4

4장 : SQL 고급 문법

이번 장에서는 고급SQL을 활용하기 위한 데이터 형식과 조인에 대해 살펴보고, SQL 프로그래밍을 가능하도록 도와주는 스토어드 프로시저 형식에 대해 알아본다.

4-1 : MySQL의 데이터 형식

테이블을 만들 때는 **데이터 형식**을 설정해야 한다. 데이터 형식에는 크게 **숫자형**, **문자형**, **날짜형**이 있으며 세부적으로 더 나뉘어지기도 한다. 저장될 데이터의 형태가 다양하기 때문에 각 데이터에 맞는 데이터 형식을 지정함으로써 효율적으로 저장해야 한다.

예를 들어 이름을 저장하기 위해 내부적으로 100글자를 저장할 칸을 준비하는 것은 상당한 낭비이다. 따라서 데이터 형식에 대해서 제대로 이해하고, 적절한 데이터 형식을 고르는 안목을 키우면 SQL도 더 고급스럽게 작성할 수 있다.

MySQL에서 제공하는 데이터 형식의 종류는 수십 개 정도이고, 각 데이터 형식마다 크기나 표현할 수 있는 숫자의 범위가 다르다. 이를 모두 외울 필요는 없으니 자주 사용하는 것만 살펴본다.

정수형

정수형은 소수점이 없는 숫자이며 정수형의 크기와 범위는 다음과 같다.

데이터 형식	바이트 수	숫자 범위
TINYINT	1	-128~127
SMALLINT	2	-32768~32767
INT	4	약 -21억~+21억
BIGINT	8	약 -900경~+900경

3장에서 만들었던 인터넷 마켓의 회원 테이블에서 인원수열은 **INT**로, 평균 키열은 **SMALLINT**로 지정했다.

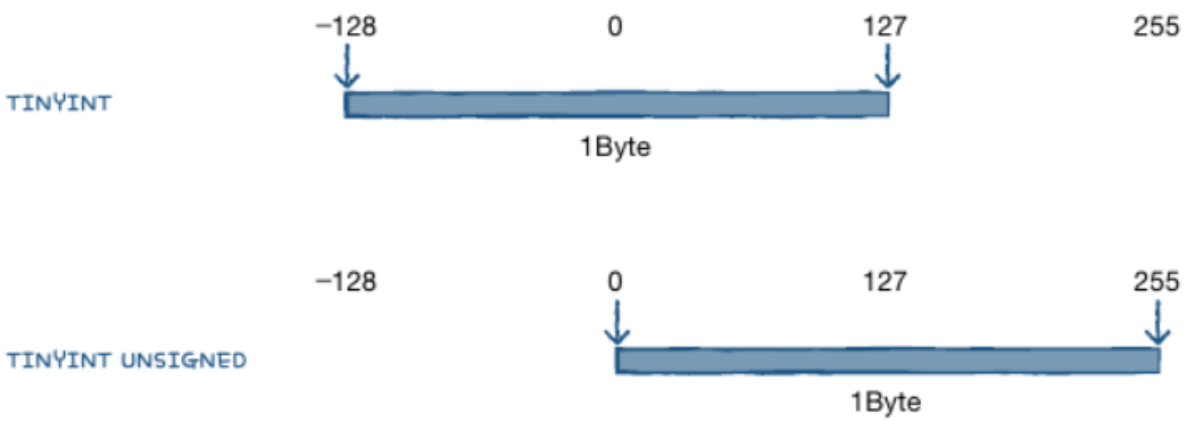
```
USE market_db;
CREATE TABLE member
( mem_id        CHAR(8) NOT NULL PRIMARY KEY,
mem_name      VARCHAR(10) NOT NULL,
mem_number
INT NOT NULL,
addr              CHAR(2) NOT NULL,
phone1      CHAR(3),
phone2          CHAR(8),

height       SMALLINT,

debut_date DATE
);
```

평균 키 열은 **SMALLINT**로 지정해서 -32768~32767까지 저장할 수 있지만 키 역시 이렇게까지 필요가 없으므로 **TINYINT**를 고려할 수 있지만 127CM 보다 큰 사람이 있으므로 범위가 부족하다.

이를 해결하기 위해 값의 범위가 0부터 시작되는 **UNSIGNED** 예약어를 사용할 수 있다. UNSIGNED 예약어는 말 그대로 **값의 범위가 0부터 시작**되게 만들어주는 예약어이다. 다음 그림과 같이 TINYINT와 TINYINT UNSIGNED 모두 1바이트의 크기이다. 1바이트는 256개를 표현하므로 -128~+127로 표현하거나, 0~255로 표현하거나 모두 256개를 표현할 수 있다.



따라서 위의 회원 테이블 작성문에서 회원 수나, 키 모두 음수값을 가질 일이 없으므로 **UNSIGNED** 예약어를 사용하는 것이 더 올바른 방법이다.

```
USE market_db;
CREATE TABLE member
( mem_id        CHAR(8) NOT NULL PRIMARY KEY,
mem_name      VARCHAR(10) NOT NULL,
mem_number
INT UNSIGNED NOT NULL,
addr              CHAR(2) NOT NULL,
phone1      CHAR(3),
phone2          CHAR(8),

height       SMALLINT UNSIGNED,

debut_date DATE
);
```

다른 정수형들도 마찬가지로 UNSIGNED를 붙이면 0부터 범위가 지정된다. 예를 들어 **SMALLINT UNSIGNED**는 0부터 65535까지 저장이 된다.

문자형

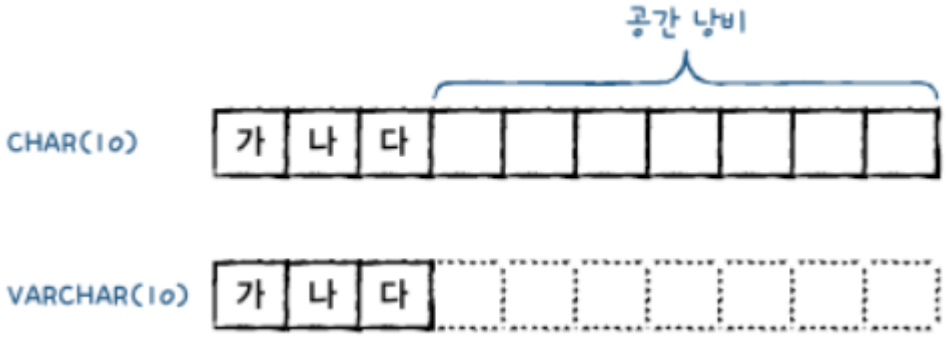
문자형은 글자를 저장하기 위해 사용하며, 입력할 최대 글자의 개수를 지정해야 한다. 대표적인 문자형은 다음과 같다.

데이터 형식	바이트 수
CHAR (개수)	1~255
VARCHAR (개수)	1~16383

CHAR는 문자는 의미하는 CHARACTER의 약자로, 고정길이 문자형이라고 부르며 말 그대로 자릿수가 고정되어 있다. 이와 달리 **VARCHAR** (VARIABLE CHARACTER)는 가변 길이 문자형이다.

예를 들어 **CHAR(10)**에 '가나다' 3글자만 저장하면 3자리를 사용하고 뒤의 7자리는 낭비하게 된다. **VARCHAR (10)**에 '가나다'를 저장하면 3자리만 사용하게 된다.

좀 더 엄밀히 말하면, **CHAR는 정의된 길이만큼 메모리를 항상 차지하고, VARCHAR는 실제 저장된 문자 수만큼만 메모리를 사용한다.**



문자 개수를 정의하지 않을 시, 1로 설정된다.

VARCHAR가 CHAR보다 공간을 효율적으로 사용한다. 하지만 MySQL 내부적으로 성능(빠른 속도)면에서는 CHAR로 설정하는 것이 조금 더 좋다. 따라서 CHAR는 주민등록번호나 성별 등 **글자의 개수가 고정된 경우**, VARCHAR는 **글자의 개수가 다양하거나 변동될 경우** 사용하는 것이 올바른 사용 방법이 될 것이다.

```
USE market_db;
CREATE TABLE member
( mem_id
CHAR(8) NOT NULL PRIMARY KEY,
mem_name
VARCHAR(10) NOT NULL,
mem_number
INT NOT NULL,
addr              CHAR(2) NOT NULL,
phone1
CHAR(3),
phone2
CHAR(8),

height        SMALLINT,

debut_date DATE
);
```

회원 테이블의 문자형을 확인해보자. mem_id에 입력되는 데이터를 살펴보면 모두 3글자로 입력되는데 데이터 형식은 8글자로 설정되어 있다. 3글자로 줄여도 되지만 추후 더 긴 글자가 나올 수 있다고 가정하면 8로 설정해도 되고 VARCHAR(8)로 설정해도 문제 없다.

관심있게 볼 부분은 연락처 국번(phone1)과 연락처 전화번호(phone2)이다. 연락처 국번은 02, 031, 055 등 과 같이 제일 앞에 0이 붙어야 하는데 **정수형으로 지정하면 0이 사라진다.** 따라서 문자형으로 지정되었다.

다음으로 연락처 전화번호는 모두 숫자로 이루어져 정수형으로 지정해야 되지 않나 생각할 수 있지만, 전화번호는 **숫자로서 의미가 없다.** 숫자로서 의미를 가지려면 **1.연산에 의미가 있다. / 2. 대소비교가 의미가 있다.**

두 가지 조건 중 한 가지는 충족을 해야 한다.

대용량 데이터 형식

문자형인 CHAR는 최대 255자까지, VARCHAR는 최대 16383까지 지정이 가능하다. 더 큰 데이터를 저장하려면 다음과 같은 형식을 사용한다.

데이터 형식		바이트 수
TEXT 형식	TEXT	1~65535
	LONGTEXT	1~4294967295
BLOB 형식	BLOB	1~65535
	LONGBLOB	1~4294967295

추가로 TINYTEXT, MEDIUMTEST, TINYBLOB, MEDIUMBLOB 등도 있지만 잘 사용하지 않는다.

TEXT로 지정하면 최대 65535자까지 저장할 수 있으며 **LONGTEXT**로 지정하면 최대 42억자까지 저장이 가능하다.

BLOB은 Binary Long Object의 약자로 글자가 아닌 **이미지, 동영상과 같은 비정형 데이터**를 위한 데이터형식 이라고 생각하면 된다. 이런 것들을 이진(Binary) 데이터라고 부른다. 테이블에 사진이나 동영상 같은 것을 저장하고 싶다면 **BLOB**이나 **LONGBLOB**으로 데이터 형식을 지정해야 한다.

LONGTEXT 및 LONGBLOB으로 데이터 형식을 설정하면 각 데이터는 최대 4GB까지 입력할 수 있다.

실수형

실수형은 소수점이 있는 숫자를 저장할 때 사용한다.

데이터 형식	바이트 수	설명
FLOAT	4	소수점 아래 7자리까지 표현
DOUBLE	8	소수점 아래 15자리까지 표현

날짜형

날짜형은 날짜 및 시간을 저장할 때 사용한다.

데이터 형식	바이트 수	설명
DATE	3	날짜만 저장. **YYYY-MM-DD** 형식으로 사용
TIME	3	시간만 저장. **HH:MM:SS** 형식으로 사용
DATETIME	8	날짜 및 시간을 저장. **YYYY-MM-DD HH:MM:SS** 형식으로 사용

테이블을 생성할 때 필요한 조건에 따라 사용하면 된다.

변수의 사용

SQL도 다른 프로그래밍 언어처럼 **변수**를 선언하고 사용할 수 있다. 변수 선언과 값의 대입은 다음 형식을 따른다.

SET @변수이름 = 변수의 값 ; →변수 선언 및 값 대입

SELECT @변수이름 ; →변수 값 출력

변수는 MySQL 워크벤치를 재시작할 때까지는 유지되지만, 종료하면 사라진다.

사용예시

```sql
USE MARKET_DB;
SET @MYVAR1 = 5;     ## (1) : 변수 선언 후 정수 대입
SET @MYVAR2 = 4.5;   ## (1) : 변수 선언 후 실수 대입

SELECT @MYVAR1;      ## (2) : 변수 출력
SELECT @MYVAR1+@MYVAR2; ## (3) : 변수끼리 연산 후 출력

SET @TXT = '가수이름==>'; ## (4) : 변수 선언 후 문자열 대입
SET @HEIGHT = 166;       ## (4) : 변수 선언 후 정수 대입
SELECT @TXT, MEM_NAME FROM MEMBER WHERE HEIGHT > @HEIGHT;  ## (5) : 테이블 조회하면서 변수 활용
```

	@TXT	MEM_NAME
▶	가수이름==>	소녀시대
	가수이름==>	잇지
	가수이름==>	트와이스

(5)의 결과

SELECT문에서 행의 개수를 제한하는 **LIMIT**에도 변수를 사용해본다. 이 SQL은 SELECT문에서 오류가 발생하는데, LIMIT에는 변수를 사용할 수 없기 때문에 문법 상 오류가 발생한다.

```sql
SET @COUNT=3;
SELECT MEM_NAME,HEIGHT FROM MEMBER ORDER BY HEIGHT LIMIT @COUNT;
```

Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use ne...

이를 해결하는 것이 **PREPARE**와 **EXECUTE**이다. PREPARE는 실행하지 않고 SQL문만 준비하고 EXECUTE에서 실행하는 방식이다.

```sql
SET @COUNT=3; ##(1)
PREPARE MYSQL FROM 'SELECT MEM_NAME, HEIGHT FROM MEMBER ORDER BY HEIGHT LIMIT ?'; ##(2)
EXECUTE MYSQL USING @COUNT; ##(3)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MEM_NAME	HEIGHT
오마이걸	160
레드벨벳	161
우주소녀	162

(1) : **COUNT** 변수에 3을 대입

(2) : **PREPARE**는 **'SELECT ~~LIMIT ?'**문을 실행하지 않고 MYSQL이라는 이름으로 준비만 해놓는다. 주의해야할 점은 LIMIT 다음에 오는 물음표인데 이는 **'현재는 모르지만 나중에 채워짐'** 정도로 이해하면 된다.

(3) : **EXECUTE**로 MYSQL에 저장된 SELECT 문을 실행할 때, **USING**으로 물음표에 @COUNT 변수의 값을 대입한다.

결론적으로 위의 SQL은 다음과 같다.

SELECT MEM_NAME, HEIGHT FROM MEMBER ORDER BY HEIGHT LIMIT 3;

데이터 형태 변환

문자형을 정수형으로 바꾸거나, 반대로 정수형을 문자형으로 바꾸는 것을 데이터의 **형 변환(Type conversion)**이라고 부른다. 형 변환에는 **직접 함수를 사용해서 변환하는 명시적 변환(explicit conversion)**과 **별도의 지시 없이 자연스럽게 변환되는 암시적인 변환(implicit conversion)**이 있다.

함수를 이용한 명시적인 변환

데이터 형식을 변환하는 함수는 **CAST( ), CONVERT( )**이다.둘은 형식만 다를 뿐, 동일한 기능을 한다.

CAST(값 AS 데이터_형식 [(길이)])

CONVERT (값, 데이터_형식 [(길이)])

간단한 예를 살펴보면서 이해하자. 다음은 구매 테이블에서 평균 가격을 구하는 SQL이다. 결과로 실수값이 나왔다.

```
16 •    SELECT AVG(PRICE) AS '평균가격' FROM BUY;
```

Result Grid | Filter Rows: | Export:

평균 가격
142.9167

결과값을 정수형으로 표시하고 싶다면 다음과 같이 **CAST**나 **CONVERT** 함수를 사용해서 데이터 형식을 바꾼다. 두 함수 안에 올 수 있는 데이터 형식은 CHAR, SIGNED, UNSIGNED, DATE, TIME, DATETIME 등이다. **SIGNED는 부호가 있는 정수, UNSIGNED는 부호가 없는 정수를 의미한다.**

```
SELECT CAST(AVG(PRICE) AS SIGNED) '평균가격' FROM BUY;
##OR
SELECT CONVERT(AVG(PRICE),SIGNED) '평균가격' FROM BUY;
```

이번엔 날짜를 확인해본다. 다음과 같이 다양한 구분자를 사용하여 날짜형으로 변경이 가능하다.

```
SELECT CAST('2022$12$12' AS DATE);
SELECT CAST('2022/12/12' AS DATE);
SELECT CAST('2022%12%12' AS DATE);
SELECT CAST('2022@12@12' AS DATE);
```

결과는 모두 2022-12-12로 나온다.

SQL의 결과를 원하는 형태로 표현할 때도 사용할 수 있다. 가격과 수량을 곱한 실제 구매액을 표시하는 SQL을 다음과 같이 작성할 수 있다.

```
27 •    SELECT NUM, CONCAT(CAST(PRICE AS CHAR),'x',CAST(AMOUNT AS CHAR),'=') '가격x수량', PRICE*AMOUNT '구매액' FROM BUY LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

NUM	가격x 수량	구매액
1	30x2=	60
2	1000x1=	1000
3	200x1=	200
4	200x5=	1000
5	50x3=	150

가격과 수량은 정수지만, **CAST** 함수를 통해 문자로 바꿨다 **CONCAT()** 함수는 문자를 이어주는 역할을 하며, 여기서는 '가격*수량=' 과 같은 형태의 문자로 만들어서 출력했다.

암시적인 변환

암시적인 변환은 CAST나 CONVERT 함수를 사용하지 않고도 자연스럽게 형이 변환되는 것을 말한다.

다음 예시를 살펴보자 문자 '100'과 '200'을 더했다. 문자는 더할 수 없으므로 자동으로 숫자 100과 200으로 변환해서 덧셈이 수행되었다.

```
29 •    SELECT '100'+'200';
```

Result Grid | Filter Rows:

'100'+'200'
300

만약 문자 '100'과 '200'을 연결한 '100200'으로 만들려면 앞에서 배운 CONCAT 함수를 사용하면 된다.

```
31 •    SELECT CONCAT('100','200');
```

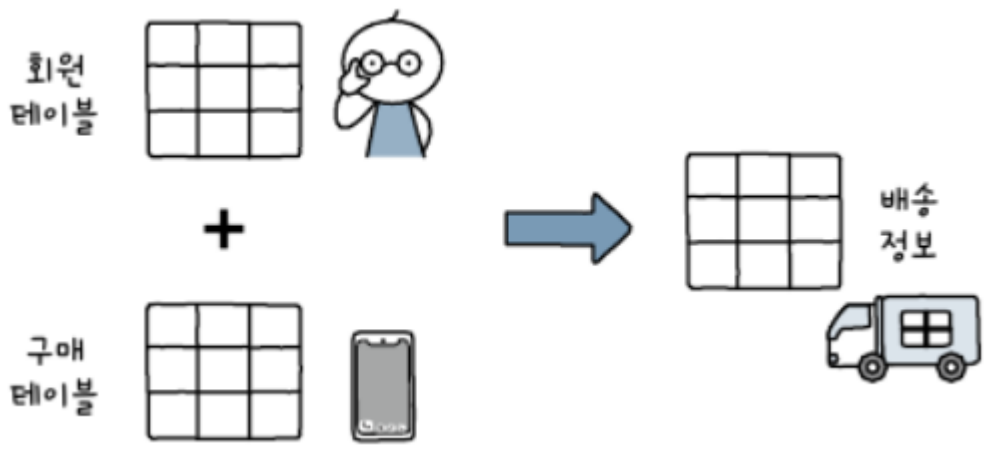
Result Grid | Filter Rows:

CONCAT('100','200')
100200

숫자와 문자를 CONCAT()으로 연결하면 **숫자가 문자**로 변하고, 수학 연산을 하면 **문자가 숫자**로 변한 후에 연산이 된다.

4-2 : 두 테이블을 묶는 조인

이제부터 두 개의 테이블이 서로 **관계되어 있는 상태**를 고려한다. **조인(JOIN)**이란 두 개의 테이블을 서로 묶어서 하나의 결과를 만들어 내는 것을 말한다. 두 테이블을 엮어야만 원하는 형태가 나오는 경우가 있는데 회원 테이블을 묶는 방법을 배운다.



내부 조인

두 테이블을 연결할 때 가장 많이 사용되는 것이 내부 조인이다. 그냥 조인이라고 하면 일반적으로 내부 조인을 의미하는 것이다.

일대다 관계의 이해

두 테이블의 조인을 위해서는 테이블이 **일대다(one to many)** 관계로 연결되어야 한다.

데이터베이스의 테이블은 하나로 구성되는 것보다 여러 정보를 **주제에 따라 분리해서 저장**하는 것이 효율적이다. 이 분리된 테이블은 서로 **관계(relation)**를 맺고 있다. 우리가 계속 사용하고 있는 인터넷 마켓 데이터베이스(market_db)의 회원 테이블과 구매 테이블이 그 예시이다.

일대다 관계란 한쪽 테이블에는 하나의 값만 존재해야 하지만, 연결된 다른 테이블에는 여러 개의 값이 존재할 수 있는 관계를 말한다.

예시

회원 테이블(member)

아이디	이름	인원	주소	국번	전화번호	평균 키	데뷔 일자
TWC	트와이스	9	서울	02	11111111	167	2015.10.19
BLK	블랙핑크	4	경남	055	22222222	163	2016.08.08
WMN	여자친구	6	경기	031	33333333	166	2015.01.15
OMY	오마이걸	7	서울			160	2015.04.21
GRL	소녀시대	8	서울	02	44444444	168	2007.08.02
ITZ	잇지	5	경남			167	2019.02.12
RED	레드벨벳	4	경북	054	55555555	161	2014.08.01
APN	에이핑크	6	경기	031	77777777	164	2011.02.10
SPC	우주소녀	13	서울	02	88888888	162	2016.02.25
MMU	마마무	4	전남	061	99999999	165	2014.06.19

PK

구매 테이블(buy)

순번	아이디	물품명	분류	단가	수량
1	BLK	지갑		30	2
2	BLK	맥북프로	디지털	1000	1
3	APN	아이폰	디지털	200	1
4	MMU	아이폰	디지털	200	5
5	BLK	청바지	패션	50	3
6	MMU	에어팟	디지털	80	10
7	GRL	혼공SQL	서적	15	5
8	APN	혼공SQL	서적	15	2
9	APN	청바지	패션	50	1
10	MMU	지갑		30	1
11	APN	혼공SQL	서적	15	1
12	MMU	지갑		30	4

PK FK

회원 테이블에서 블랙핑크의 아이디는 'BLK'로 1명**(one)** 밖에 없다. 그래서 회원 테이블의 아이디를 **기본 키(PK)**로 지정했다. 구매 테이블의 아이디에서는 3개의 BLK를 찾을 수 있다. 즉, 회원은 1명이지만 이 회원은 구매를 여러 번**(many)** 할 수 있다. 그래서 구매 테이블의 아이디는 기본 키가 아닌 **외래 키(Foreign key, FK)**로 설정했다.

> 💬 일대다 관계는 주로 기본 키와 외래 키 관계로 맺어져 있다. 그래서 일대다 관계를 **'PK-FK 관계'** 라고 부르기도 한다.

인터넷 마켓 데이터 이외에도 회사원 테이블과 급여 테이블도 마찬가지로 1명의 회사원이 여러 번의 급여를 받으므로 일대다 관계라고 할 수 있다.

내부 조인의 기본

일반적으로 조인이라고 부르는 것은 **내부 조인(inner join)**을 말하는 것으로, 조인 중에 가장 많이 사용된다. 조인은 3개 이상의 테이블로 할 수 있지만 대부분 2개로 조인하므로 먼저 2개의 테이블을 내부 조인하는 방법을 살펴본다.

내부 조인의 형식은 다음과 같다.

SELECT <열 목록> FROM <첫 번째 테이블> INNER JOIN <두 번째 테이블> ON <조인 조건> [WHERE 검색 조건]

INNER JOIN을 그냥 JOIN이라고만 써도 INNER JOIN으로 인식한다.

구매 테이블에는 물건을 구매한 회원의 아이디와 물건 등의 정보만 있다. 이 물건을 배송하기 위해서는 구매한 회원의 주소 및 연락처를 알아야 한다. 이 회원의 주소, 연락처를 알기 위해 정보가 있는 회원 테이블과 결합하는 것이 내부 조인이다.

구매 테이블에서 GRL 이라는 아이디를 가진 사람이 구매한 물건을 발송하기 위해 다음과 같이 조인해서 이름/주소/연락처 등을 검색할 수 있다.

```
34 •    SELECT * FROM BUY
35          INNER JOIN MEMBER
36          ON BUY.MEM_ID = MEMBER.MEM_ID
37          WHERE BUY.MEM_ID='GRL';
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

num	mem_id	prod_name	group_name	price	amount	mem_id	mem_name	mem_number	addr	phone1	phone2	height	debut_date
7	GRL	혼공SQL	서적	15	5	GRL	소녀시대	8	서울	02	44444444	168	2007-08-02

두 개의 테이블을 조인하는 경우 동일한 열 이름이 존재한다면 꼭 테이블_이름.열_이름 형식으로 표기해야 된다.

만약 WHERE을 생략하면 1번째 BLK부터 MMU까지 구매 테이블의 모든 행이 회원 테이블과 결합하게 된다.

내부 조인 과정

❷ GRL과 동일한 아이디 검색 ❶ GRL을 추출

회원 테이블(member)

아이디	이름	인원	주소	국번	전화번호	평균 키	데뷔 일자
TWC	트와이스	9	서울	02	11111111	167	2015.10.19
BLK	블랙핑크	4	경남	055	22222222	163	2016.08.08
WMN	여자친구	6	경기	031	33333333	166	2015.01.15
OMY	오마이걸	7	서울			160	2015.04.21
GRL	소녀시대	8	서울	02	44444444	168	2007.08.02
ITZ	잇지	5	경남			167	2019.02.12
RED	레드벨벳	4	경북	054	55555555	161	2014.08.01
APN	에이핑크	6	경기	031	77777777	164	2011.02.10
SPC	우주소녀	13	서울	02	88888888	162	2016.02.25
MMU	마마무	4	전남	061	99999999	165	2014.06.19

PK

❸ 두 행을 결합

구매 테이블(buy)

순번	아이디	물품명	분류	단가	수량
1	BLK	지갑		30	2
2	BLK	맥북프로	디지털	1000	1
3	APN	아이폰	디지털	200	1
4	MMU	아이폰	디지털	200	5
5	BLK	청바지	패션	50	3
6	MMU	에어팟	디지털	80	10
7	GRL	혼공SQL	서적	15	5
8	APN	혼공SQL	서적	15	2
9	APN	청바지	패션	50	1
10	MMU	지갑		30	1
11	APN	혼공SQL	서적	15	1
12	MMU	지갑		30	4

PK FK

(1) : 구매 테이블의 MEM_ID인 'GRL'을 추출한다.

(2): 'GRL'과 동일한 값을 회원 테이블의 MEM_ID 열에서 검색한다.

(3): 'GRL'이라는 아이디를 찾으면 구매 테이블과 회원 테이블의 두 행을 결합한다.

내부 조인의 간결한 표현

결과가 너무 많아 복잡해 보이므로 이번에는 필요한 열만 추출해본다.

```
39 •  SELECT BUY.MEM_ID,MEM_NAME,PROD_NAME,ADDR, CONCAT(PHONE1,PHONE2) '연락처'
40        FROM BUY
41        INNER JOIN MEMBER
42        ON BUY.MEM_ID=MEMBER.MEM_ID LIMIT 5;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

MEM_ID	MEM_NAME	PROD_NAME	ADDR	연락처
BLK	블랙핑크	지갑	경남	05522222222
BLK	블랙핑크	맥북프로	경남	05522222222
APN	에이핑크	아이폰	경기	03177777777
MMU	마마무	아이폰	전남	06199999999
BLK	블랙핑크	청바지	경남	05522222222

좀 더 명확히 하기 위해서 SELECT 다음의 열 이름에도 모두 테이블_이름.열_이름 형식으로 작성하면 각 열이 어느 테이블에 속한 것인지 명확하지만 코드가 너무 길어 복잡해 보인다.

```
SELECT BUY.MEM_ID,MEMBER.MEM_NAME,BUY.PROD_NAME,MEMBER.ADDR, CONCAT(MEMBER.PHONE1,MEMBER.PHONE2) '연락처'
    FROM BUY
    INNER JOIN MEMBER
    ON BUY.MEM ID=MEMBER.MEM ID LIMIT 5;
```

이를 간결하게 표현하기 위해서는 다음과 같이 FROM 절에 나오는 테이블 이름 뒤에 별칭(Alias)를 주어 사용하는 방식을 적극 권장한다.

```
SELECT B.MEM_ID, M.MEM_NAME, B.PROD_NAME, M.ADDR, CONCAT(M.PHONE1,M.PHONE2) '연락처'
    FROM BUY B
    INNER JOIN MEMBER M
    ON B.MEM_ID=M.MEM_ID LIMIT 5;
```

내부 조인 활용

이번엔 **전체 회원**의 아이디/이름/구매한 제품/주소를 출력한다. 결과는 보기 쉽게 회원 아이디 순으로 정렬한다.

```
SELECT M.MEM_ID,M.MEM_NAME,B.PROD_NAME,M.ADDR
    FROM BUY B
    INNER JOIN MEMBER M
    ON B.MEM_ID=M.MEM_ID
    ORDER BY M.MEM_ID;
```

구매 테이블의 목록이 12건이었으므로 이상 없이 잘 나왔다. 결과는 문제가 없지만, 위에서 말한 **전체 회원** 과는 차이가 있다. 위의 SQL은 전체 회원이 아닌 **구매기록이 있는 회원들의 목록이다.** 지금까지 사용한 내부 조인은 두 테이블에 모두 있는 내용만 조인되는 방식이다. 만약 양쪽 중에 한 곳이라도 내용이 있을 때 조인하려면 **외부 조인(Outer join)**을 사용해야한다.

내부 조인이 양쪽에 모두 있는 내용만 나오기 때문에 유용한 경우도 있다. 예를 들어 인터넷 마켓 운영자라면 사이트에서 한 번이라도 구매한 기록이 있는 회원들에게 홍보 안내문을 발송할 수도 있다.

이런 경우라면 앞의 SQL처럼 내부 조인을 사용해서 추출한 회원에게만 안내문을 발송하면 된다. 그리고 어차피 중복된 이름은 필요가 없으므로 3장에서 배운 **DISTINCT** 문을 활용해서 회원의 주소를 조회할 수 있다.

```sql
SELECT DISTINCT M.MEM_ID,M.MEM_NAME,M.ADDR
    FROM BUY B
    INNER JOIN MEMBER M
    ON B.MEM_ID=M.MEM_ID
    ORDER BY M.MEM_ID;
```

Result Grid | Filter Rows: | Export:

MEM_ID	MEM_NAME	ADDR
APN	에이핑크	경기
BLK	블랙핑크	경남
GRL	소녀시대	서울
MMU	마마무	전남

외부 조인

내부 조인은 두 테이블에 모두 데이터가 있어야만 결과가 나온다. 이와 달리 **외부 조인(outer join)**은 한쪽에만 데이터가 있어도 결과가 나온다.

외부 조인의 기본

외부 조인의 형식은 다음과 같다. 내부 조인보다 조금 복잡해 보이지만 사용 방법은 거의 비슷하다.

SELECT <열 목록>

FROM <첫 번째 테이블(left table)>

<LEFT | RIGHT | FULL> OUTER JOIN <두 번째 테이블(right table)>

ON <조인될 조건>

[WHERE 검색 조건] ;

내부 조인의 예시에서 해결하지 못한 **'전체 회원'**의 구매 기록 출력을 외부 조인으로 해본다.

```sql
SELECT M.MEM_ID, M.MEM_NAME,B.PROD_NAME, M.ADDR
    ###############
    FROM MEMBER M
    LEFT OUTER JOIN BUY B
    ############### 왼쪽에 있는 회원 테이블을 기준으로 외부 조인
    ON M.MEM_ID = B.MEM_ID
    ORDER BY M.MEM ID;
```

MEM_ID	MEM_NAME	PROD_NAME	ADDR
APN	에이핑크	아이폰	경기
APN	에이핑크	홍공SQL	경기
APN	에이핑크	청바지	경기
APN	에이핑크	홍공SQL	경기
BLK	블랙핑크	지갑	경남
BLK	블랙핑크	맥북프로	경남
BLK	블랙핑크	청바지	경남
GRL	소녀시대	홍공SQL	서울
ITZ	잇지	NULL	경남
MMU	마마무	아이폰	전남
MMU	마마무	에어팟	전남
MMU	마마무	지갑	전남
MMU	마마무	지갑	전남
OMY	오마이걸	NULL	서울
RED	레드벨벳	NULL	경북
SPC	우주소녀	NULL	서울
TWC	트와이스	NULL	서울
WMN	여자친구	NULL	경기

LEFT OUTER JOIN 문의 의미를 **'왼쪽 테이블의 내용은 모두 출력되어야 한다'** 정도로 해석하면 된다. LEFT OUTER JOIN을 줄여서 LEFT JOIN이라고만 써도 된다.

RIGHT OUTER JOIN으로 동일한 결과를 출력하려면 다음과 같이 단순히 왼쪽과 오른쪽 테이블의 위치만 바꿔주면 된다.

```sql
SELECT M.MEM_ID, M.MEM_NAME,B.PROD_NAME, M.ADDR
FROM BUY B
RIGHT OUTER JOIN MEMBER M
ON M.MEM_ID = B.MEM_ID
ORDER BY M.MEM_ID;
```

외부 조인의 활용

내부 조인으로 구매한 기록이 있는 회원들의 목록만 추출해서 감사문을 보냈다. 이번에는 반대로 회원으로 가입만 하고, 한 번도 구매한 적이 없는 회원의 목록을 추출해본다.

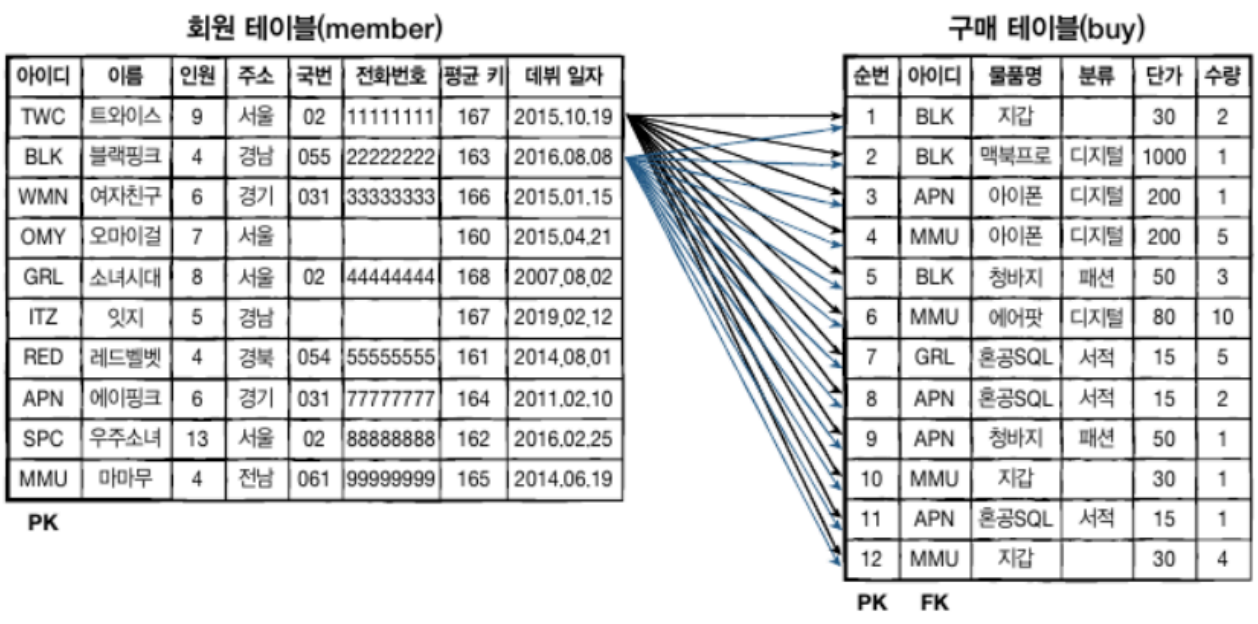
```sql
SELECT DISTINCT M.MEM_ID, B.PROD_NAME, M.MEM_NAME, M.ADDR
FROM MEMBER M
LEFT OUTER JOIN BUY B
ON M.MEM_ID =B.MEM_ID
WHERE B.PROD_NAME IS NULL
ORDER BY M.MEM_ID;
```

FULL OUTER JOIN은 왼쪽 외부 조인과 오른쪽 외부 조인이 합쳐진 것이라고 생각하면 된다. **왼쪽이든 오른쪽이든 한쪽에 들어 있는 내용이면 출력한다.** 자주 사용하지 않는다.

상호 조인

상호 조인(cross join)은 한쪽 테이블의 모든 행과 다른 쪽 테이블의 모든 행을 조인시키는 기능을 말한다. 따라서 상호 조인 결과의 전체 행 개수는 두 테이블의 각 행의 개수를 곱한 개수가 된다.

그림을 통해 알아본다.



아이디	이름	인원	주소	국번	전화번호	평균 키	데뷔 일자
TWC	트와이스	9	서울	02	11111111	167	2015.10.19
BLK	블랙핑크	4	경남	055	22222222	163	2016.08.08
WMN	여자친구	6	경기	031	33333333	166	2015.01.15
OMY	오마이걸	7	서울			160	2015.04.21
GRL	소녀시대	8	서울	02	44444444	168	2007.08.02
ITZ	잇지	5	경남			167	2019.02.12
RED	레드벨벳	4	경북	054	55555555	161	2014.08.01
APN	에이핑크	6	경기	031	77777777	164	2011.02.10
SPC	우주소녀	13	서울	02	88888888	162	2016.02.25
MMU	마마무	4	전남	061	99999999	165	2014.06.19

순번	아이디	물품명	분류	단가	수량
1	BLK	지갑		30	2
2	BLK	맥북프로	디지털	1000	1
3	APN	아이폰	디지털	200	1
4	MMU	아이폰	디지털	200	5
5	BLK	청바지	패션	50	3
6	MMU	에어팟	디지털	80	10
7	GRL	혼공SQL	서적	15	5
8	APN	혼공SQL	서적	15	2
9	APN	청바지	패션	50	1
10	MMU	지갑		30	1
11	APN	혼공SQL	서적	15	1
12	MMU	지갑		30	4

상호 조인을 카타시안 곱(Cartesian product)라고도 부른다.

회원 테이블의 첫 행은 구매 테이블의 모든 행과 조인된다. 나머지 행도 마찬가지이다. 이런 식으로 회원 테이블의 모든 행이 구매 테이블의 모든 행과 결합되어 최종적으로 각 테이블의 행끼리 곱한 개수의 행이 결과로 생성된다.

```sql
SELECT * FROM BUY CROSS JOIN MEMBER;
```

상호 조인은 다음과 같은 특징을 갖는다.

- ON 구문을 사용할 수 없다.
- 랜덤으로 조인되기 때문에 결과의 내용은 의미가 없다
- 상호 조인의 주 용도는 테스트를 위한 데이터를 생성할 때이다.

자체 조인

자체 조인(self join)은 자신이 자신과 조인한다는 의미이다. 그래서 자체 조인은 1개의 테이블을 사용한다. 또, 별도의 문법이 있는 것은 아니고 1개로 조인하면 자체 조인이 되는 것이다.

자체 조인의 형식은 다음과 같다. 테이블이 1개지만 다른 별칭을 사용하여 서로 다른 것처럼 사용하면 된다.

SELECT <열 목록> FROM <테이블> 별칭A

INNER JOIN <테이블> 별칭B

ON <조인될 조건>

[WHERE 검색 조건];

다음 그림과 같이 2개의 테이블이 조인되는 것처럼 구성된다. 이렇듯 하나의 테이블에 같은 데이터가 있지만 2개 이상의 열로 존재할 때 자체 조인을 할 수 있다.

4-3 : SQL 프로그래밍

SQL은 앞서 배운 것처럼 SELECT, INSERT, UPDATE, DELETE 등을 사용하기에 파이썬과 같은 프로그래밍 언어와는 많이 달라 보인다. 하지만 필요하다면 SQL만으로도 프로그래밍이 가능하다.

스토어드 프로시저는 MySQL에서 프로그래밍이 필요할 때 사용하는 데이터베이스 개체이다. **SQL 프로그래밍은 기본적으로 스토어드 프로시저 안에서 만들어야 한다.**

스토어드 프로시저는 다음과 같은 구조를 갖는다.

```
DELIMITER $$
CREATE PROCEDURE 스토어드_프로시저_이름( )
BEGIN
    프로그래밍_코드 작성
END $$
DELIMITER ;
CALL 스토어드_프로시저_이름( );
```

스토어드 프로시저는 **DELIMITER $$ ~ END $$** 안에 작성하고 CALL로 호출한다. 일반적으로 구분 문자(DELIMITER)는 $$를 많이 사용하지만, 원한다면 /, &, @ 등을 사용해도 상관없다. 다른 기호와 중복되지 않게 기호 2개를 연속해서 사용한다.

IF 문

IF 문은 조건문으로 가장 많이 사용되는 프로그래밍 문법 중 하나이다. 구조를 살펴보자

IF <조건식> THEN

SQL 문장들

END IF;

가운데의 **SQL 문장들**이 한 문장이라면 그 문장만 써도 되지만, 두 문장 이상이 처리되어야 할 때는 **BEGIN~END** 로 묶어줘야 하므로 습관적으로 **BEGIN~END**로 묶는 걸 권장한다.

```
DROP PROCEDURE IF EXISTS IFPROC1; ##IFPROC1()이 존재하면 삭제

DELIMITER $$                      ##세미콜론으로는 SQL의 끝인지,스토어드 프로시저의 끝인지 구별이 안되기 때문에 $$ 사용
CREATE procedure IFPROC1()        ##이름지정
BEGIN
    IF 100 = 100 THEN             ## 조건 지정
        SELECT '100 IS 100.';
    END IF;
END $$
DELIMITER ;

CALL IFPROC1();                   ##호출
```

IF ~ ELSE 문

IF ~ ELSE 문은 조건에 따라 다른 부분을 수행한다. 조건식이 참이라면 'SQL문장1' 을 실행하고, 거짓이면 'SQL문장2'를 실행한다.

```
DELIMITER $$
CREATE procedure IFPROC2()
BEGIN
    DECLARE MYNUM INT;       ##DECLARE 예약어를 사용해서 MYNUM 변수,형태 선언
    SET MYNUM = 200;         ##SET 예약어로 MYNUM 변수에 200 대입
    IF MYNUM =100 THEN
        SELECT 'IS 100';
    ELSE
        SELECT 'NOT 100';
    END IF;
END $$
DELIMITER ;

CALL IFPROC1();
```

IF 문 활용

기존 테이블과 함께 IF 문을 활용해본다. 아이디가 APK인 회원의 데뷔 일자가 5년이 넘었는지 확인해보고 5년이 넘었으면 축하 메시지를 출력해본다.

```
DELIMITER $$
CREATE PROCEDURE IFPROC3()
BEGIN
    DECLARE DEBUTDATE DATE; -- 데뷔일자
    DECLARE CURDATE DATE;   -- 현재일자
    DECLARE DAYS INT;       -- 활동 일수

    SELECT DEBUT_DATE INTO DEBUTDATE
        FROM MARKET_DB.MEMBER
        WHERE MEM_ID = 'APN';

    SET CURDATE = CURRENT_DATE();             -- 현재 날짜
    SET DAYS = DATEDIFF(CURDATE,DEBUTDATE);   -- 날짜의 차이(일 단위)

    IF (DAYS/365) >=5 THEN
        SELECT CONCAT('데뷔한지',DAYS,'일이나 지났습니다 축하합니다') AS MASSAGE;
    ELSE
        SELECT '데뷔한지'+DAYS+'일밖에 안됐네여' AS MASSAGE ;
    END IF;
END $$
DELIMITER ;
```

💬 **날짜 관련 함수**

- **CURRENT_DATE( )** : 오늘 날짜를 알려줌
- **CURRENT_TIMESTAMP( )** : 오늘 날짜 및 시간을 함께 알려줌
- **DATEDIFF(날짜1, 날짜2)** : 날짜2부터 날짜1까지 일수로 몇일인지 알려줌

CASE 문

여러 가지 조건 중에 선택해야 하는 경우도 있다. 이럴 때 **CASE** 문을 사용해서 조건을 설정할 수 있다. IF 문은 참 아니면 거짓 두 가지만 있기 때문에 2중 분기라는 용어를 사용한다. **CASE**문은 2가지 이상의 여러 가지 경우일 때 처리가 가능하므로 '다중 분기' 라고 부른다. CASE문의 형식을 살펴보자.

```
CASE
    WHEN 조건1 THEN
        SQL 문장1
    WHEN 조건2 THEN
        SQL 문장2
    WHEN 조건3 THEN
        SQL 문장3
        ..............
    ELSE
        SQL 문장4
END CASE;
```

```
DELIMITER $$
CREATE PROCEDURE CASEPROC()
BEGIN
    DECLARE POINT INT;
    DECLARE CREDIT CHAR(1);
    SET POINT = 88;

    CASE
    WHEN POINT >=90 THEN
        SET CREDIT='A';
    WHEN POINT >=80 THEN
        SET CREDIT='B';
    WHEN POINT >=70 THEN
        SET CREDIT='C';
    WHEN POINT >=60 THEN
        SET CREDIT='D';
    ELSE
        SET CREDIT='F';
    END CASE;
    SELECT CONCAT('취득점수==>',POINT) AS '점수',
           CONCAT('학점==>',CREDIT) AS '학점';
END $$
DELIMITER $$
```

파이썬이나 R 과 같이 점수를 함수의 파라미터로 넣을 수도 있다. 다음 코드를 보면 된다.

```
DELIMITER $$
CREATE PROCEDURE CASEPROC(IN POINT INT)
BEGIN
    DECLARE CREDIT CHAR(1);

    CASE
        WHEN POINT >= 90 THEN
            SET CREDIT = 'A';
        WHEN POINT >= 80 THEN
            SET CREDIT = 'B';
        WHEN POINT >= 70 THEN
            SET CREDIT = 'C';
        WHEN POINT >= 60 THEN
            SET CREDIT = 'D';
        ELSE
            SET CREDIT = 'F';
    END CASE;

    SELECT CONCAT('취득점수 ==>', POINT) AS '점수',
           CONCAT('학점 ==>', CREDIT) AS '학점';
END $$
DELIMITER ;
```

CASE 문 활용

인터넷 마켓 데이터베이스에서 회원들의 총 구매액을 계산해서 회원의 등급을 다음과 같이 4단계로 나눈다.

총 구매액	회원 등급
1500 이상	최우수 고객
1000~1499	우수 고객
1~999	일반 고객
0	유령 고객

실행결과는 MEM_ID,MEM_NAME, 총구매액, 회원 등급이 나오도록 한번 만들어본다.

```sql
SELECT M.MEM_ID, M.MEM_NAME, SUM(PRICE*AMOUNT) '총구매액',
    CASE
        WHEN (SUM(PRICE*AMOUNT)>=1500) THEN '최우수고객'
        WHEN (SUM(PRICE*AMOUNT)>=1000) THEN '우수고객'
        WHEN (SUM(PRICE*AMOUNT)>= 1  ) THEN '일반고객'
        ELSE '유령고객'
    END '회원등급'
FROM BUY B
    RIGHT OUTER JOIN MEMBER M
    ON B.MEM_ID = M.MEM_ID
GROUP BY M.MEM_ID
ORDER BY SUM(PRICE*AMOUNT) DESC;
```

유령 고객도 고려해야하므로 OUTER JOIN을 사용해야 올바른 방식이다.

MEM_ID	MEM_NAME	총구매액	회원등급
MMU	마마무	1950	최우수고객
BLK	블랙핑크	1210	우수고객
APN	에이핑크	295	일반고객
GRL	소녀시대	75	일반고객
ITZ	잇지	NULL	유령고객
OMY	오마이걸	NULL	유령고객
RED	레드벨벳	NULL	유령고객
SPC	우주소녀	NULL	유령고객
TWC	트와이스	NULL	유령고객
WMN	여자친구	NULL	유령고객

WHILE 문

WHILE 문은 조건식이 참인 동안에 ‘SQL문장들’을 계속 반복한다. WHILE 문의 기본 형식은 다음과 같다.

```
WHILE <조건식> DO
    SQL 문장들
END WHILE;
```

먼저 간단히 1에서 100까지의 값들을 모두 더하는(4의 배수 제외) 간단한 기능을 WHILE 문으로 구현해본다.

```
DELIMITER $$
CREATE PROCEDURE WHILEPROC1()
BEGIN
    DECLARE I INT; -- 1에서 100까지 증가할 변수
    DECLARE SUMMATION INT; -- 누적 합 변수
    SET I = 1;
    SET SUMMATION = 0;

    MYWHILE:          -- 레이블 지정
    WHILE(I <=100) DO
        IF (I%4 = 0) THEN
            SET I = I +1;
            ITERATE MYWHILE; -- 지정한 레이블 문으로 가서 ITERATION 수행
        END IF;
        -- I가 4의 배수라면 I를 1 증가시키고 ITERATE를 만나 ITERATION
        SET SUMMATION = SUMMATION + I ;  -- I가 4의 배수가 아니면 SUMMATON에 누적
        IF (SUMMATION > 1000) THEN
            LEAVE MYWHILE; -- WHILE문 종료
        END IF;
        -- SUMMATION이 1000을 초과하면 LEAVE를 만나 LOOP 탈출
        SET I = I+1;
    END WHILE;

    SELECT SUMMATION;
END $$
DELIMITER ;
```

ITERATE는 반복문을 계속 진행하고, LEAVE는 반복문을 빠져 나가게 해준다.

동적 SQL

SQL 문은 내용이 고정되어 있는 경우가 대부분이다. 하지만 상황에 따라 내용 변경이 필요할 때 동적 SQL을 사용하면 변경되는 내용을 실시간으로 적용시켜 사용할 수 있다.

PREPARE & EXECUTE

이번 장의 앞에서 잠깐 살펴본 PREPARE와 EXECUTE 문을 다시 살펴본다. **PREPARE**는 SQL문을 실행하지는 않고 미리 준비만 하고, **EXECUTE**는 준비한 SQL문을 실행한다. 그리고 실행 후에는 **DEALLOCATE PREPARE**로 문장을 헤제해주는 것이 바람직하다.

```
USE MARKET_DB;
PREPARE MYQUERY FROM 'SELECT * FROM MEMBER WHERE MEM_ID ="BLK"';
EXECUTE MYQUERY;
DEALLOCATE PREPARE MYQUERY;
```

이렇게 미리 SQL을 준비한 후에 나중에 실행하는 것을 동적SQL이라고 부른다. 이 동적 SQL은 종종 유용하게 사용될 수 있으니 기억하길 바란다.

동적 SQL의 활용

PREPARE 문에서는 **?** 로 향후에 입력될 값을 비워 놓고, **EXECUTE**에서 **USING**으로 **?** 에 값을 전달할 수 있다. 그러면 실시간으로 필요한 값들을 전달해서 동적으로 SQL이 실행된다. 예제를 보면서 더 이해해보자.

보안이 중요한 출입문에서는 출입한 내역을 테이블에 기록해 놓는다. 이때 출입증을 태그하는 순간의 날짜와 시간이 **INSERT** 문으로 만들어져서 입력되도록 해야 한다.

```
#DROP TABLE IF EXISTS GATE_TABLE;
CREATE TABLE GATE_TABLE (ID INT AUTO_INCREMENT PRIMARY KEY, ENTRY_TIME DATETIME);
## 출입용 테이블 생성

SET @CURDATE = current_timestamp();

PREPARE MYQUERY FROM 'INSERT INTO GATE_TABLE VALUES(NULL,?)'; -- ?F를 사용하여 ENTRY_TIME에 입력할 값을 비워놓음
EXECUTE MYQUERY USING @CURDATE;                               -- USING문으로 앞에서 준비한 @CURDATE 변수를 넣은 후에 실행
DEALLOCATE PREPARE MYQUERY;

SELECT * FROM GATE_TABLE
```