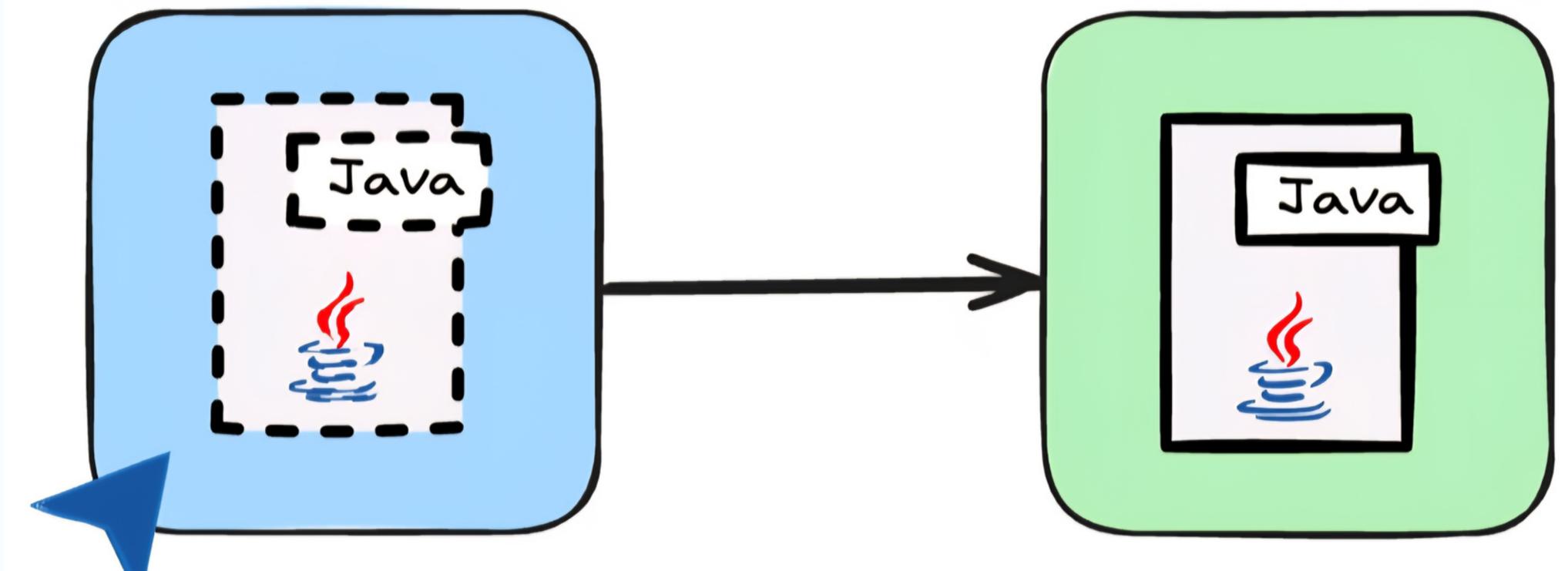




View  
co2plant

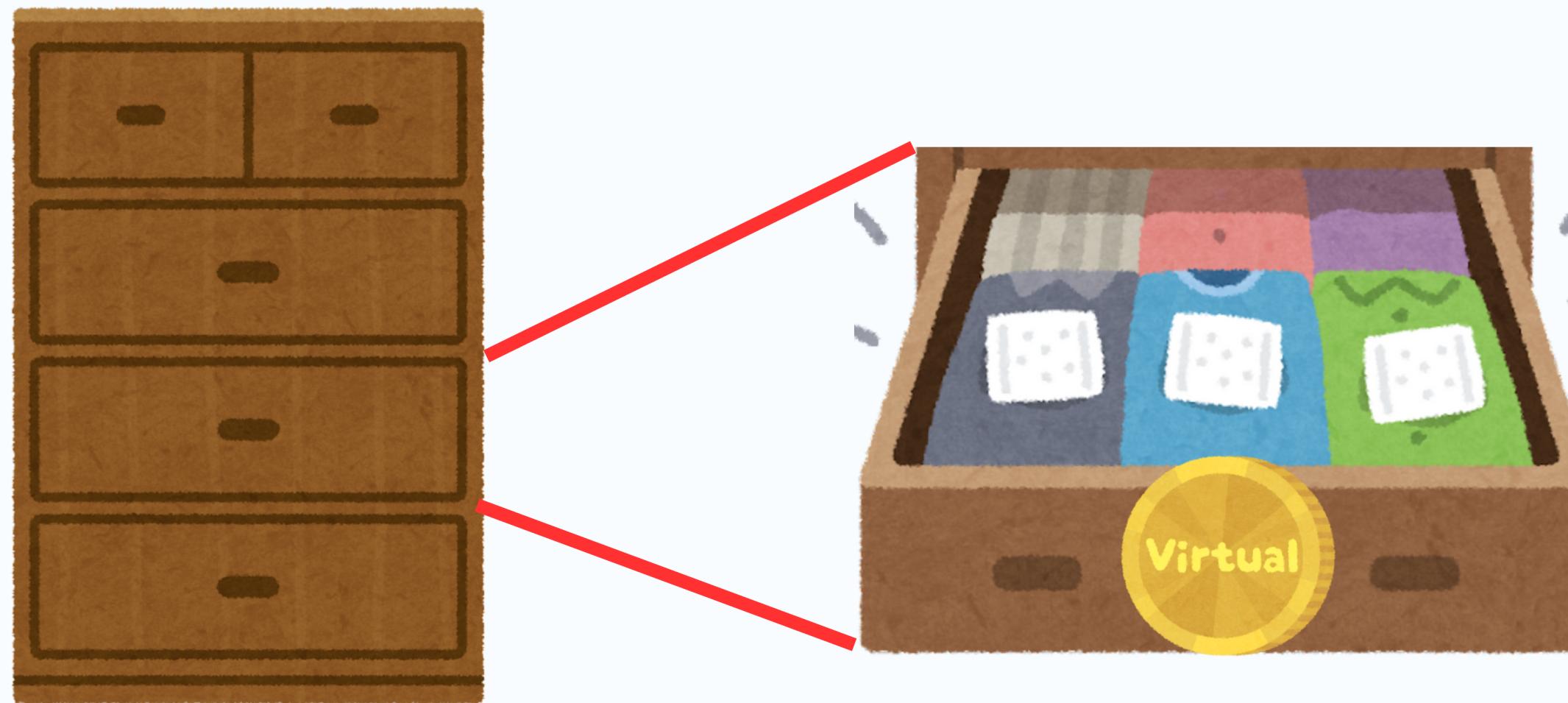
# View란?



데이터베이스에 저장된 데이터를 가상 테이블 형태로 표  
현하는 것

# 특징

# 가상 테이블



쿼리에 따라 동적으로 생성되는 가상테이블  
뷰의 행을 DB에 명시적으로 저장하지 않음

# 중첩 뷰



이미 만들어진 뷰를 바탕으로  
새로운 뷰를 만들 수 있음

# 추상화



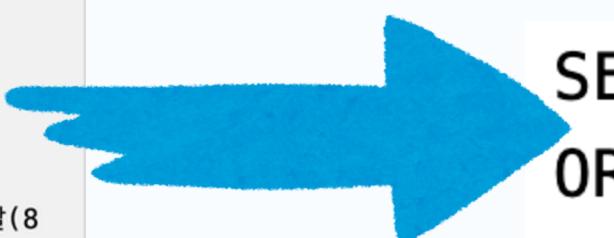
사용자가 뷰가 가상이라는 것을 인지할 필요없이  
테이블처럼 다룰 수 있음

# 왜 사용하는 건가요?

- 추상화 및 단순화
- 보안 강화
- 재사용성
- 논리적 데이터 독립성

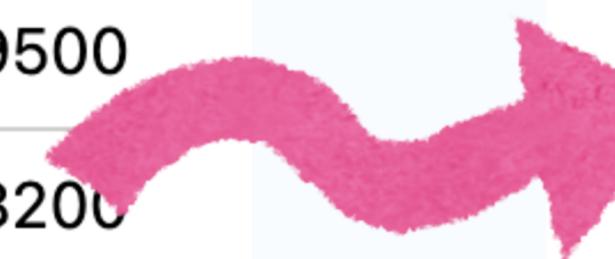
# 추상화와 단순화

```
SELECT
    c.name,                                -- 고객 이름
    c.email,                                 -- 고객 이메일
    SUM(od.price) AS total_purchase_amount, -- 총 구매액
    COUNT(od.detail_id) AS total_items_bought -- 총 구매 상품 수
FROM
    Customers c
JOIN
    Orders o ON c.customer_id = o.customer_id
JOIN
    Order_Details od ON o.order_id = od.order_id
WHERE
    o.order_date >= '2025-08-01' AND o.order_date < '2025-09-01' -- 이번 달(8
월) 조건
GROUP BY
    c.customer_id, c.name, c.email
HAVING
    COUNT(od.detail_id) >= 2 -- 상품 2개 이상 구매 조건
ORDER BY
    total_purchase_amount DESC;
```

 `SELECT * FROM v_monthly_vip_customers  
ORDER BY total_purchase_amount DESC;`

복잡하고 자주 사용되는 쿼리를 뷰로 만들면,  
뷰 이름만으로 복잡한 쿼리를 구현 가능

# 맞춤형 데이터 제공 및 보안강화

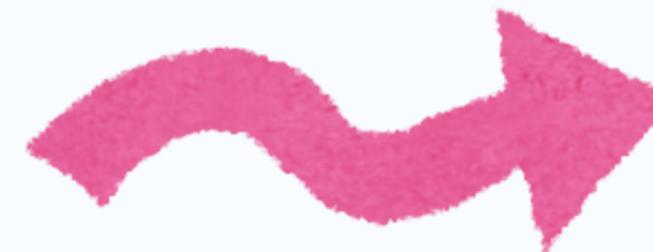
emp_id	emp_name	department	salary		emp_id	emp_name	department
101	박서준	개발팀	9500		101	박서준	개발팀
201	최은우	기획팀	8200		201	최은우	기획팀
202	오기획	기획팀	7000		202	오기획	기획팀

사용자가 필요로 하는 데이터의  
특정 부분만 접근 허용

# 논리적 데이터 독립성

추가됨

emp_id	emp_name	department	salary	validation
101	박서준	개발팀	9500	true
201	최은우	기획팀	8200	true
202	오기획	기획팀	7000	false



emp_id	emp_name	department
101	박서준	개발팀

소스테이블의 논리적 구조가 변경되더라도,  
뷰를 통해 동일한 데이터를 제공 가능

# 사용 예시

# 생성

## 뷰 생성

```
CREATE OR REPLACE VIEW v_developers AS  
SELECT emp_id, emp_name, department, salary  
FROM employees  
WHERE department = '개발팀';
```

## 뷰 조회

```
SELECT * FROM v_developers;
```

## 결과

emp_id	emp_name	department	salary
101	박서준	개발팀	9000
102	김지민	개발팀	7500

1. 'v\_'를 앞에 붙임
2. 원본 테이블 명시
3. 제약조건 설정

# 수정

뷰 행 수정(조건부 가능)

```
UPDATE v_developers  
SET salary = 9500  
WHERE emp_name = '박서준';
```

결과

emp_id	emp_name	department	salary
101	박서준	개발팀	9500

# 삭제

## 뷰 행 삭제(조건부 가능)

```
-- '김지민' 직원 데이터 삭제  
DELETE FROM v_developers  
WHERE emp_name = '김지민';
```

결과

TABLE employees

emp_id	emp_name	department	salary
101	박서준	개발팀	9500
201	최은우	기획팀	8200

# 수정, 삭제 안된다면서요?

1. 수정, 삭제가 불가능한 것은 맞는 말임
2. 단, **복잡한 뷰의 경우에 삭제, 수정이 어려움**
3. 여러 줄의 데이터 중 어떤 원본 데이터를 바꿔야 할지 모르기 때문

뷰의 수정/삭제 조건

뷰와 원본 테이블이 '**대 | 매핑**'이 가능한지 여부

# 삽입

```
-- department 컬럼이 포함되지 않은 뷰 생성
```

```
CREATE VIEW v_dev_names AS  
SELECT emp_id, emp_name FROM employees WHERE  
department = '개발팀';
```

```
-- 삽입 시 오류 발생
```

```
INSERT INTO v_dev_names (emp_id, emp_name) VALUES  
(103, '이신입');
```

department값이 not null,  
뷰에서 포함하지 않음 -> Insert 거부

# 삽입

TABLE employees

emp_id	emp_name	department	salary
101	박서준	개발팀	9500
201	최은우	기획팀	8200
202	오기획	기획팀	7000

VIEW v\_developers

emp_id	emp_name	department	salary
101	박서준	개발팀	9500

테이블에 제대로 입력됐는데,  
뷰에는 뜨지 않음

# 해결방안

```
CREATE VIEW v_developers AS  
SELECT emp_id, emp_name, department, salary  
FROM employees  
WHERE department = '개발팀'  
WITH CHECK OPTION;
```

WITH CHECK OPTION을 사용해  
VIEW 조건에 맞는 삽입만 허용

# 구체화된 뷰 (Materialized View)

# 실제 데이터 저장



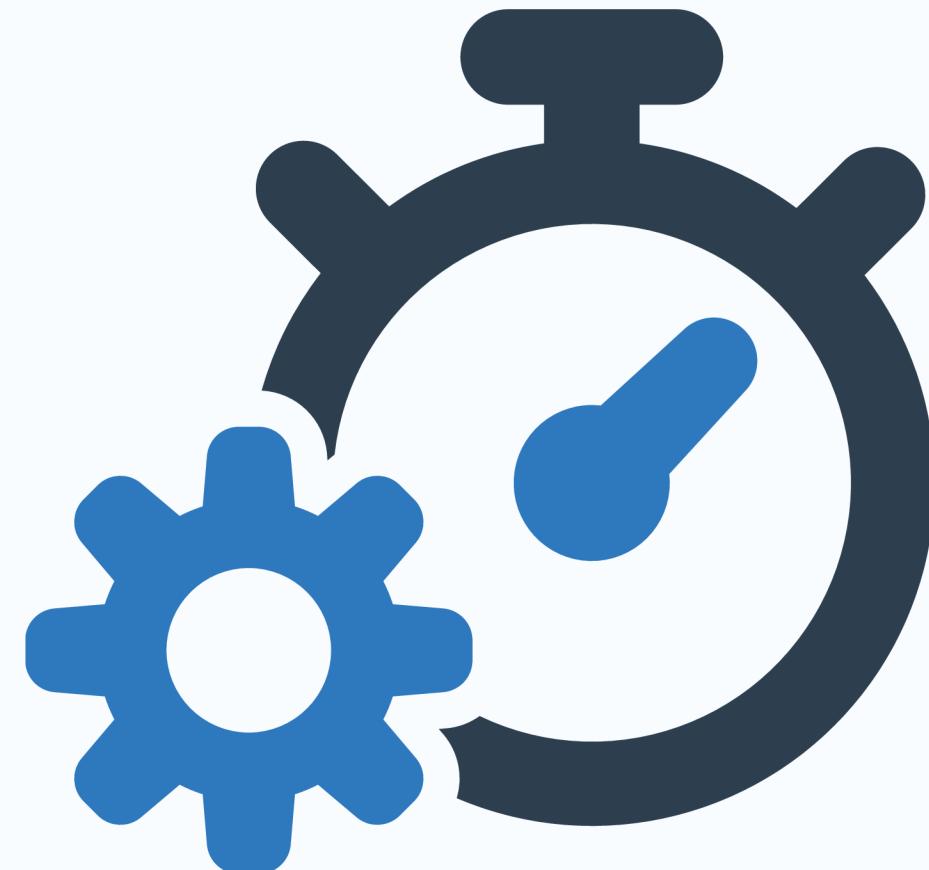
쿼리 결과를 **물리적으로** 저장  
이를 통해 **조회 성능 향상**

# 주기적 갱신



기본 테이블의 데이터 변경에 따라  
주기적으로 갱신함

# 복잡한 쿼리 최적화



복잡한 쿼리의 결과를  
미리 계산 → 물리적 저장 → 응답시간 단축

# 구체화된 뷰 단점 (Materialized View)

# 오버헤드



## 저장공간 오버헤드

물리적인 저장을 하므로,  
추가적 저장공간 필요

## 업데이트 오버헤드(본 유지보수)

데이터 변경을 뷰에 반영  
해야하므로 갱신비용 발생

# 일관성 문제

## Eager/Immediate 갱신

- 기본 데이터가 업데이트될 때 즉시 뷰를 갱신
  - 데이터 최신성을 보장
  - 업데이트 오버헤드 발생

## - \*\*Lazy/Periodic 갱신\*\*

- 뷰가 접근될 때 또는 주기적으로 갱신
  - 최신성을 보장하지 않음
  - 업데이트 오버헤드가 적음



# 사용처

- 대량 데이터 처리 시
- 복잡한 조인 및 집계 연산이 포함된 쿼리에서 성능 필요 시
- 읽기 빈도가 높지만 데이터 불일치를 허용 가능할 때

# INTERVIEW



# 수정, 삭제가 가능하다?



# 조건부로 가능하다

뷰의 수정/삭제 조건

뷰와 원본 테이블이 '**대 | 매핑**'이 가능한지 여부

# 구체화된 뷰는?



# 뷰를 물리적 저장소에 저장

조회 성능 향상 및 복잡한 쿼리를 최적화 하기위한 전략

일관성 문제가 있을 수 있음