

<저장 프로그램 연습

-marketDB의 저장프로시저,트리거,저장함수만들기_20221612김서윤 >

1. 데이터베이스와 테이블 생성 및 기본 데이터 입력

1-1. :데이터베이스 설정 및 테이블 생성

market_db로 해당 데이터베이스를 사용하도록 설정하고 회원(member) 테이블과 구매(buy) 테이블을 생성한다.

```
1 • DROP DATABASE IF EXISTS market_db;
2 • CREATE DATABASE market_db;
3
4 • USE market_db;
5 • CREATE TABLE member -- 회원 테이블
6 • ( mem_id      CHAR(8) NOT NULL PRIMARY KEY, -- 사용자 아이디(PK)
7 •   mem_name    VARCHAR(10) NOT NULL, -- 이름
8 •   mem_number  INT NOT NULL, -- 인원수
9 •   addr        CHAR(2) NOT NULL, -- 지역(경기,서울,경남 식으로 2글자만입력)
10 •  phone1      CHAR(3), -- 연락처의 국번(02, 031, 055 등)
11 •  phone2      CHAR(8), -- 연락처의 나머지 전화번호(하이픈제외)
12 •  height      SMALLINT, -- 평균 키
13 •  debut_date  DATE -- 데뷔 일자
14 • );

15 • CREATE TABLE buy -- 구매 테이블
16 • ( num        INT AUTO_INCREMENT NOT NULL PRIMARY KEY, -- 순번(PK)
17 •   mem_id     CHAR(8) NOT NULL, -- 아이디(FK)
18 •   prod_name  CHAR(6) NOT NULL, -- 제품이름
19 •   group_name CHAR(4) , -- 분류
20 •   price      INT NOT NULL, -- 가격
21 •   amount     SMALLINT NOT NULL, -- 수량
22 •   FOREIGN KEY (mem_id) REFERENCES member(mem_id)
23 • );
```

회원 테이블 (member): 각 회원의 아이디, 이름, 인원수, 지역, 연락처, 키, 데뷔일 정보를 저장.

mem_id는 member 테이블의 기본 키

구매 테이블 (buy): 회원들이 구매한 제품과 가격, 수량 등의 정보를 저장하며, member 테이블의 mem_id와 외래 키 관계를 맺어 연결됨. 여기서 mem_id는 buy 테이블에서 외래 키로 사용되고 있음.

buy 테이블의 mem_id는 member 테이블의 mem_id를 참조하여, buy 테이블의 각 구매 기록이 어떤 멤버에 속하는지를 알 수 있게 해준다.

이렇게 외래 키를 통해 두 테이블 간의 관계를 형성하여 데이터의 무결성과 일관성을 유지할 수 있다.

구조: CREATE TABLE 문을 사용하여 테이블을 생성할 때, 테이블의 이름, 각 컬럼의 데이터 타입 및 제약 조건을 정의

1-2. :데이터 삽입

```
25 • INSERT INTO member VALUES('TWC', '트와이스', 9, '서울', '02', '11111111', 167, '2015.10.19');
26 • INSERT INTO member VALUES('BLK', '블랙핑크', 4, '경남', '055', '22222222', 163, '2016.08.08');
27 • INSERT INTO member VALUES('WMN', '여자친구', 6, '경기', '031', '33333333', 166, '2015.01.15');
28 • INSERT INTO member VALUES('OMY', '오마이걸', 7, '서울', NULL, NULL, 160, '2015.04.21');
29 • INSERT INTO member VALUES('GRL', '소녀시대', 8, '서울', '02', '44444444', 168, '2007.08.02');
30 • INSERT INTO member VALUES('ITZ', '잇지', 5, '경남', NULL, NULL, 167, '2019.02.12');
31 • INSERT INTO member VALUES('RED', '레드벨벳', 4, '경북', '054', '55555555', 161, '2014.08.01');
32 • INSERT INTO member VALUES('APN', '에이핑크', 6, '경기', '031', '77777777', 164, '2011.02.10');
33 • INSERT INTO member VALUES('SPC', '우주소녀', 13, '서울', '02', '88888888', 162, '2016.02.25');
34 • INSERT INTO member VALUES('MMU', '마마무', 4, '전남', '061', '99999999', 165, '2014.06.19');
35 •
36 • INSERT INTO buy VALUES(NULL, 'BLK', '지갑', NULL, 30, 2);
37 • INSERT INTO buy VALUES(NULL, 'BLK', '맥북프로', '디지털', 1000, 1);
38 • INSERT INTO buy VALUES(NULL, 'APN', '아이폰', '디지털', 200, 1);
39 • INSERT INTO buy VALUES(NULL, 'MMU', '아이폰', '디지털', 200, 5);
40 • INSERT INTO buy VALUES(NULL, 'BLK', '청바지', '패션', 50, 3);
41 • INSERT INTO buy VALUES(NULL, 'MMU', '에어팟', '디지털', 80, 10);
42 • INSERT INTO buy VALUES(NULL, 'GRL', '홍콩SQL', '서적', 15, 5);
43 • INSERT INTO buy VALUES(NULL, 'APN', '홍콩SQL', '서적', 15, 2);
44 • INSERT INTO buy VALUES(NULL, 'APN', '청바지', '패션', 50, 1);
45 • INSERT INTO buy VALUES(NULL, 'MMU', '지갑', NULL, 30, 1);
46 • INSERT INTO buy VALUES(NULL, 'APN', '홍콩SQL', '서적', 15, 1);
47 • INSERT INTO buy VALUES(NULL, 'MMU', '지갑', NULL, 30, 4);
```

1-3. :데이터 조회

```
49 • SELECT * FROM member;
50 • SELECT * FROM buy;
```

mem_id	mem_name	mem_number	addr	phone1	phone2	height	debut_date
APN	에이핑크	6	경기	031	77777777	164	2011-02-10
BLK	블랙핑크	4	경남	055	22222222	163	2016-08-08
GRL	소녀시대	8	서울	02	44444444	168	2007-08-02
ITZ	잇지	5	경남	NULL	NULL	167	2019-02-12
MMU	마마무	4	전남	061	99999999	165	2014-06-19
OMY	오마이걸	7	서울	NULL	NULL	160	2015-04-21
RED	레드벨벳	4	경북	054	55555555	161	2014-08-01
SPC	우주소녀	13	서울	02	88888888	162	2016-02-25
TWC	트와이스	9	서울	02	11111111	167	2015-10-19
WMN	여자친구	6	경기	031	33333333	166	2015-01-15
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

num	mem_id	prod_name	group_name	price	amount
1	BLK	지갑	NULL	30	2
2	BLK	악북프로	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	홍콩SQL	서적	15	5
8	APN	홍콩SQL	서적	15	2
9	APN	청바지	패션	50	1
10	MMU	지갑	NULL	30	1
11	APN	홍콩SQL	서적	15	1
12	MMU	지갑	NULL	30	4
NULL	NULL	NULL	NULL	NULL	NULL

2.저장 프로시저 생성 및 호출 저장 프로시저 생성 및 호출:

데이터를 처리하기 위한 여러 개의 저장 프로시저를 생성하는 과정이다. 다음의 각 프로시저는 다양한 입력을 받아 특정한 작업을 수행하고 결과를 출력한다.

2-1. : user_proc1: 그룹명으로 멤버 정보 출력-입력 받은 그룹명()을 기준으로 그룹의 이름, 멤버 수, 데뷔 날짜를 출력한다.

```

52  #<1>저장 프로시저 만들기
53  -- 1번 user_proc1('에이핑크'):그룹명을 입력으로 받아 그룹 이름, 멤버수와 데뷔 날짜 출력
54  • use market_db;
55  • DROP procedure if exists user_proc1;
56  delimiter $$
57  • create procedure user_proc1(in username VARCHAR(10))
58  • begin
59      select mem_name, mem_number, debut_date from member where mem_name = username;
60  • end $$
61  delimiter ;
62
63  • call user_proc1('에이핑크');
64

```

mem_name	mem_number	debut_date
에이핑크	6	2011-02-10

-begin ... end: 이 부분은 프로시저의 본체를 나타낸다. 여기서 SQL 문을 작성하여 프로시저가 수행할 작업

을 정의한다. select mem_name, mem_number, debut_date from member where mem_name = username;

이 SQL 문은 member 테이블에서 mem_name, mem_number, debut_date 컬럼의 값을 선택하여 가져온다.

-where mem_name = username: 입력받은 username 값과 member 테이블의 mem_name 컬럼을 비교하여 일치하는 레코드만 조회한다.

결과값: 그룹명이 '에이핑크'일 경우, 그들의 이름, 멤버 수(6명), 데뷔 날짜(2011-02-10)가 출력된다.

2-2. :user_proc2: 멤버 수와 평균 키 기준으로 그룹 정보 출력-입력받은 숫자보다 멤버 수가 많고, 입력받은 키보다 평균 키가 큰 그룹의 정보를 출력한다.

두 개의 입력 파라미터를 받는다.

SELECT 문은 member 테이블에서 모든 컬럼(*)을 선택하여 가져온다.

```
65 -- 2번 user_proc2(6, 165)
66 -- : 입력받은 숫자보다 멤버수가 많고, 입력받은 키보다 멤버 평균키가 큰 그룹의 모든 정보 출력
67 • DROP procedure if exists user_proc2;
68 delimiter $$
69 • create procedure user_proc2(in usernumber INT,in userheight SMALLINT)
70 begin
71     select * from member where mem_number> usernumber and height > userheight ;
72 end $$
73 delimiter ;
74
75 • call user_proc2(6, 165);
76
```

Result Grid Filter Rows: Export: Wrap Cell Content:								
	mem_id	mem_name	mem_number	addr	phone1	phone2	height	debut_date
▶	GRL	소녀시대	8	서울	02	44444444	168	2007-08-02
	TWC	트와이스	9	서울	02	11111111	167	2015-10-19

결과값: 예를 들어, CALL user_proc2(6, 165);를 호출하면, 멤버 수가 6명보다 많고 평균 키가 165cm보다 큰 그룹의 정보가 출력된다.

2.3. :message_proc: 그룹의 데뷔 년도에 따른 메시지 출력-입력받은 그룹의 데뷔 연도에 따라 메시지를 출력한다. 2015년 이후면 "신인가수네요. 화이팅하세요", 이전이면 "고참가수네요. 그동안 수고하셨습니다"라는 메시지를 출력한다.

입력 파라미터 memname(그룹의 이름)을 받는다.

DECLARE debutyear INT : debutyear라는 변수를 선언한다. 이 변수는 그룹의 데뷔 연도를 저장할 용도로 사용된다.

select문은 member 테이블에서 debut_date의 연도를 추출하여 debutyear 변수에 저장한다.

```
77 -- 3번 message_proc('오마이걸')
78 -- : 입력받은 그룹이 데뷔연도가 2015년 이후이면 '신인가수네요. 화이팅하세요' 출력,
79 -- 2015년 이전이면 '고참가수네요. 그동안 수고하셨습니다' 출력
80 • DROP PROCEDURE IF EXISTS message_proc;
81 DELIMITER $$
82 • CREATE PROCEDURE message_proc(IN memname varchar(10))
83 BEGIN
84     declare debutyear int;
85     select year(debut_date) into debutyear from member
86         where mem_name = memname;
87     if (debutyear >= 2015) then
88         select '신인가수네요. 화이팅하세요';
89     else
90         select '고참가수네요. 그동안 수고하셨습니다';
91     end if;
92 END $$
93 delimiter ;
94 • call message_proc('오마이걸');
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	신인가수네요. 화이팅하 세요
▶	신인가수네요. 화이팅하세요

결과값: '오마이걸' 그룹을 입력하면, 그들의 데뷔 연도가 2015년 이후이므로 "신인가수네요. 화이팅하세요"라는 메시지가 출력된다.

2.4 : avg_member: 멤버들의 평균 수 출력-모든 그룹의 멤버 수의 평균을 계산하여 출력한다.

```

97  -- 4번 avg_member( ) : 멤버들의 평균 수 출력
98  • DROP procedure if exists avg_member;
99  delimiter $$
100 • create procedure avg_member()
101 BEGIN
102     DECLARE membernumber FLOAT;-- 멤버 수
103     DECLARE membercount INT DEFAULT 0;-- 멤버 수를 저장할 변수(읽을 행의 수 카운트)
104     DECLARE totalmember INT DEFAULT 0; -- 총 멤버 수
105
106     DECLARE endOfRow BOOLEAN DEFAULT FALSE;
107     -- 커서 선언
108     DECLARE member_Cursor CURSOR FOR
109         SELECT mem_number FROM member;
110     -- 반복조건선언(더 이상 읽을 행이 없을 때 실행할 내용 설정)
111     DECLARE CONTINUE HANDLER
112         FOR NOT FOUND SET endOfRow = TRUE;
113
114     -- 커서 열기
115     OPEN member_Cursor;
116     -- 커서 반복문, 커서에서 데이터 가져오고 데이터 처리
117     cursor_loop: LOOP
118         FETCH member_Cursor INTO membernumber;
119
120         IF endOfRow THEN
121             LEAVE cursor_loop;
122         END IF;
123
124         SET membercount = membercount + 1;
125         SET totalmember = totalmember + membernumber;
126     END LOOP cursor_loop;
127
128     SELECT CONCAT('멤버 수의 평균 ==> ', (totalmember/membercount));
129     -- 커서 닫기
130     CLOSE member_Cursor;
131 END $$
132 DELIMITER ;
133
134 • CALL avg_member();

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	CONCAT('멤버 수의 평균 ==> ', (totalmember/membercount))		
▶	멤버 수의 평균 ==> 6.6000		

1) 변수 선언

DECLARE membernumber FLOAT;; 각 멤버의 수를 저장

DECLARE membercount INT DEFAULT 0;; 커서가 읽어들이는 멤버 수를 카운트

DECLARE totalmember INT DEFAULT 0;; 멤버 수의 총합을 저장

2) 커서 선언

3) CONTINUE HANDLER: 커서에서 더 이상 읽을 행이 없을 때 실행될 코드를 정의

4)커서 열기: OPEN member_Cursor;

5)커서 반복문

- LOOP: 커서가 데이터를 반복적으로 읽을 수 있는 루프를 시작합니다.
- FETCH member_Cursor INTO membernumber;; 커서에서 다음 행의 mem_number 값을 읽어 membernumber 변수에 저장합니다.
- IF endOfRow THEN LEAVE cursor_loop; END IF;; 만약 더 이상 읽을 데이터가 없다면 루프를 종료합니다.
- SET membercount = membercount + 1;; 읽은 멤버 수를 카운트합니다.
- SET totalmember = totalmember + membernumber;; 현재 멤버의 수를 총 멤버 수에 추가합니다.

6)결과 출력

7)커서 닫기 : CLOSE member_Cursor;

결과값: 계산된 평균 멤버 수가 출력된다

3.트리거 생성. 트리거란 insert update.delete문이 작동할 때 자동으로 실행되는 프로그램 기능이다.

3-1 : singer 테이블 생성-member 테이블의 데이터를 복사

backup_singer 테이블 생성: 수정 또는 삭제된 회원의 정보를 백업하기 위해.

```
137 #<2>트리거 만들기
138 -- singer테이블 생성
139 • DROP TABLE IF EXISTS singer;
140 • CREATE TABLE singer (SELECT mem_id, mem_name, mem_number, addr FROM member);
141 -- backup_singer테이블 생성
142 • DROP TABLE IF EXISTS backup_singer;
143 • CREATE TABLE backup_singer
144 ( mem_id CHAR(8) NOT NULL ,
145   mem_name VARCHAR(10) NOT NULL,
146   mem_number INT NOT NULL,
147   addr CHAR(2) NOT NULL,
148   modType CHAR(2), -- 변경된 타입. '수정' 또는 '삭제'
149   modDate DATE, -- 변경된 날짜
150   modUser VARCHAR(30) -- 변경한 사용자
151 );
```

3.2 : Update 트리거 - singer 테이블의 데이터가 업데이트될 때마다

백업 테이블에 저장함 . 이렇게 함으로써, 나중에 업데이트된 데이터를 추적하고, 필요할 경우 이전 상태로 복원할 수 있도록 한다.

Delete 트리거 - singer 테이블의 데이터가 삭제될 때마다 백업 테이블에 저장함

```
153 -- 1)Update 트리거
154 • DROP TRIGGER IF EXISTS singer_update_Trg;
155 DELIMITER $$
156 • CREATE TRIGGER singer_update_Trg
157     AFTER UPDATE
158     ON singer
159     FOR each row
160 BEGIN
161     INSERT INTO backup_singer VALUES( OLD.mem_id, OLD.mem_name, OLD.mem_number,
162         OLD.addr, '수정', curdate(), current_user() );
163 END $$
164 DELIMITER ;
165 • -- 2) delete 트리거
166 DROP TRIGGER IF EXISTS singer_delete_Trg;
167 DELIMITER $$
168 • CREATE TRIGGER singer_delete_Trg
169     AFTER delete
170     ON singer
171     FOR each row
172 BEGIN
173     INSERT INTO backup_singer VALUES( OLD.mem_id, OLD.mem_name, OLD.mem_number,
174         OLD.addr, '삭제', curdate(), current_user() );
175 END $$
176 DELIMITER ;
```

AFTER UPDATE(DELETE): 이 트리거는 singer 테이블의 데이터가 업데이트된(삭제된) 후에 실행. 즉, 어떤 행이 업데이트된(삭제된) 후에 이 트리거가 활성화.

ON singer: 이 트리거가 적용될 테이블을 지정->singer 테이블

FOR EACH ROW: 이 트리거는 singer 테이블의 각 행이 업데이트될 때마다 실행.즉, 여러 행이 동시에 업데이트되면 각각에 대해 트리거가 실행.

BEGIN ... END: 트리거의 실행 블록을 정의. 이 블록 안에 트리거가 수행할 SQL 문이 포함.

INSERT INTO backup_singer:

이 부분은 backup_singer 테이블에 새로운 레코드를 삽입하는 SQL 문이다..

-OLD: 업데이트되기 전의 값을 참조. 즉, OLD.mem_id는 업데이트되기 전의 singer 테이블의 mem_id 값을 의미

- '수정' '삭제': 업데이트된 행에 대한 작업 유형

- CURDATE(): 현재 날짜. (업데이트가 발생한 날짜)

● - CURRENT_USER(): 현재 트리를 실행하는 사용자의 이름.

```
178 • -- Update실행 후 backup_singer테이블 확인
179   SET SQL_SAFE_UPDATES = 0; -- 안전 모드 해제
180 • Update singer set addr = '영국' where mem_id = 'BLK';
181 • SET SQL_SAFE_UPDATES = 1; -- 안전 모드 다시 활성화
182
183 • Select * from backup_singer;
```

Result Grid Filter Rows: Export: Wrap Cell Content:							
	mem_id	mem_name	mem_number	addr	modType	modDate	modUser
▶	BLK	블랙핑크	4	경남	수정	2024-10-13	root@localhost

결과값: singer 테이블의 주소가 업데이트된 경우, 변경된 데이터의 이전 값이 backup_singer 테이블에 삽입된다.

```
186 -- Delete실행 후 backup_singer테이블 확인
187 • SET SQL_SAFE_UPDATES = 0;
188 • Delete from singer where mem_number >= 7;
189 • SET SQL_SAFE_UPDATES = 1;
190
191 • Select * from backup_singer;
192
```

Result Grid Filter Rows: Export: Wrap Cell Content:							
	mem_id	mem_name	mem_number	addr	modType	modDate	modUser
▶	BLK	블랙핑크	4	경남	수정	2024-10-13	root@localhost
	GRL	소녀시대	8	서울	삭제	2024-10-13	root@localhost
	OMY	오마이걸	7	서울	삭제	2024-10-13	root@localhost
	SPC	우주소녀	13	서울	삭제	2024-10-13	root@localhost
	TWC	트와이스	9	서울	삭제	2024-10-13	root@localhost

결과값: singer 테이블에서 멤버 수가 7 이상인 그룹이 삭제된 경우, 해당 데이터의 이전 값이 backup_singer 테이블에 삽입된다.

4. 저장 함수 생성

calcYearFunc: 데뷔 연도를 입력받아 활동 햇수를 계산하는 함수이다. 저장 프로시저와 구조는 비슷하다. Return문을 사용하며 호출시 call이 아닌 select를 쓴다.

반환 시 주어진 연도(debutyear)를 기반으로 현재 연도와 차이를 계산하여 활동한 연수를 반환한다.

```

194  #<3>저장 함수 만들기->데뷔 연도를 입력하면 활동 햇수를 출력하는 함수
195 • SET GLOBAL log_bin_trust_function_creators = 1;
196
197 • DROP FUNCTION IF EXISTS calcYearFunc;
198 DELIMITER $$
199 • CREATE FUNCTION calcYearFunc(debutyear INT)
200     RETURNS INT
201     BEGIN
202         RETURN YEAR(CURDATE()) - debutyear ;
203     END $$
204
205 DELIMITER ;
206
207 • SELECT calcYearFunc(2010) AS '활동햇수'

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	활동 햇수			
▶	14			

-CREATE FUNCTION: 새로운 함수를 생성하는 명령어

-debutyear INT: 함수에 입력될 데뷔 연도 매개변수를 정의. 여기서는 정수형(INT)의 debutyear라는 매개변수를 받는다.

-RETURNS INT: 이 함수가 정수형 값을 반환한다는 것을 지정.->계산된 활동한 연수를 정수형으로 반환.

-RETURN YEAR(CURDATE()) - debutyear : 현재 연도에서 주어진 debutyear를 빼서 활동한 연수를 계산. 예를 들어, 만약 현재 연도가 2024년이고, debutyear가 2010년이라면, 계산 결과는 $2024 - 2010 = 14$ 가 된다.

결과값: 예를 들어, 2010년을 입력하면 현재 연도(2024년)와의 차이인 14가 출력된다.