER(ORDER BY height DESC) "키큰순위", userName, addr, height
   FROM userTBL;
```

	키큰순위	userName	addr	height
▶	1	박수홍	서울	183
	2	강호동	경북	182
	3	이휘재	경기	180
	3	남희석	충남	180
	5	유재석	서울	178
	6	김용만	서울	177
	7	신동엽	경기	176
	8	김제동	경남	173
	9	김국진	서울	171
	10	이경규	경남	170

[실습 6-3] 순위 함수 사용하기

교재 201~204p 참고

2-6 전체 인원을 키가 큰 순으로 정렬한 후 몇 개의 그룹으로 분할

```
SELECT NTILE(2) OVER(ORDER BY height DESC) "반번호",
userName, addr, height
   FROM userTBL;
```

반번호	userName	addr	height
1	박수홍	서울	183
1	강호동	경북	182
1	이휘재	경기	180
1	남희석	충남	180
1	유재석	서울	178
2	김용만	서울	177
2	신동엽	경기	176
2	김제동	경남	173
2	김국진	서울	171
2	이경규	경남	170

2-7 3개 반으로 분리

```
SELECT NTILE(4) OVER(ORDER BY height DESC) "반번호",
userName, addr, height
   FROM userTBL;
```

반번호	userName	addr	height
1	박수홍	서울	183
1	강호동	경북	182
1	이휘재	경기	180
2	남희석	충남	180
2	유재석	서울	178
2	김용만	서울	177
3	신동엽	경기	176
3	김제동	경남	173
4	김국진	서울	171
4	이경규	경남	170

[실습 6-4] 분석 함수 사용하기

교재 205~207p 참고

1 특정 데이터와의 차이 값 구하기

1-1 회원 테이블(userTBL)에서 키가 큰 순으로 정렬한 후 다음 사람과의 키 차이 구하기

```
USE cookDB;
SELECT userName, addr, height AS "키",
        height - (LEAD(height, 1, 0) OVER (ORDER BY height DESC)) AS "다음 사람과 키 차이"
   FROM userTBL;
```

userName	addr	키	다음 사람과 키 차이
박수홍	서울	183	1
강호동	경북	182	2
이휘재	경기	180	0
남희석	충남	180	2
유재석	서울	178	1
김용만	서울	177	1
신동엽	경기	176	3
김제동	경남	173	2
김국진	서울	171	1
이경규	경남	170	170

[실습 6-4] 분석 함수 사용하기

교재 205~207p 참고

1-2 지역별로 가장 키가 큰 사람과의 차이 구하기

```
SELECT addr, userName, height AS "키",
        height - (FIRST_VALUE(height) OVER (PARTITION BY addr ORDER BY height DESC))
            AS "지역별 최대키와 차이"
   FROM userTBL;
```

addr	userName	키	지역별 최대키와 차이
경기	이휘재	180	0
경기	신동엽	176	-4
경남	김제동	173	0
경남	이경규	170	-3
경북	강호동	182	0
서울	박수홍	183	0
서울	유재석	178	-5
서울	김용만	177	-6
서울	김국진	171	-12
충남	남희석	180	0

[실습 6-4] 분석 함수 사용하기

교재 205~207p 참고

2 누적 백분율 구하기

2-1 같은 지역의 회원과 비교하여 키가 크거나 같은 사람이 전체의 몇 %인지 누적 백분율 구하기

```
SELECT addr, userName, height AS "키",
    (CUME_DIST() OVER (PARTITION BY addr ORDER BY height DESC)) * 100 AS "누적인원 백분율%"
  FROM userTBL;
```

addr	userName	키	누적인원 백분율%
경기	이휘재	180	50
경기	신동엽	176	100
경남	김제동	173	50
경남	이경규	170	100
경북	강호동	182	100
서울	박수홍	183	25
서울	유재석	178	50
서울	김용만	177	75
서울	김국진	171	100
충남	남희석	180	100

2-4 피벗

- 피벗(pivot)
  - 한 열에 포함된 여러 값을 여러 열로 변환하여 출력하고 필요하면 집계까지 수행하는 기능
  - 수행 결과 피벗 테이블이 생성

	uName	season	amount
▶	유재석	겨울	10
	강호동	여름	15
	유재석	가을	25
	유재석	봄	3
	유재석	봄	37
	강호동	겨울	40
	유재석	여름	14
	유재석	겨울	22
	강호동	여름	64

	uName	봄	여름	가을	겨울
▶	유재석	40	14	25	32
	강호동	NULL	79	NULL	40

[실습 6-5] 피벗 테이블 만들기

교재 207~208p 참고

1 샘플 테이블 만들기

1-1 샘플 테이블 생성

```
USE cookDB;
CREATE TABLE pivotTest
( uName CHAR(3),
  season CHAR(2),
  amount INT
);
```

1-2 데이터 9건 삽입

```
INSERT INTO pivotTest VALUES ('유재석', '겨울', 10);
INSERT INTO pivotTest VALUES ('강호동', '여름', 15);
INSERT INTO pivotTest VALUES ('유재석', '가을', 25);
INSERT INTO pivotTest VALUES ('유재석', '봄', 3);
INSERT INTO pivotTest VALUES ('유재석', '봄', 37);
INSERT INTO pivotTest VALUES ('강호동', '겨울', 40);
INSERT INTO pivotTest VALUES ('유재석', '여름', 14);
INSERT INTO pivotTest VALUES ('유재석', '겨울', 22);
INSERT INTO pivotTest VALUES ('강호동', '여름', 64);
SELECT * FROM pivotTest;
```

[실습 6-5] 피벗 테이블 만들기

교재 207~208p 참고

2 피벗 테이블 만들기

2-1 SUM( ) 함수, CASE 문, GROUP BY 절을 활용하여 피벗 테이블 만들기

```
SELECT uName,
    SUM(CASE WHEN season='봄' THEN amount END) AS '봄',
    SUM(CASE WHEN season='여름' THEN amount END) AS '여름',
    SUM(CASE WHEN season='가을' THEN amount END) AS '가을',
    SUM(CASE WHEN season='겨울' THEN amount END) AS '겨울'
FROM pivotTest
GROUP BY uName;
```

3-1 WITH 절과 CTE의 개요

- WITH 절
  - 기존의 뷰, 파생 테이블, 임시 테이블 등을 더 간결하게 표현하는 CTE(Common Table Expression)를 포함한 구문

- CTE
  - 비재귀적(non-recursive) CTE와 재귀적 (recursive) CTE로 구분

3-2 비재귀적 CTE

- 비재귀적 CTE의 형식

```
WITH CTE_테이블이름(열이름)
AS
(
   <쿼리문>
)
SELECT 열이름 FROM CTE_테이블이름;
```

- 구매 테이블(buyTBL)에서 총구매액을 구하는 쿼리문

```
USE cookDB;
SELECT userid AS '사용자', SUM(price * amount) AS '총구매액'
  FROM buyTBL GROUP BY userid;
```

	사용자	총구매액
▶	KHD	1210
	KJD	75
	KYM	200
	LHJ	95
	PSH	1920

3-2 비재귀적 CTE

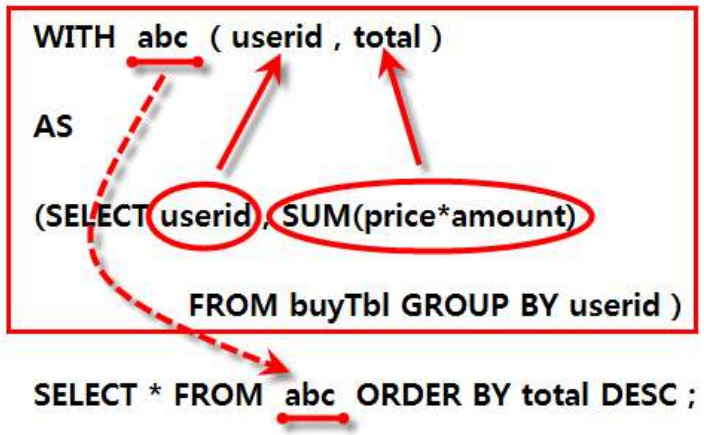
- CTE를 사용하면 쿼리가 단순해짐

```
SELECT * FROM abc ORDER BY 총구매액 DESC;
```

- CTE를 이용하여 총구매액이 많은 사용자 순으로 정렬하는 전체 쿼리

```
WITH abc(userid, total)
AS
(SELECT userid, SUM(price * amount)
      FROM buyTBL GROUP BY userid)
SELECT * FROM abc ORDER BY total DESC;
```

	userid	total
▶	PSH	1920
	KHD	1210
	KYM	200
	LHJ	95
	KJD	75



3-2 비재귀적 CTE

- 1단계: '각 지역별로 가장 키가 큰 사람'을 뽑는 쿼리를 작성한다.

```
SELECT addr, MAX(height) FROM userTBL GROUP BY addr
```

- 2단계: 위 쿼리를 WITH 구문으로 묶는다.

```
WITH cte_userTBL(addr, maxHeight)
AS
  (SELECT addr, MAX(height) FROM userTBL GROUP BY addr)
```

- 3단계: '키의 평균'을 구하는 쿼리를 작성한다.

```
SELECT AVG(키) FROM CTE_테이블이름
```

- 4단계: 2단계와 3단계의 쿼리를 합친다. 키의 평균을 실수로 만들기 위해 키에 1.0을 곱하여 실수로 변환한다.

```
WITH cte_userTBL(addr, maxHeight)
AS
   (SELECT addr, MAX(height) FROM userTBL GROUP BY addr)
SELECT AVG(maxHeight * 1.0) AS '각 지역별 최고키의 평균' FROM
cte_userTBL;
```

	각 지역별 최고키의 평균
▶	179.60000

3-2 비재귀적 CTE

- CTE는 중복 CTE로 사용 가능
  - CCC의 쿼리문에서는 AAA나 BBB를 참조할 수 있지만 AAA나 BBB의 쿼리문에서는 CCC를 참조할 수 없음

```
WITH
AAA (칼럼들)
AS (AAA의 쿼리문),
    BBB (칼럼들)
       AS (BBB의 쿼리문),
    CCC (칼럼들)
       AS (CCC의 쿼리문)
SELECT * FROM [AAA 또는 BBB 또는 CCC]
```