요약 정리

✅ 제약조건 (Constraints)

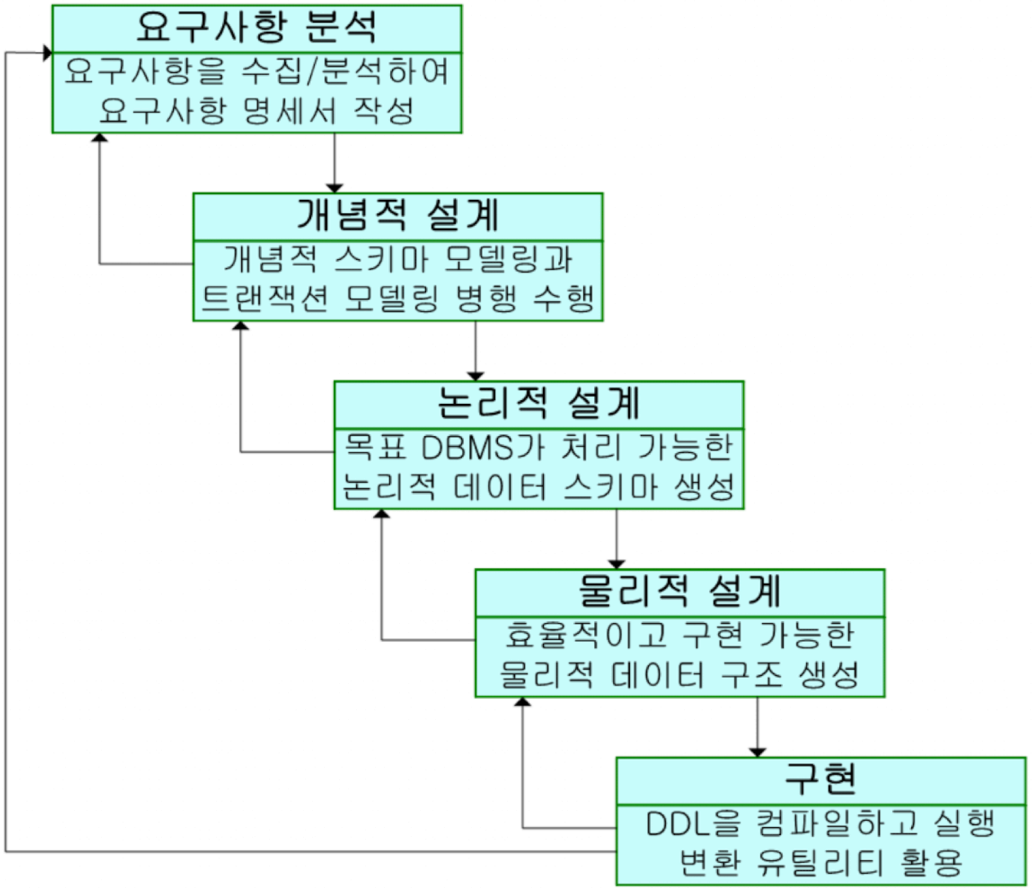
- 원하는 데이터값만을 유지, 보관하기 위해 특정 '컬럼'마다 설정하는 제약
- 목적: 데이터의 무결성 보장 (권한 없는자가 데이터 위변조)

제약조건	설명
NOT NULL	NULL을 허용하지 않음
UNIQUE	중복된 값을 허용하지 않음
PRIMARY KEY	NOT NULL + UNIQUE : 고유키
FOREIGN KEY	다른 테이블 참조
DEFAULT	기본값 설정
CHECK	값의 범위 제한
AUTO_INCREMENT	자동 증가

```
CREATE TABLE 테이블명 (
    컬럼명1 데이터타입 DEFAULT [DEFAULT값] [컬럼 레벨 제약조건],
    컬럼명2 데이터타입 DEFAULT [DEFAULT값] [컬럼 레벨 제약조건],
    컬럼명3 데이터타입 DEFAULT [DEFAULT값] [컬럼 레벨 제약조건],

    [테이블 레벨 제약조건]
)
```

✅ 데이터베이스 설계



- 데이터베이스 설계시 고려 사항

- **무결성(Integrity)** : DB에 대한 변경 연산(갱신, 삽입, 삭제)이 수행된 뒤에도 데이터 값이 만족해야 될 주어진 제약 조건을 항상 만족해야 한다.
- **일관성(Consistency)** : 저장된 두 데이터 값 사이나 특정 질의에 대한 응답 간에 모순성 없이 일치해야 한다.
- **회복(Recovery)** : 시스템에 장애가 발생했을 때 장애 발생 이전의 일관된 상태로 복수할 수 있는 능력을 말한다.
- **보안(Security)** : 권한이 없는 사람이 의도적, 불법적으로 데이터를 변경, 손실, 노출시키는 것을 방지하는 것을 말한다.
- **효율성(Efficiency)** : 응답 시간 단축, 저장 공간 최적화, 시스템 생산성 향상 등이 포함된다.
- **확장성(Extensibility)** : 시스템 운영에 영향을 주지 않으면서 새로운 데이터를 계속적으로 추가시켜 나갈 수 있는 기법을 가져야 한다.

✅정규화된 테이블 설계

- **정규화(Normalization)**: 데이터 **중복 최소화** + **무결성 보장**
- **단계별 특징**
  - 1NF: 컬럼은 원자값만 가진다
  - 2NF: 부분 함수 종속 제거
  - 3NF: 이행 함수 종속 제거
  - BCNF: 후보키 종속성 철저히 제거
- **효과**
  - 데이터 이상 현상(삽입·삭제·갱신 Anomaly) 방지
  - 유지보수, 확장성 향상

✅ 트랜젝션

- 데이터베이스에서 **하나의 논리적 작업 단위**
- 여러 SQL문을 하나의 묶음으로 처리 → **모두 성공**하거나 **모두 실패**해야 함 (All or Nothing)
- 예: 계좌 이체
  1. A 계좌 → 10만원 출금
  2. B 계좌 → 10만원 입금

     → 두 쿼리 중 하나라도 실패하면 전체 취소(Rollback)

✅ 트랜잭션의 4가지 특징 (ACID)

1. **원자성 (Atomicity)**
   - 트랜잭션의 모든 작업은 **전부 수행되거나 전부 수행되지 않아야 함**
   - 부분 성공 없음
2. **일관성 (Consistency)**
   - 트랜잭션 수행 전과 후, 데이터베이스는 항상 **일관된 상태** 유지
3. **격리성 (Isolation)**
   - 동시에 실행되는 트랜잭션들이 서로 영향을 주지 않아야 함

- 격리 수준에 따라 **동시성 문제** 발생 가능 (Dirty Read, Non-repeatable Read, Phantom Read)

4. **지속성 (Durability)**
   - 트랜잭션이 성공적으로 완료되면 결과는 **영구히 저장**됨 (시스템 장애 발생해도 보존)

✅ 트랜잭션 제어 명령어 (TCL)

- **시작**

```
START TRANSACTION;
```

- **커밋 (Commit)** → 성공 확정

```
COMMIT;
```

- **롤백 (Rollback)** → 이전 상태로 되돌림

```
ROLLBACK;
```

- **저장점 설정 (Savepoint)** → 특정 지점까지 되돌림

```
SAVEPOINT sp1;
ROLLBACK TO sp1;
```

✅ 트랜잭션 격리 수준 (Isolation Level)

수준	설명	문제점
READ UNCOMMITTED	커밋되지 않은 데이터 읽기 허용	Dirty Read
READ COMMITTED	커밋된 데이터만 읽기	Non-repeatable Read
REPEATABLE READ	같은 쿼리 반복 시 결과 동일	Phantom Read
SERIALIZABLE	직렬 실행처럼 동작	성능 저하

💡 **TIP**

- 실무에서는 보통 **READ COMMITTED** 또는 **REPEATABLE READ** 사용

- 은행/금융 시스템 같은 중요한 곳은 **SERIALIZABLE** 고려

👉 이렇게 정리하면 트랜잭션 개념부터 **ACID → TCL → Isolation Level** 흐름이 잘 잡히고, 노션에 정리하기 딱 좋아요.

원해요? 제가 이걸 **실습 SQL 예제**(예: 계좌이체 시뮬레이션)까지 추가해서 정리해드릴까요?

✅ 실제 쿼리 구현 예제

📌 테이블 생성

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50) NOT NULL,
    department_id INT,
    hire_date DATE
);
```

📌 데이터 삽입

```
INSERT INTO employees (emp_id, emp_name, department_id, hire_date)
VALUES (1, '홍길동', 10, '2025-09-01');
```

📌 데이터 조회

```
SELECT emp_name, hire_date
FROM employees
WHERE department_id = 10;
```

📌 데이터 수정

```
UPDATE employees
SET department_id = 20
```

```
WHERE emp_id = 1;
```

📌 데이터 삭제

```
DELETE FROM employees
WHERE emp_id = 1;
```

✅ ALTER TABLE (테이블 구조 변경)

- 📍 **컬럼 추가**

```
ALTER TABLE employees
ADD email VARCHAR(100);
```

- 📍 **컬럼 삭제**

```
ALTER TABLE employees
DROP COLUMN email;
```

- 📍 **컬럼 타입 변경**

```
ALTER TABLE employees
MODIFY emp_name VARCHAR(100);
```

- 📍 **컬럼 이름 변경**

```
ALTER TABLE employees
RENAME COLUMN emp_name TO full_name;
```

- 📍 **제약조건 추가**

```
ALTER TABLE employees
ADD CONSTRAINT fk_dept
FOREIGN KEY (department_id) REFERENCES departments(dept_id);
```

- 📍 **제약조건 삭제**

```
ALTER TABLE employees
DROP FOREIGN KEY fk_dept;
```

💡 **TIP**

- 설계 단계에서 정규화를 적용하면 **데이터 품질↑, 유지보수 비용↓**
- ALTER TABLE은 **DDL(데이터 정의어)** → 실행 시 자동 커밋!
