6. SOFT DELETE

#0.강의/2.데이터베이스로드맵/4.설계2

SOFT DELETE가 필요한 이유

실무에서 데이터를 삭제할 때 가장 먼저 고민해야 할 것이 있다. "정말로 데이터를 지워도 될까?"라는 질문이다. 이번 수업에서는 데이터 삭제 시 발생할 수 있는 문제점을 살펴보고, 왜 SOFT DELETE라는 기법이 필요한지 알아보겠다.

문제 상황 - 데이터를 실수로 삭제했다

우리 쇼핑몰에서 회원 관리를 하고 있다고 가정해보자. 먼저 간단한 회원 테이블을 만들어보겠다.

```
DROP TABLE IF EXISTS member;

CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO member (username, email, phone) VALUES
('kim', 'kim@example.com', '010-1111-1111'),
('lee', 'lee@example.com', '010-2222-2222'),
('park', 'park@example.com', '010-3333-3333'),
```

```
('choi', 'choi@example.com', '010-4444-4444');
```

```
SELECT * FROM member;
```

[실행 결과]

member_id	username	email	phone	created_at
1	kim	kim@example.com	010-1111-1111	...
2	lee	lee@example.com	010-2222-2222	...
3	park	park@example.com	010-3333-3333	...
4	choi	choi@example.com	010-4444-4444	...

이제 `park` 회원이 탈퇴를 요청해서 삭제했다고 가정하자.

```
DELETE FROM member WHERE member_id = 3;
```

```
SELECT * FROM member;
```

[실행 결과]

member_id	username	email	phone	created_at
1	kim	kim@example.com	010-1111-1111	...
2	lee	lee@example.com	010-2222-2222	...
4	choi	choi@example.com	010-4444-4444	...

데이터가 깔끔하게 삭제되었다. 그런데 문제가 발생했다.

문제 1 - 복구가 어렵다.

삭제 후 1시간이 지났는데, 고객센터에서 연락이 왔다. `park` 회원이 실수로 탈퇴 버튼을 눌렀다며 복구를 요청한다.
어떻게 해야 할까?

```
-- park 회원을 다시 찾아보자
SELECT * FROM member WHERE username = 'park';
```

[실행 결과]

member_id	username	email	phone	created_at

아무것도 없다. `DELETE`로 삭제한 데이터는 영구적으로 사라진다. 데이터베이스에서 물리적으로 제거되었기 때문에 일반적인 방법으로는 복구할 수 없다.

물론 데이터베이스 백업이나 데이터베이스 로그를 통해서 복구할 수 있지만 복구에 시간이 오래 걸린다.
DBA에게 요청하고, 백업 파일을 찾고, 데이터를 추출하는 과정이 필요하다. 고객은 그동안 기다려야 한다.

문제 2 - 연관된 데이터가 함께 사라진다

더 심각한 문제가 있다. 회원이 주문한 데이터가 있다면 어떻게 될까?

```
DROP TABLE IF EXISTS payments; -- 이전에 학습한 테이블
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS member;

CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE orders (
```

```
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_name VARCHAR(100) NOT NULL,
    amount INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (member_id) REFERENCES member(member_id)
);
```

데이터를 입력하자.

```
INSERT INTO member (username, email) VALUES
('kim', 'kim@example.com'),
('lee', 'lee@example.com'),
('park', 'park@example.com');

INSERT INTO orders (member_id, product_name, amount) VALUES
(1, '노트북', 1500000),
(2, '키보드', 150000),
(3, '마우스', 50000),
(3, '모니터', 300000);
```

`park` 회원은 주문 내역이 2건 있다.

```
SELECT m.username, o.product_name, o.amount
FROM member m
JOIN orders o ON m.member_id = o.member_id
WHERE m.username = 'park';
```

[실행 결과]

username	product_name	amount
park	마우스	50000
park	모니터	300000

이제 `park` 회원을 삭제하려고 하면 어떻게 될까?

```sql
SET SQL_SAFE_UPDATES = 0; -- 안전 업데이트 모드 끄기

DELETE FROM member WHERE username = 'park';

SET SQL_SAFE_UPDATES = 1; -- 안전 업데이트 모드 활성화
```

[실행 결과]

```
Error Code: 1451. Cannot delete or update a parent row: a foreign key
constraint fails (`my_shop4`.`orders`, CONSTRAINT `orders_ibfk_1` FOREIGN KEY
(`member_id`) REFERENCES `member` (`member_id`))
```

외래 키 제약 조건 때문에 삭제가 실패한다. 주문 테이블에서 `park` 회원을 참조하고 있기 때문이다.

이 문제를 해결하는 방법은 몇 가지가 있다.

방법 1 - 연관된 데이터를 먼저 삭제

```sql
-- 주문 데이터 먼저 삭제
DELETE FROM orders WHERE member_id = 3;
-- 그 다음 회원 삭제
DELETE FROM member WHERE member_id = 3;
```

이 방법은 주문 데이터까지 모두 사라진다. 매출 분석, 통계, 감사(audit) 등을 위해 주문 데이터가 필요하다면 큰 문제가 된다.

방법 2 - CASCADE 옵션 사용

테이블 생성 시 `ON DELETE CASCADE` 옵션을 사용하면 부모 데이터 삭제 시 자식 데이터도 자동으로 삭제된다.

```sql
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS member;

CREATE TABLE member (
```

```
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_name VARCHAR(100) NOT NULL,
    amount INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (member_id) REFERENCES member(member_id) ON DELETE CASCADE
);
```

- 데이터를 입력하고 삭제해보면 연관된 데이터가 함께 제거된다.

하지만 이 방법도 결국 주문 데이터가 사라지는 것은 마찬가지다. 오히려 자동으로 삭제되기 때문에 실수로 대량의 데이터가 삭제될 위험이 있다.

방법 3 - SET NULL 옵션 사용

```
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS member;

CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT,  -- NULL 허용
    product_name VARCHAR(100) NOT NULL,
    amount INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (member_id) REFERENCES member(member_id) ON DELETE SET NULL
);
```

- 데이터를 입력하고 삭제해보면 `orders` 테이블의 `member_id`에 NULL이 입력된다.

이 방법은 회원이 삭제되면 주문의 `member_id`가 `NULL`로 변경된다. 주문 데이터는 유지되지만, 누가 주문했는지 알 수 없게 된다. "이 주문을 누가 했지?"라는 질문에 답할 수 없다.

문제 3 - 법적, 비즈니스적 요구사항

실무에서는 데이터를 함부로 삭제하면 안 되는 경우가 많다.

첫째, 법적 요구사항이 있다. 전자상거래법에 따르면 거래 기록은 5년간 보관해야 한다. 세금 관련 자료도 일정 기간 보관 의무가 있다. 회원이 탈퇴했다고 관련 거래 기록까지 삭제하면 법적 문제가 생길 수 있다.

둘째, 비즈니스 분석이 필요하다. 탈퇴한 회원도 포함해서 매출 분석, 고객 행동 분석을 해야 할 때가 있다. 데이터가 삭제되면 정확한 분석이 불가능하다.

셋째, 감사(Audit) 요구사항이 있다. 금융, 의료 등의 분야에서는 누가 언제 무엇을 했는지 추적할 수 있어야 한다. 데이터가 물리적으로 삭제되면 추적이 불가능하다.

문제 4 - 삭제 시점을 알 수 없다

`DELETE`로 삭제하면 언제 삭제되었는지 알 수 없다. 고객센터에서 "이 회원이 언제 탈퇴했나요?"라고 물어보면 대답할 수 없다.

```
-- park 회원이 언제 탈퇴했는지 알 수 없다
SELECT * FROM member WHERE username = 'park';
```

[실행 결과]

member_id	username	email	created_at

데이터 자체가 없으니 탈퇴 시점도 알 수 없다.

정리 - HARD DELETE의 문제점

지금까지 살펴본 것처럼 `DELETE` 문으로 데이터를 물리적으로 삭제하는 것을 **HARD DELETE**라고 한다. HARD DELETE는 다음과 같은 문제가 있다.

문제	설명
복구 불가	삭제된 데이터는 일반적인 방법으로 복구할 수 없다
연관 데이터 처리	외래 키로 연결된 데이터 처리가 복잡하다
법적 요구사항	일정 기간 데이터 보관 의무를 충족하기 어렵다
분석 불가	삭제된 데이터는 통계나 분석에 포함할 수 없다
감사 추적 불가	언제, 누가 삭제했는지 추적하기 어렵다
삭제 시점 불명	데이터가 언제 삭제되었는지 알 수 없다

이런 문제들을 해결하기 위해 **SOFT DELETE**라는 기법이 등장했다. 다음 수업에서는 SOFT DELETE가 무엇이고 어떻게 구현하는지 알아보겠다.

SOFT DELETE - is_deleted 컬럼 방식

앞서 HARD DELETE의 문제점을 살펴보았다. 이번 수업에서는 이 문제를 해결하는 **SOFT DELETE**에 대해 알아보겠다. 가장 기본적인 방식인 `is_deleted` 컬럼을 사용하는 방법부터 시작하겠다.

SOFT DELETE란?

SOFT DELETE는 데이터를 실제로 삭제하지 않고, 삭제되었다는 표시만 해두는 방식이다. 데이터베이스에서 물리적으로 제거하는 대신, "이 데이터는 삭제된 것으로 간주해라"라는 표시를 남긴다. 보통 "논리적 삭제"라고 한다.

가장 간단한 방법은 `is_deleted`라는 컬럼을 추가하는 것이다.

```
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS member;
```

```sql
CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    is_deleted BOOLEAN DEFAULT FALSE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO member (username, email, phone) VALUES
('kim', 'kim@example.com', '010-1111-1111'),
('lee', 'lee@example.com', '010-2222-2222'),
('park', 'park@example.com', '010-3333-3333'),
('choi', 'choi@example.com', '010-4444-4444');
```

```sql
SELECT * FROM member;
```

[실행 결과]

member_id	username	email	phone	is_deleted	created_at
1	kim	kim@example.com	010-1111-1111	0	2026-01-15 10:00:00
2	lee	lee@example.com	010-2222-2222	0	2026-01-15 10:00:00
3	park	park@example.com	010-3333-3333	0	2026-01-15 10:00:00
4	choi	choi@example.com	010-4444-4444	0	2026-01-15 10:00:00

모든 회원의 `is_deleted`가 `0`이다. 아직 아무도 삭제되지 않았다는 뜻이다.

SOFT DELETE 실행하기

이제 `park` 회원이 탈퇴를 요청했다. HARD DELETE와 달리, `DELETE` 문을 사용하지 않고 `UPDATE` 문을 사용한다.

```
-- SOFT DELETE: 실제로 삭제하지 않고 is_deleted를 TRUE(1)로 변경
UPDATE member
SET is_deleted = TRUE
WHERE member_id = 3;
```

```
SELECT * FROM member;
```

[실행 결과]

member_id	username	email	phone	is_deleted	created_at
1	kim	kim@example.com	010-1111-1111	0	2026-01-15 10:00:00
2	lee	lee@example.com	010-2222-2222	0	2026-01-15 10:00:00
3	park	park@example.com	010-3333-3333	1	2026-01-15 10:00:00
4	choi	choi@example.com	010-4444-4444	0	2026-01-15 10:00:00

`park` 회원의 `is_deleted`가 `TRUE(1)`로 변경되었다. 데이터는 그대로 있지만, "삭제된 것으로 간주"하라는 표시가 되었다.

삭제되지 않은 데이터만 조회하기

일반적인 조회에서는 삭제된 데이터를 보여주면 안 된다. 조회할 때 `is_deleted = FALSE` 조건을 추가한다.

```sql
-- 활성 회원만 조회 (삭제되지 않은 회원)
SELECT * FROM member WHERE is_deleted = FALSE;
```

[실행 결과]

member_id	username	email	phone	is_deleted	created_at
1	kim	kim@example.com	010-1111-1111	0	2026-01-15 10:00:00
2	lee	lee@example.com	010-2222-2222	0	2026-01-15 10:00:00
4	choi	choi@example.com	010-4444-4444	0	2026-01-15 10:00:00

사용자 입장에서는 `park` 회원이 삭제된 것처럼 보인다. 하지만 데이터는 여전히 데이터베이스에 존재한다.

삭제된 데이터 복구하기

이제 `park` 회원이 복구를 요청했다. SOFT DELETE의 가장 큰 장점이 여기서 드러난다.

```sql
-- 복구: is_deleted를 다시 FALSE로 변경
UPDATE member
SET is_deleted = FALSE
WHERE member_id = 3;
```

```sql
SELECT * FROM member WHERE is_deleted = FALSE;
```

[실행 결과]

member_id	username	email	phone	is_deleted	created_at
1	kim	kim@example.com	010-1111-1111	0	2026-01-15 10:00:00

2	lee	lee@example.com	010-2222-2222	0	2026-01-15 10:00:00
3	park	park@example.com	010-3333-3333	0	2026-01-15 10:00:00
4	choi	choi@example.com	010-4444-4444	0	2026-01-15 10:00:00

단 한 번의 UPDATE로 복구가 완료되었다. HARD DELETE였다면 백업에서 데이터를 찾아 복원하는 복잡한 과정이 필요했을 것이다.

연관된 데이터와 함께 사용하기

주문 테이블도 SOFT DELETE를 적용해보자.

```
DROP TABLE IF EXISTS orders;

CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_name VARCHAR(100) NOT NULL,
    amount INT NOT NULL,
    is_deleted BOOLEAN DEFAULT FALSE,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (member_id) REFERENCES member(member_id)
);

INSERT INTO orders (member_id, product_name, amount) VALUES
(1, '노트북', 1500000),
(2, '키보드', 150000),
(3, '마우스', 50000),
(3, '모니터', 300000),
(3, '헤드셋', 80000);
```

`park` 회원(member_id = 3)이 탈퇴하면서 주문 내역도 함께 SOFT DELETE 하고 싶다면 다음과 같이 한다.

```sql
-- 회원 SOFT DELETE
UPDATE member SET is_deleted = TRUE WHERE member_id = 3;

-- 해당 회원의 주문도 SOFT DELETE
UPDATE orders SET is_deleted = TRUE WHERE member_id = 3;
```

```sql
-- 삭제되지 않은 주문만 조회
SELECT * FROM orders WHERE is_deleted = FALSE;
```

[실행 결과]

order_id	member_id	product_name	amount	is_deleted	order_date
1	1	노트북	1500000	0	2026-01-15 10:00:00
2	2	키보드	150000	0	2026-01-15 10:00:00

`park` 회원(`member_id=3`)의 주문 3건이 모두 보이지 않는다. 하지만 데이터는 남아있어서 필요할 때 조회할 수 있다.

```sql
-- 관리자가 삭제된 주문 포함 전체 조회
SELECT * FROM orders;
```

[실행 결과]

order_id	member_id	product_name	amount	is_deleted	order_date
1	1	노트북	1500000	0	2026-01-15 10:00:00
2	2	키보드	150000	0	2026-01-15 10:00:00

3	3	마우스	50000	1	2026-01-15 10:00:00
4	3	모니터	300000	1	2026-01-15 10:00:00
5	3	헤드셋	80000	1	2026-01-15 10:00:00

통계와 분석에 활용하기

SOFT DELETE의 또 다른 장점은 삭제된 데이터도 분석에 활용할 수 있다는 것이다.

```
-- 전체 매출 (삭제된 주문 포함)
SELECT SUM(amount) AS total_revenue FROM orders;
```

[실행 결과]

total_revenue
2080000

```
-- 활성 주문의 매출만
SELECT SUM(amount) AS active_revenue FROM orders WHERE is_deleted = FALSE;
```

[실행 결과]

active_revenue
1650000

```
-- 탈퇴한 회원의 매출 기여도 분석
```

```sql
SELECT
    SUM(CASE WHEN is_deleted = FALSE THEN amount ELSE 0 END) AS
active_revenue,
    SUM(CASE WHEN is_deleted = TRUE THEN amount ELSE 0 END) AS
churned_revenue,
    SUM(amount) AS total_revenue
FROM orders;
```

[실행 결과]

active_revenue	churned_revenue	total_revenue
1650000	430000	2080000

이처럼 탈퇴한 회원의 데이터도 분석에 활용할 수 있다. HARD DELETE였다면 이런 분석이 불가능했을 것이다.

is_deleted 방식의 한계

`is_deleted` 컬럼 방식은 간단하고 이해하기 쉽다. 하지만 실무에서 사용하다 보면 몇 가지 한계에 부딪힌다.

한계 1 - 삭제 시점을 알 수 없다

고객센터에서 이런 질문이 들어왔다. "park 회원이 언제 탈퇴했나요?"

```sql
SELECT * FROM member WHERE member_id = 3;
```

[실행 결과]

member_id	username	email	phone	is_deleted	created_at
3	park	park@example.com	010-3333-3333	1	2026-01-15 10:00:00

`is_deleted = 1`(`TRUE`)이라는 것은 알 수 있지만, 언제 삭제되었는지는 알 수 없다. `created_at`은 회원 가입 시점이지 탈퇴 시점이 아니다.

한계 2 - 일정 기간 후 영구 삭제 정책을 적용하기 어렵다

개인정보보호법에 따르면 탈퇴한 회원의 정보는 일정 기간(예: 1년) 후 완전히 삭제해야 한다. 이 정책을 적용하려면 "언제 탈퇴했는지" 알아야 한다.

```
-- 1년 전에 탈퇴한 회원을 영구 삭제하고 싶다면?
-- 탈퇴 시점을 알 수 없어서 불가능!
DELETE FROM member
WHERE is_deleted = TRUE
AND ??? < DATE_SUB(NOW(), INTERVAL 1 YEAR);
```

삭제 시점이 없으니 이런 쿼리를 작성할 수 없다.

한계 3 - 복구 이력을 관리할 수 없다

회원이 탈퇴했다가 복구하고, 다시 탈퇴하는 경우가 있다. `is_deleted` 컬럼만으로는 이 이력을 추적할 수 없다.

```
-- park 회원의 탈퇴/복구 이력을 알고 싶다면?
-- 현재 상태만 알 수 있고, 이력은 알 수 없다
SELECT is_deleted FROM member WHERE member_id = 3;
```

[실행 결과]

is_deleted
1

현재 탈퇴 상태라는 것만 알 수 있다. 언제 탈퇴하고 언제 복구했는지 이력은 알 수 없다.

SOFT DELETE 정리

`is_deleted` 컬럼을 사용한 SOFT DELETE는 다음과 같은 장단점이 있다.

장점	설명
구현이 간단하다	컬럼 하나만 추가하면 된다

복구가 쉽다	UPDATE 한 번으로 복구 가능하다
연관 데이터 유지	외래 키 관계를 유지하면서 삭제 처리할 수 있다
분석 가능	삭제된 데이터도 통계에 활용할 수 있다

단점	설명
삭제 시점 불명	언제 삭제되었는지 알 수 없다
보관 기간 관리 불가	일정 기간 후 영구 삭제 정책을 적용하기 어렵다
이력 추적 불가	삭제/복구 이력을 관리할 수 없다

이런 한계를 해결하기 위해 다음 수업에서는 `deleted_at` 날짜 컬럼을 사용하는 방식을 알아보겠다.

SOFT DELETE - deleted_at 컬럼 방식 1

앞서 `is_deleted` 컬럼 방식의 한계를 살펴보았다. 삭제 시점을 알 수 없고, 보관 기간 관리가 어렵다는 문제가 있었다.
이번 수업에서는 이 문제를 해결하는 `deleted_at` 컬럼 방식을 알아보겠다.

deleted_at 컬럼의 아이디어

`is_deleted`는 "삭제되었는가?"라는 상태만 저장한다. 반면 `deleted_at`은 "언제 삭제되었는가?"라는 시점을 저장한다. 그리고 이 시점 정보만으로 삭제 여부도 함께 판단할 수 있다.

deleted_at 값	의미
NULL	삭제되지 않음 (활성 상태)
2026-01-20 14:30:00	해당 시점에 삭제됨

`deleted_at`이 `NULL`이면 삭제되지 않은 것이고, 값이 있으면 그 시점에 삭제된 것이다. 하나의 컬럼으로 삭제 여부와 삭제 시점을 모두 알 수 있다.

deleted_at 컬럼 적용하기

회원 테이블에 `deleted_at` 컬럼을 적용해보자.

```
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS member;

CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(20),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME DEFAULT NULL
);

INSERT INTO member (username, email, phone) VALUES
('kim', 'kim@example.com', '010-1111-1111'),
('lee', 'lee@example.com', '010-2222-2222'),
('park', 'park@example.com', '010-3333-3333'),
('choi', 'choi@example.com', '010-4444-4444');
```

```
SELECT * FROM member;
```

[실행 결과]

member_id	username	email	phone	created_at	delete
1	kim	kim@example.com	010-1111-1111	2026-01-15 10:00:00	NULL
2	lee	lee@example.com	010-2222-2222	2026-01-15 10:00:00	NULL

3	park	park@example.com	010-3333-3333	2026-01-15 10:00:00	NULL
4	choi	choi@example.com	010-4444-4444	2026-01-15 10:00:00	NULL

모든 회원의 `deleted_at`이 `NULL`이다. 아무도 삭제되지 않았다는 뜻이다.

SOFT DELETE 실행하기

`park` 회원이 2026년 1월 20일 오후 2시 30분에 탈퇴를 요청했다.

```
UPDATE member
SET deleted_at = '2026-01-20 14:30:00'
WHERE member_id = 3;
```

물론 실무에서는 보통 다음과 같이 현재 시간을 사용한다. 원활한 예제 진행을 위해 여기서는 시간을 직접 입력하겠다.

```
-- 실무에서는 이렇게 현재 시간을 사용
UPDATE member
SET deleted_at = NOW()
WHERE member_id = 3;
```

```
SELECT * FROM member;
```

[실행 결과]

member_id	username	email	phone	created_at	deleted_at
1	kim	kim@example.com	010-1111-1111	2026-01-15 10:00:00	NULL

2	lee	lee@example.com	010-2222-2222	2026-01-15 10:00:00	NULL
3	park	park@example.com	010-3333-3333	2026-01-15 10:00:00	2026-01-2(14:30:00
4	choi	choi@example.com	010-4444-4444	2026-01-15 10:00:00	NULL

`park` 회원의 `deleted_at`에 삭제 시점이 기록되었다.

삭제되지 않은 데이터 조회하기

`is_deleted` 방식에서는 `WHERE is_deleted = FALSE` 조건을 사용했다. `deleted_at` 방식에서는 `WHERE deleted_at IS NULL` 조건을 사용한다.

```
-- 활성 회원만 조회
SELECT * FROM member WHERE deleted_at IS NULL;
```

[실행 결과]

member_id	username	email	phone	created_at	deleted_a
1	kim	kim@example.com	010-1111-1111	2026-01-15 10:00:00	NULL
2	lee	lee@example.com	010-2222-2222	2026-01-15 10:00:00	NULL
4	choi	choi@example.com	010-4444-4444	2026-01-15 10:00:00	NULL

삭제된 `park` 회원이 보이지 않는다.

삭제된 회원만 조회해보자.

```sql
-- 삭제된 회원만 조회
SELECT * FROM member WHERE deleted_at IS NOT NULL;
```

[실행 결과]

member_id	username	email	phone	created_at	deleted_at
3	park	park@example.com	010-3333-3333	2026-01-15 10:00:00	2026-01-20 14:30:00

복구하기

복구는 `deleted_at`을 다시 `NULL`로 변경하면 된다.

```sql
UPDATE member
SET deleted_at = NULL
WHERE member_id = 3;
```

```sql
SELECT * FROM member WHERE deleted_at IS NULL;
```

[실행 결과]

member_id	username	email	phone	created_at	deleted_at
1	kim	kim@example.com	010-1111-1111	2026-01-15 10:00:00	NULL
2	lee	lee@example.com	010-2222-2222	2026-01-15 10:00:00	NULL
3	park	park@example.com	010-3333-3333	2026-01-15 10:00:00	NULL

4	choi	choi@example.com	010-4444-4444	2026-01-15 10:00:00	NULL

삭제 시점 활용하기

`deleted_at`의 가장 큰 장점은 삭제 시점을 알 수 있다는 것이다. 다양한 시나리오에서 이 정보를 활용할 수 있다.

시나리오 1 - 고객센터 문의 대응

다시 `park` 회원을 삭제하고, 고객센터 문의에 대응해보자.

```
UPDATE member
SET deleted_at = '2026-01-20 14:30:00'
WHERE member_id = 3;
```

"park 회원이 언제 탈퇴했나요?"

```
SELECT username, deleted_at
FROM member
WHERE member_id = 3;
```

[실행 결과]

username	deleted_at
park	2026-01-20 14:30:00

정확한 탈퇴 시점을 바로 알 수 있다.

시나리오 2 - 일정 기간 후 영구 삭제

개인정보보호법에 따라 탈퇴 후 1년이 지난 회원의 데이터를 영구 삭제해야 한다.

먼저 테스트를 위해 다양한 시점에 탈퇴한 회원 데이터를 준비하자.

```sql
DROP TABLE IF EXISTS member;

CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME DEFAULT NULL
);

INSERT INTO member (username, email, deleted_at) VALUES
('kim', 'kim@example.com', NULL),
('lee', 'lee@example.com', NULL),
('park', 'park@example.com', '2025-01-15 10:00:00'),
('choi', 'choi@example.com', '2025-06-20 15:30:00'),
('jung', 'jung@example.com', '2026-01-10 09:00:00'),
('kang', 'kang@example.com', '2026-01-18 11:00:00');
```

```sql
SELECT username, deleted_at FROM member;
```

[실행 결과]

username	deleted_at
kim	NULL
lee	NULL
park	2025-01-15 10:00:00
choi	2025-06-20 15:30:00
jung	2026-01-10 09:00:00
kang	2026-01-18 11:00:00

현재 날짜가 2026년 1월 20일이라고 가정하면, 1년 전인 2025년 1월 20일 이전에 탈퇴한 회원을 찾아야 한다.

```sql
-- 1년 전에 탈퇴한 회원 조회 (현재: 2026-01-20 기준)
SELECT username, deleted_at
FROM member
WHERE deleted_at IS NOT NULL
AND deleted_at < '2025-01-20 00:00:00';
```

[실행 결과]

username	deleted_at
park	2025-01-15 10:00:00

`park` 회원만 1년이 지났다. 이제 이 회원의 데이터를 영구 삭제(HARD DELETE) 하면 된다.

```sql
-- 1년 지난 회원 영구 삭제
SET SQL_SAFE_UPDATES = 0; -- 안전 업데이트 모드 끄기

DELETE FROM member
WHERE deleted_at IS NOT NULL
AND deleted_at < '2025-01-20 00:00:00';

SET SQL_SAFE_UPDATES = 1; -- 안전 업데이트 모드 활성화
```

```sql
SELECT username, deleted_at FROM member;
```

[실행 결과]

username	deleted_at
kim	NULL
lee	NULL
choi	2025-06-20 15:30:00
jung	2026-01-10 09:00:00

kang	2026-01-18 11:00:00

`park` 회원이 영구 삭제되었다. 실무에서는 이런 작업을 배치 작업(Batch Job)으로 매일 자동 실행한다.

```
-- 실무에서 사용하는 형태 (현재 시간 기준 1년 전)
DELETE FROM member
WHERE deleted_at IS NOT NULL
AND deleted_at < DATE_SUB(NOW(), INTERVAL 1 YEAR);
```

시나리오 3 - 특정 기간 탈퇴 현황 분석

마케팅팀에서 "2026년 1월에 탈퇴한 회원이 몇 명인가요?"라고 물어왔다.

```
-- 2026년 1월에 탈퇴한 회원 수
SELECT COUNT(*) AS churned_count
FROM member
WHERE deleted_at >= '2026-01-01 00:00:00'
AND deleted_at < '2026-02-01 00:00:00';
```

[실행 결과]

churned_count
2

월별 탈퇴 현황도 물어본다.

```
-- 월별 탈퇴 현황 분석
SELECT
    DATE_FORMAT(deleted_at, '%Y-%m') AS month,
    COUNT(*) AS churned_count
FROM member
WHERE deleted_at IS NOT NULL
GROUP BY DATE_FORMAT(deleted_at, '%Y-%m')
ORDER BY month;
```

[실행 결과]

month	churned_count
2025-06	1
2026-01	2

이런 분석은 `is_deleted` 방식으로는 불가능하다.

SOFT DELETE - deleted_at 컬럼 방식 2

주문 테이블에도 적용하기

주문 테이블에도 `deleted_at`을 적용해보자.

```
DROP TABLE IF EXISTS orders;

CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_name VARCHAR(100) NOT NULL,
    amount INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME DEFAULT NULL,
    FOREIGN KEY (member_id) REFERENCES member(member_id)
);

INSERT INTO orders (member_id, product_name, amount, order_date) VALUES
(1, '노트북', 1500000, '2026-01-10 10:00:00'),
(1, '마우스', 50000, '2026-01-12 14:00:00'),
(2, '키보드', 150000, '2026-01-11 11:00:00'),
(2, '모니터', 300000, '2026-01-15 09:00:00');
```

회원이 특정 주문을 취소했다고 가정하자.

```
-- 주문 취소 (SOFT DELETE)
UPDATE orders
SET deleted_at = NOW()
WHERE order_id = 2;
```

```
SELECT * FROM orders;
```

[실행 결과]

order_id	member_id	product_name	amount	order_date	deleted_at
1	1	노트북	1500000	2026-01-10 10:00:00	NULL
2	1	마우스	50000	2026-01-12 14:00:00	2026-01-20 14:30:00
3	2	키보드	150000	2026-01-11 11:00:00	NULL
4	2	모니터	300000	2026-01-15 09:00:00	NULL

활성 주문만 조회할 때는 `deleted_at IS NULL` 조건을 추가한다.

```
-- 활성 주문만 조회
SELECT * FROM orders WHERE deleted_at IS NULL;
```

[실행 결과]

order_id	member_id	product_name	amount	order_date	deleted_at

1	1	노트북	1500000	2026-01-10 10:00:00	NULL
3	2	키보드	150000	2026-01-11 11:00:00	NULL
4	2	모니터	300000	2026-01-15 09:00:00	NULL

조인 시 주의사항

SOFT DELETE를 사용하면 조인할 때 양쪽 테이블 모두에서 삭제 여부를 확인해야 한다.

```
-- 잘못된 예: 회원만 삭제 여부 확인
SELECT m.member_id, m.username, o.order_id,
       o.product_name, o.amount, o.deleted_at as order_deleted_at
FROM member m
JOIN orders o ON m.member_id = o.member_id
WHERE m.deleted_at IS NULL;
```

[실행 결과]

member_id	username	order_id	product_name	amount	order_deleted_at
1	kim	1	노트북	1500000	
1	kim	2	마우스	50000	2026-01-20 14:30:00
2	lee	3	키보드	150000	
2	lee	4	모니터	300000	

취소된 주문(마우스)이 포함되어 있다. 주문의 삭제 여부도 확인해야 한다.

```
```

```sql
-- 올바른 예: 양쪽 모두 삭제 여부 확인
SELECT m.member_id, m.username, o.order_id,
       o.product_name, o.amount, o.deleted_at as order_deleted_at
FROM member m
JOIN orders o ON m.member_id = o.member_id
WHERE m.deleted_at IS NULL
AND o.deleted_at IS NULL;
```

[실행 결과]

member_id	username	order_id	product_name	amount	order_deleted_at
1	kim	1	노트북	1500000	
2	lee	3	키보드	150000	
2	lee	4	모니터	300000	

이제 취소된 주문이 제외되었다.

뷰(View)를 활용한 편의성 개선

매번 `WHERE deleted_at IS NULL`을 작성하는 것은 번거롭고, 실수하기 쉽다. 뷰를 사용하면 이 문제를 해결할 수 있다.

```sql
-- 활성 회원만 보여주는 뷰
CREATE VIEW active_member AS
SELECT member_id, username, email, created_at
FROM member
WHERE deleted_at IS NULL;

-- 활성 주문만 보여주는 뷰
CREATE VIEW active_orders AS
SELECT order_id, member_id, product_name, amount, order_date
FROM orders
WHERE deleted_at IS NULL;
```

이제 뷰를 사용하면 삭제 조건을 신경 쓰지 않아도 된다.

```
SELECT * FROM active_member;
```

[실행 결과]

member_id	username	email	created_at
1	kim	kim@example.com	2026-01-15 10:00:00
2	lee	lee@example.com	2026-01-15 10:00:00

```
-- 뷰끼리 조인
SELECT m.username, o.product_name, o.amount
FROM active_member m
JOIN active_orders o ON m.member_id = o.member_id;
```

[실행 결과]

username	product_name	amount
kim	노트북	1500000
lee	키보드	150000
lee	모니터	300000

뷰를 사용하면 삭제된 데이터가 자동으로 필터링되어 실수를 방지할 수 있다.

뷰 삭제 쿼리

```
DROP VIEW IF EXISTS active_member;
DROP VIEW IF EXISTS active_orders;
```

is_deleted vs deleted_at 비교

두 방식을 비교해보자.

항목	is_deleted	deleted_at
저장 정보	삭제 여부만	삭제 여부 + 삭제 시점
삭제 확인	`is_deleted = 0`	`deleted_at IS NULL`
삭제 시점	알 수 없음	정확히 알 수 있음
보관 기간 관리	불가능	가능
기간별 분석	불가능	가능
저장 공간	1바이트 (BOOLEAN)	8바이트 (DATETIME)
인덱스 효율	좋음	보통

저장 공간과 인덱스 효율 면에서 `is_deleted`가 약간 유리하지만, 실무에서는 삭제 시점 정보가 거의 항상 필요하다.
따라서 `deleted_at` **방식을 권장한다.**

deleted_at 정리

`deleted_at` 컬럼 방식은 `is_deleted` 방식의 한계를 해결한다.

기능	is_deleted	deleted_at
삭제 여부 확인	O	O
삭제 시점 확인	X	O
일정 기간 후 영구 삭제	X	O
기간별 탈퇴 분석	X	O
복구	O	O

실무에서는 `is_deleted` 보다는 `deleted_at` 방식을 사용하는 경우가 많다.

다음 수업에서는 SOFT DELETE와 HARD DELETE를 언제 사용해야 하는지, 각각의 장단점을 비교해보겠다.

SOFT DELETE vs HARD DELETE

지금까지 SOFT DELETE의 개념과 구현 방법을 배웠다. 그렇다면 모든 테이블에 SOFT DELETE를 적용해야 할까? 이번 수업에서는 SOFT DELETE와 HARD DELETE의 장단점을 비교하고, 언제 어떤 방식을 선택해야 하는지 알아보겠다.

HARD DELETE의 장점

SOFT DELETE의 장점을 많이 이야기했지만, HARD DELETE에도 분명한 장점이 있다.

장점 1 - 저장 공간 절약

SOFT DELETE는 삭제된 데이터도 계속 보관하기 때문에 테이블 크기가 계속 증가한다. 반면 HARD DELETE는 데이터를 실제로 제거하므로 저장 공간을 절약할 수 있다.

```
DROP TABLE IF EXISTS log_data;

CREATE TABLE log_data (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    message VARCHAR(500) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME DEFAULT NULL
);
```

- 100만 건의 로그 데이터가 있다고 가정
- 그 중 90만 건이 SOFT DELETE 되었다면?
- 실제로 필요한 데이터는 10만 건인데, 100만 건을 저장하고 있는 것

특히 로그성 데이터나 대용량 데이터의 경우 SOFT DELETE로 인해 저장 공간이 급격히 증가할 수 있다.

장점 2 - 쿼리 성능

SOFT DELETE를 사용하면 모든 조회 쿼리에 `WHERE deleted_at IS NULL` 조건이 추가된다. 데이터가 많아지

면 이 조건으로 인한 성능 저하가 발생할 수 있다.

```
-- HARD DELETE: 단순한 쿼리
SELECT * FROM member WHERE username = 'kim';

-- SOFT DELETE: 항상 조건이 추가됨
SELECT * FROM member WHERE username = 'kim' AND deleted_at IS NULL;
```

삭제된 데이터가 많을수록 불필요한 데이터를 스캔하게 되어 성능이 저하된다.

장점 3 - 단순한 데이터 모델

SOFT DELETE는 모든 테이블에 `deleted_at` 컬럼을 추가해야 한다. 테이블이 많아지면 관리해야 할 컬럼도 늘어난다.

```
-- SOFT DELETE 적용 시 모든 테이블에 deleted_at 추가
CREATE TABLE member (
    ...
    deleted_at DATETIME DEFAULT NULL
);

CREATE TABLE orders (
    ...
    deleted_at DATETIME DEFAULT NULL
);

CREATE TABLE product (
    ...
    deleted_at DATETIME DEFAULT NULL
);

CREATE TABLE review (
    ...
    deleted_at DATETIME DEFAULT NULL
);
```

HARD DELETE를 사용하면 이런 추가 컬럼이 필요 없다.

장점 4 - 쿼리 작성의 단순함과 실수 방지

실무에서 우리가 작성하는 쿼리의 99%는 '삭제되지 않은 유효한 데이터'를 조회하는 것이다. 탈퇴한 회원이 아니라 현재 활동 중인 회원을 조회하고, 삭제된 상품이 아니라 현재 판매 중인 상품을 리스트로 보여줘야 한다.

SOFT DELETE를 사용하면 이 모든 조회 쿼리에 개발자가 매번 조건을 붙여야 한다.

```
-- SOFT DELETE: 단순히 전체 상품을 조회하고 싶어도 조건을 신경 써야 한다.
SELECT * FROM product WHERE deleted_at IS NULL;

-- 만약 조인을 한다면? 조인하는 테이블마다 조건을 걸어야 할 수도 있다.
SELECT p.*
  FROM product p
  JOIN category c ON p.category_id = c.category_id
 WHERE p.deleted_at IS NULL
   AND c.deleted_at IS NULL;
```

이것은 단순한 귀찮음의 문제가 아니다. **휴먼 에러(Human Error)** 의 가능성을 열어두는 것이다. 팀원이 실수로 `deleted_at IS NULL` 조건을 빠뜨린다면, 삭제된 상품이 메인 페이지에 노출되거나 탈퇴한 회원이 버젓이 조회되는 심각한 버그가 발생한다.

반면, HARD DELETE는 데이터가 물리적으로 사라지기 때문에 별도의 조건이 필요 없다.

```
-- HARD DELETE: 우리가 생각하는 직관 그대로 쿼리를 작성하면 된다.
SELECT * FROM product;
```

[실행 결과]

product_id	name	price	stock_quantity	...
1	스마트폰 케이스	12000	95	...
2	무선 이어폰	89000	50	...
...	...	...	...	...

이처럼 HARD DELETE를 사용하면 개발자가 신경 써야 할 로직이 줄어들고, 실수를 원천적으로 차단할 수 있다는 큰

장점이 있다.

SOFT DELETE의 장점 (복습)

앞서 배운 내용을 정리하면 다음과 같다.

장점	설명
복구 용이	UPDATE 한 번으로 즉시 복구 가능
데이터 보존	삭제 후에도 분석, 감사에 활용 가능
참조 무결성 유지	외래 키 관계를 깨지 않고 삭제 처리
법적 요구사항 충족	일정 기간 데이터 보관 의무 준수
삭제 시점 추적	`deleted_at`으로 언제 삭제되었는지 확인

실무에서의 선택 기준

모든 테이블에 SOFT DELETE를 적용할 필요는 없다. 비즈니스 요구사항에 따라 선택해야 한다.

SOFT DELETE를 사용하는 경우

1. 복구 요청이 자주 발생하는 데이터

```
-- 회원 데이터: 탈퇴 후 복구 요청이 종종 발생
CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    deleted_at DATETIME DEFAULT NULL  -- SOFT DELETE 적용
);
```

회원이 실수로 탈퇴 버튼을 누르거나, 마음이 바뀌어 복구를 요청하는 경우가 있다. SOFT DELETE를 사용하면 즉시 복구할 수 있다.

2. 법적 보관 의무가 있는 데이터

```
-- 주문 데이터: 전자상거래법에 따라 5년 보관 의무
CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    total_amount INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME DEFAULT NULL  -- SOFT DELETE 적용
);

-- 결제 데이터: 금융 관련 법규에 따라 보관 의무
CREATE TABLE payment (
    payment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    order_id BIGINT NOT NULL,
    amount INT NOT NULL,
    payment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    deleted_at DATETIME DEFAULT NULL  -- SOFT DELETE 적용
);
```

3. 연관 데이터가 많은 핵심 엔티티

```
-- 상품 데이터: 주문, 리뷰, 장바구니 등 많은 테이블이 참조
CREATE TABLE product (
    product_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    price INT NOT NULL,
    deleted_at DATETIME DEFAULT NULL  -- SOFT DELETE 적용
);
```

상품을 HARD DELETE하면 해당 상품을 참조하는 주문, 리뷰, 장바구니 등의 데이터도 처리해야 한다. SOFT DELETE를 사용하면 참조 무결성을 유지하면서 삭제 처리할 수 있다.

4. 감사(Audit) 추적이 필요한 데이터

```
-- 계약 데이터: 누가 언제 삭제했는지 추적 필요
CREATE TABLE contract (
    contract_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    client_name VARCHAR(100) NOT NULL,
```

```
    amount BIGINT NOT NULL,
    deleted_at DATETIME DEFAULT NULL,   -- SOFT DELETE 적용
    deleted_by BIGINT DEFAULT NULL      -- 삭제한 사람도 기록
);
```

HARD DELETE를 사용하는 경우

1. 로그성 데이터

```
-- API 호출 로그: 일정 기간 후 삭제해도 문제없음
CREATE TABLE api_log (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    endpoint VARCHAR(200) NOT NULL,
    response_time INT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    -- deleted_at 없음: HARD DELETE 사용
);

-- 오래된 로그 삭제
DELETE FROM api_log
WHERE created_at < DATE_SUB(NOW(), INTERVAL 30 DAY);
```

로그 데이터는 일정 기간이 지나면 가치가 떨어지고, 복구할 필요도 없다. HARD DELETE로 저장 공간을 관리하는 것이 효율적이다.

2. 임시 데이터

```
-- 장바구니: 임시 저장 데이터
CREATE TABLE cart_item (
    cart_item_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_id BIGINT NOT NULL,
    quantity INT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    -- deleted_at 없음: HARD DELETE 사용
);

-- 장바구니에서 상품 제거
DELETE FROM cart_item WHERE cart_item_id = 1;
```

장바구니 데이터는 임시 데이터이다. 사용자가 상품을 제거하면 복구할 필요가 없다.

3. 개인정보 완전 삭제 요청

개인정보보호법에 따르면, 사용자가 개인정보 삭제를 요청하면 완전히 삭제해야 할 수 있다.

```
-- 개인정보 완전 삭제 요청 처리
-- 1. 먼저 관련 데이터를 익명화하거나 삭제
UPDATE orders SET member_id = NULL WHERE member_id = 3;

-- 2. 회원 데이터 완전 삭제
DELETE FROM member WHERE member_id = 3;
```

이 경우 SOFT DELETE로는 법적 요구사항을 충족하지 못할 수 있다.

테이블별 삭제 전략 예시

쇼핑몰 프로젝트 예시를 기반으로 테이블별로 어떤 삭제 전략을 선택할지 정리해보자.

```
-- 회원: SOFT DELETE (복구 요청, 주문 내역 보존)
CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    deleted_at DATETIME DEFAULT NULL
);

-- 상품: SOFT DELETE (주문, 리뷰 참조, 판매 중단 처리)
CREATE TABLE product (
    product_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    price INT NOT NULL,
    deleted_at DATETIME DEFAULT NULL
);

-- 주문: SOFT DELETE (법적 보관 의무, 매출 분석)
```

```sql
CREATE TABLE orders (
    order_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT,
    total_amount INT NOT NULL,
    deleted_at DATETIME DEFAULT NULL
);

-- 리뷰: SOFT DELETE (신고 처리, 복구 가능)
CREATE TABLE review (
    review_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_id BIGINT NOT NULL,
    content TEXT,
    deleted_at DATETIME DEFAULT NULL
);

-- 장바구니: HARD DELETE (임시 데이터)
CREATE TABLE cart_item (
    cart_item_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    product_id BIGINT NOT NULL,
    quantity INT NOT NULL
);

-- 알림: HARD DELETE (일정 기간 후 삭제)
CREATE TABLE notification (
    notification_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    member_id BIGINT NOT NULL,
    message VARCHAR(500),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- API 로그: HARD DELETE (로그성 데이터)
CREATE TABLE api_log (
    log_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    endpoint VARCHAR(200),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

테이블	삭제 전략	이유

member	SOFT DELETE	복구 요청, 주문 내역 참조
product	SOFT DELETE (실무에선 status 권장)	주문/리뷰 참조, 판매 중단 처리
orders	SOFT DELETE (실무에선 status 권장)	법적 보관 의무, 매출 분석
review	SOFT DELETE	신고 처리 후 복구 가능성
cart_item	HARD DELETE	임시 데이터
notification	HARD DELETE	일정 기간 후 불필요
api_log	HARD DELETE	로그성 데이터

이것은 아주 단순한 예시이다. 실무에서는 상태(status)를 활용한 더욱 다양한 선택지가 있는데, 그 부분을 다음 시간에 알아볼 것이다.

우리는 지금까지 `deleted_at` 컬럼을 사용하는 SOFT DELETE 방식을 배웠다. 그리고 많은 테이블에 이 방식이 적용되어야 할 것 처럼 느껴진다. 하지만 실제 실무에서는 생각보다 SOFT DELETE를 많이 사용하지는 않는다. 왜 그런지 다음 시간에는 실무 이야기를 들어보자.

SOFT vs HARD vs STATUS

실무 팁: 상태(Status) 기반의 데이터 관리

이번에는 이론적인 이야기 보다는 실무 이야기를 해보자.
앞서 우리는 데이터를 삭제하는 대신 보관하기 위해 `deleted_at` 컬럼을 사용하는 SOFT DELETE 방식을 배웠다. 하지만 실무, 특히 규모가 있는 커머스 시스템에서는 상품이나 주문 테이블에 `deleted_at` 컬럼을 만들지 않는 경우가 많다.

왜 그럴까? **'삭제'라는 행위보다 더 구체적인 '상태' 변경이 필요하기 때문이다.**

1. 상품의 '삭제' vs '판매 중지'
쇼핑몰 운영자가 "이 상품 이제 안 팔 거니까 삭제해 주세요"라고 요청했다고 가정하자. 개발자가 이 요청을 듣고 `deleted_at`에 날짜를 채워 목록에서 안 보이게 만들었다.

그런데 며칠 뒤 운영자가 다시 와서 이렇게 말한다. "아까 삭제한 상품, 재고가 남아서 다시 팔아야겠어요. 그리고 아예 안 파는 게 아니라 '일시 품절' 상태로 보여줄 순 없나요?"

단순히 `deleted_at`이 `NULL`이냐 아니냐로 판단하는 이분법적인 SOFT DELETE로는 이러한 다양한 비즈니스 요구사항을 처리하기 어렵다. 상품은 '삭제'되는 것이 아니라 '상태'가 변하는 것이다.

따라서 실무에서는 `deleted_at` 대신 `status` 컬럼을 사용하여 상품의 생명주기를 관리한다.

```
-- status 컬럼을 사용한 상품 테이블 예시
CREATE TABLE product (
    product_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price INT NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'ON_SALE', -- 판매 상태 (ON_SALE,
STOP_SALE, SOLD_OUT)
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

- 예시이다. 실행하지 말자.

이제 상품을 삭제하는 대신 상태를 변경하면 된다.

```
-- 상품 판매 중지 처리 (삭제가 아님)
UPDATE product
   SET status = 'STOP_SALE'
 WHERE product_id = 1;
```

조회할 때도 훨씬 직관적이고 유연하다.

```
-- 판매 중인 상품만 조회
SELECT * FROM product WHERE status = 'ON_SALE';

-- 관리자 페이지: 판매 중지된 상품도 포함해서 조회하되, 상태를 표시
SELECT * FROM product;
```

[실행 결과 - 예시]

product_id	name	price	status
1	스마트폰 케이스	12000	**STOP_SALE**
2	무선 이어폰	89000	**ON_SALE**

이렇게 하면 데이터는 그대로 보존되면서(SOFT DELETE의 장점), 비즈니스 상황을 훨씬 더 명확하게 표현할 수 있다.

2. 주문의 '삭제' vs '주문 취소'

주문 데이터는 더욱 명확하다. 회계 및 정산 이슈 때문에 주문 데이터는 절대 삭제하면 안 된다. 사용자가 "주문 취소할게요"라고 했을 때, 데이터를 지우는 것이 아니라 상태를 '취소(CANCEL)'로 변경해야 한다.

이미 앞서 우리가 설계한 `orders` 테이블을 다시 살펴보자.

```
CREATE TABLE orders (
    order_id        BIGINT          NOT NULL AUTO_INCREMENT,
    -- ... 생략 ...
    status          VARCHAR(20)     NOT NULL DEFAULT 'ORDERED', -- 주문 상태
    -- ... 생략 ...
);
```

우리는 이미 `status` 컬럼을 가지고 있다. 이 `status` 컬럼이 `ORDERED`(주문 완료), `SHIPPED`(배송 중), `DELIVERED`(배송 완료), `CANCELLED`(주문 취소) 등의 값을 가지며 주문의 전체 수명주기를 관리한다.

이미 `status`가 '주문 취소'라는 상태를 표현할 수 있는데, 굳이 `deleted_at` 컬럼을 추가해서 이중으로 관리할 필요가 있을까?

- **나쁜 설계**: `status = 'CANCELLED'` 이면서 `deleted_at`도 값이 들어있는 경우
- **좋은 설계**: `status = 'CANCELLED'` 하나로 모든 상황을 설명하는 경우

따라서 주문 테이블에는 `deleted_at` 컬럼이 필요 없다. 상태 값 하나로 유효한 데이터인지, 취소된 데이터인지, 진행중인 데이터인지 모두 구분할 수 있기 때문이다.

정리: 언제 무엇을 써야 할까?

실무에서 데이터 삭제 전략을 결정할 때 다음 기준을 참고하자.

1. **상태(Status) 기반 관리**: 비즈니스 프로세스가 복잡하고 데이터의 생명주기가 중요한 경우
    - **예시**: 상품(판매중/품절/판매중지), 주문(결제대기/완료/취소/환불), 배송(준비/이동/완료)
    - **방법**: `deleted_at` 없이 `status` 컬럼만 사용한다.

2. **SOFT DELETE (`deleted_at`)**: 단순히 데이터의 존재 유무가 중요하고, 복구 가능성만 열어두면 되는 경우
    - **예시**: 댓글, 게시글, 좋아요, 회원(탈퇴)
    - **방법**: `status` 컬럼 없이 `deleted_at` 컬럼을 사용한다.
    - 참고: 회원의 경우에도 서비스가 복잡해서 다양한 상태가 있다면 상태 기반 관리를 적용하는 것이 좋다.

3. **HARD DELETE**: 데이터 보존 가치가 없는 경우
    - **예시**: 장바구니, 로그, 임시 데이터

마무리

- **삭제는 기술이 아니라 정책**이다. (복구/법/감사/분석/무결성)
- **핵심 도메인(주문/결제/정산)은 보통 삭제가 아니라 "상태"**로 관리한다.
- **SOFT DELETE는 편하지만 비용(쿼리/인덱스/휴먼에러/용량)을 치른다.**

결론적으로, **모든 테이블에 기계적으로 SOFT DELETE를 적용하지 마라.** 상품이나 주문처럼 비즈니스의 핵심이 되는 데이터는 '삭제'라는 개념보다 '상태 변경'이라는 개념이 훨씬 더 적합하다.

SOFT DELETE와 이력 테이블

우리는 앞서 **데이터 변경 이력(History Table)** 관리 방법과 **SOFT DELETE** 방식을 각각 배웠다.
여기서 다음과 같은 예리한 질문을 할 수 있다.

> "강사님, 이력 테이블(`product_history`)에 모든 데이터가 남아있잖아요. 그러면 메인 테이블(`product`)에서는 그냥 HARD DELETE로 지워버려도 되지 않나요? 어차피 과거 데이터는 이력 테이블에서 찾으면 되니까요."

아주 좋은 질문이다. 이력 테이블이 '데이터의 백업' 역할을 한다면, 메인 테이블은 굳이 SOFT DELETE를 해서 용량을 낭비할 필요가 없어 보인다. 이력 테이블을 SOFT DELETE의 대안으로 사용할 수 있을까?

결론부터 말하자면, **테이블의 관계(Relationship)에 따라 다르다.**

1. 이상적인 시나리오: HARD DELETE + 이력 테이블

만약 메인 테이블의 데이터를 참조하는 다른 테이블이 없다면, 이 방식은 저장 공간을 절약하는 최고의 방법이다.

작동 방식

1. 데이터를 삭제하기 직전에 현재 상태를 이력 테이블에 `DELETE` 타입으로 저장한다.
2. 메인 테이블에서 데이터를 HARD DELETE 한다.

```
-- 예시: 로그 설정 테이블 (참조하는 테이블이 없다고 가정)
START TRANSACTION;

-- 1. 이력 테이블에 마지막 상태 저장 (삭제 이력)
INSERT INTO config_history (config_id, value, change_type, history_created_at)
SELECT config_id, value, 'DELETE', NOW()
FROM config
WHERE config_id = 100;

-- 2. 메인 테이블에서 제거
DELETE FROM config WHERE config_id = 100;

COMMIT;
```

장점

- **메인 테이블 경량화**: 메인 테이블에는 오직 '활성 데이터'만 남는다. 조회 속도가 빠르고 인덱스 크기도 작게 유지된다.
- **조회 쿼리 단순화**: `WHERE deleted_at IS NULL` 조건을 매번 붙일 필요가 없다.

하지만, 실무의 핵심 데이터(회원, 상품, 주문)는 이런 방식을 사용하기 어렵다. 바로 **외래 키(Foreign Key)** 때문이다.

2. 현실적인 제약: 참조 무결성 (Foreign Key)

우리가 다루는 `product` 테이블을 생각해보자. `orders` 테이블이나 `cart` 테이블이 `product_id`를 참조하고 있다.

이 상황에서 `product` 테이블의 데이터를 HARD DELETE 하려고 하면 어떻게 될까?

```
-- 상품 삭제 시도
DELETE FROM product WHERE product_id = 1;
```

[실행 결과]

```
Error Code: 1451. Cannot delete or update a parent row: a foreign key
constraint fails...
```

데이터베이스가 삭제를 막는다. `orders` 테이블에 주문 내역이 남아있기 때문이다.
주문 내역까지 모두 지우지 않는 한, 상품을 물리적으로 삭제할 수 없다. 하지만 주문 내역은 매출 자료이므로 절대 지우면 안 된다.

결국 **참조되는 부모 데이터는 HARD DELETE를 할 수 없다.** 이것이 우리가 메인 테이블에 SOFT DELETE(또는 상태 변경)를 적용하는 이유가 된다.

정리

SOFT DELETE와 이력 테이블은 서로 대체재가 아니라 **보완재**다.

구분	참조 관계가 없는 테이블 (로그, 단순 설정)	참조 관계가 있는 핵심 테이블 (회원, 상품, 주문)
삭제 전략	**HARD DELETE + 이력 보관**	**SOFT DELETE(또는 상태) + 이력 보관**
이유	용량 절약 및 성능 최적화가 우선	데이터 무결성(FK) 유지가 우선
메인 테이블	데이터 삭제됨	`deleted_at` 업데이트 또는 상태 변경
이력 테이블	삭제 전 마지막 상태 보관 (아카이빙)	삭제 이벤트(DELETE) 기록

SOFT DELETE와 인덱스 설계

SOFT DELETE를 도입하면 모든 조회 쿼리에 `WHERE deleted_at IS NULL` 조건이 붙게 된다. 이는 데이터베이스 성능, 특히 **인덱스(Index)** 설계에 큰 영향을 미친다.

문제 상황: 인덱스의 효율성 저하

`product` 테이블에 상품이 1,000만 개 있고, 그중 100만 개(10%)가 삭제된 상태라고 가정하자.
우리는 상품명을 검색할 때 빠르게 찾기 위해 `name` 컬럼에 인덱스를 걸어두었다 가정하자.

```
-- 인덱스 생성
CREATE INDEX idx_product_name ON product (name);

-- 상품 검색 쿼리 (SOFT DELETE 적용)
SELECT * FROM product
 WHERE name = '스마트폰 케이스'
   AND deleted_at IS NULL;
```

이 쿼리는 효율적일까?
대부분의 경우 효율적이다. `name` 인덱스를 통해 '스마트폰 케이스'를 먼저 찾고, 그 데이터가 삭제되었는지(`IS NULL`) 확인하기 때문이다.

하지만 문제는 **대부분의 데이터가 삭제되지 않은 상태(`NULL`)** 라는 점이다.
데이터베이스 입장에서 `deleted_at IS NULL` 조건은 변별력(Cardinality)이 매우 낮다. 전체 데이터의 90% 이상이 `NULL`일 것이기 때문이다.

잘못된 인덱스 설계

초보자들이 흔히 하는 실수는 `deleted_at`을 인덱스의 첫 번째 컬럼으로 두는 것이다.

```
-- 잘못된 인덱스 설계: deleted_at이 선두 컬럼
```

```
CREATE INDEX idx_deleted_name ON product (deleted_at, name);
```

이렇게 하면 인덱스 구조상 `deleted_at`이 `NULL`인 데이터를 먼저 다 모아놓고, 그 안에서 이름을 찾게 된다.
전체 데이터의 90% 이상이 `NULL`이므로, 인덱스의 효과가 거의 없고 비효율적이다.
추가로 대부분 조회가 `(비즈니스 조건) + deleted_at IS NULL`이므로 선두는 비즈니스 조건이 되는 게 자연스럽다.

올바른 인덱스 설계 전략

1. 일반적인 경우: `deleted_at`은 인덱스에 포함하지 않거나 마지막에 둔다
대부분의 경우 비즈니스 키(예: `name`, `category_id`, `status`)의 변별력이 훨씬 높다. 따라서 기존 인덱스를 그대로 사용해도 무방하다. 데이터베이스 옵티마이저는 `name`으로 데이터를 거른 뒤, 남은 소수의 데이터 중에서 `deleted_at`을 필터링하는 것이 훨씬 빠르다고 판단한다.

2. 커버링 인덱스(Covering Index)가 필요한 경우
쿼리 성능을 극대화하기 위해 인덱스만으로 쿼리를 끝내는 '커버링 인덱스'를 사용한다면 `deleted_at`을 인덱스 뒤쪽에 포함시켜야 한다.

```
-- 좋은 설계: 비즈니스 키 + deleted_at
CREATE INDEX idx_name_deleted ON product (name, deleted_at);
```

이렇게 하면 `name`으로 검색한 후, 테이블을 열어보지 않고 인덱스 자체에서 `deleted_at` 정보까지 확인하여 바로 결과를 반환할 수 있다.

UNIQUE 제약조건과 SOFT DELETE의 충돌

SOFT DELETE 사용 시 가장 골치 아픈 문제가 **UNIQUE 인덱스**다.

회원 테이블의 `login_id`는 중복되면 안 되므로 `UNIQUE` 제약조건이 걸려있다.

1. `userA`가 가입한다.
2. `userA`가 탈퇴한다. (`deleted_at` 업데이트)
3. 다른 사람이 `userA`라는 아이디로 다시 가입하려고 한다.

```sql
-- 1. userA 가입
INSERT INTO member (login_id, deleted_at) VALUES ('userA', NULL);

-- 2. userA 탈퇴
UPDATE member SET deleted_at = NOW() WHERE login_id = 'userA';

-- 3. 새로운 userA 가입 시도
INSERT INTO member (login_id, deleted_at) VALUES ('userA', NULL);
```

[실행 결과]

```
Error Code: 1062. Duplicate entry 'userA' for key 'member.uq_login_id'
```

탈퇴한 회원이지만 데이터가 물리적으로 남아있고, `login_id` 컬럼에 `UNIQUE` 제약조건이 걸려있어서 재가입이 불가능하다.

해결 방법 1: 탈퇴 시 값을 변경하기 (추천)

탈퇴할 때 `login_id`를 변경하여 충돌을 피한다. 보통 타임스탬프 등을 붙인다.

```sql
-- 탈퇴 처리: 아이디 뒤에 탈퇴 시간을 붙임
UPDATE member
   SET deleted_at = NOW(),
       login_id = CONCAT(login_id, '_deleted_', UNIX_TIMESTAMP())
 WHERE member_id = 3;
```

- 이렇게 하면 `userA`라는 아이디 공간이 비워지므로 재가입이 가능하다.
  - `userA` ➜ `userA_deleted_1209381209`

해결 방법 2: deleted_at을 UNIQUE 인덱스에 포함하기 (비추천)

`(login_id, deleted_at)`으로 UNIQUE 인덱스를 건다.
하지만 이 방식은 `deleted_at`이 `NULL`인 경우(활성 회원) 중복 체크가 제대로 되지 않을 수 있다(DB 종류에 따라 `NULL`은 중복으로 치지 않는 경우가 있음). 또한 탈퇴한 회원이 여러 번 생기면 `deleted_at`이 달라져야만 저장되는 등 관리가 복잡하다.

따라서 실무에서는 **해결 방법 1 (탈퇴 시 식별자 변경)** 을 주로 사용한다.

정리

1. SOFT DELETE를 써도 인덱스 선두 컬럼으로 `deleted_at`을 쓰지 마라.
2. 성능 최적화가 필요하면 `(비즈니스 컬럼, deleted_at)` 순서로 복합 인덱스를 구성해라.
3. UNIQUE 컬럼이 있다면 탈퇴 시 값을 변경(마스킹)하여 충돌을 방지해라.

이제 여러분은 데이터 삭제에 대한 모든 준비를 마쳤다. HARD DELETE, SOFT DELETE, 그리고 이력 테이블까지. 자신의 프로젝트 환경에 맞는 적절한 무기를 선택해서 데이터를 안전하게 지키길 바란다.

정리

SOFT DELETE가 필요한 이유

HARD DELETE(물리 삭제)의 문제점

- **복구 불가**: `DELETE`로 삭제된 데이터는 백업 없이는 복구가 불가능하다.
- **연관 데이터 소실**: 외래 키 제약 조건으로 인해 자식 데이터도 함께 삭제하거나(`CASCADE`), 부모 정보를 잃게 (`SET NULL`) 되어 데이터 무결성이 깨질 수 있다.
- **법적/비즈니스 요구사항 미충족**: 거래 기록 보관 의무(전자상거래법 등)나 매출 분석, 감사(Audit) 추적이 불가능해진다.
- **삭제 시점 불명**: 언제 삭제되었는지 알 수 없다.

SOFT DELETE - is_deleted 컬럼 방식

- **개념**: 데이터를 실제로 지우지 않고 `is_deleted` 컬럼을 `TRUE`로 업데이트하여 논리적으로만 삭제 처리한다.
- **장점**: 구현이 간단하고 즉시 복구가 가능하며 연관 데이터를 유지할 수 있다.
- **단점**:
  - 삭제된 시점을 알 수 없다.
  - 일정 기간 후 영구 삭제와 같은 정책을 적용하기 어렵다.
  - 탈퇴 및 복구 이력을 추적할 수 없다.

SOFT DELETE - deleted_at 컬럼 방식 1

- **개념**: `deleted_at` 컬럼에 삭제 시간을 기록한다. (`NULL`이면 활성, 시간이 있으면 삭제됨)
- **장점**:

- `is_deleted`의 한계를 극복하여 정확한 삭제 시점을 알 수 있다.
- 법적 보관 기간(예: 1년) 경과 후 영구 삭제하는 배치를 구현할 수 있다.
- 월별 탈퇴 현황 등 기간별 분석이 가능하다.

SOFT DELETE - deleted_at 컬럼 방식 2

- **조인 시 주의사항**: 조인하는 모든 테이블에 대해 `deleted_at IS NULL` 조건을 확인해야 삭제된 데이터가 포함되지 않는다.
- **뷰(View) 활용**: 매번 조건을 붙이는 실수를 방지하기 위해, 활성 데이터만 조회하는 뷰(`active_member` 등)를 생성하여 사용할 수 있다.
- **비교**: 저장 공간을 조금 더 차지하더라도 정보 가치가 높은 `deleted_at` **방식이 실무에서 권장**된다.

SOFT DELETE vs HARD DELETE

- **HARD DELETE의 장점**: 저장 공간 절약, 쿼리 성능 우수(조건절 불필요), 모델과 로직의 단순함, 휴먼 에러 방지.
- **선택 기준**:
  - **SOFT DELETE**: 복구 요청이 잦은 데이터(회원), 법적 보관 의무가 있는 데이터(주문, 결제), 참조가 많은 핵심 데이터(상품).
  - **HARD DELETE**: 로그 데이터, 임시 데이터(장바구니), 개인정보 완전 삭제가 필요한 경우.

SOFT vs HARD vs STATUS

- **실무 팁**: 단순한 삭제 여부(`deleted_at`)보다 **상태(Status)** 기반 관리가 필요한 경우가 많다.
- **상태 관리의 필요성**:
  - 상품: 삭제뿐만 아니라 '판매 중', '품절', '판매 중지' 등 다양한 상태가 존재한다.
  - 주문: '주문 완료', '배송 중', '취소', '환불' 등 생명주기관리가 필요하다.
- **전략**:
  - 복잡한 생명주기 → **Status 컬럼** 사용.
  - 단순 존재 여부 및 복구 → **Soft Delete (`deleted_at`)**.
  - 보관 가치 없음 → **Hard Delete**.

SOFT DELETE와 이력 테이블

- **이력 테이블과 삭제 전략의 관계**:
  - **참조 관계가 없는 경우(로그 등)**: 메인 테이블은 HARD DELETE 하고, 삭제 전 데이터를 이력 테이블에 백업하여 용량과 성능을 최적화한다.
  - **참조 관계가 있는 경우(회원, 주문 등)**: 외래 키 제약 때문에 부모 데이터를 HARD DELETE 할 수 없으므로, **SOFT DELETE(또는 상태 변경) + 이력 보관**을 병행해야 한다.

SOFT DELETE와 인덱스 설계

- **인덱스 효율성**: 대부분의 데이터가 활성(`NULL`) 상태이므로, `deleted_at`을 인덱스 선두 컬럼으로 두면 효율이 떨어진다.
- **올바른 설계**:
  - 비즈니스 키(예: `name`)를 선두로 두고 기존 인덱스를 그대로 사용한다.
  - 커버링 인덱스가 필요한 경우에는 `deleted_at`을 인덱스 마지막에 추가한다.
- **Unique 제약조건 충돌**:
  - 탈퇴 후 재가입 시 `Unique` 제약조건(아이디 등) 충돌이 발생할 수 있다.
  - **해결책**: 탈퇴 시 식별자 값을 변경(예: `id_deleted_{timestamp}`)하여 충돌을 방지하는 방식을 주로 사용한다.