<3과목 데이터베이스 구축>

1. 꼭 알아야 할 키워드 = \_\_\_\_\_(밑줄)

2. # = 두음 암기 or 한 칸 띄어 쓴 건 산출물

3. 시나공 + 수제비 정리 (페이지 참고)

4. “Ctrl+F” 탐색 → 제목 활용하기

**데이터베이스 설계 ★★**

**p.298**

**1) 데이터베이스 설계 시 고려사항**

- 무결성, 일관성, 회복, 보안, 효율성, 데이터베이스 확장

**2) 데이터베이스 설계 순서 ★★ \_\_ 20년 1, 2회 기출문제**

|  |  |
| --- | --- |
| **순서** | **설명** |
| **요**구 조건 분석 | 요구 조건 명세서 작성 |
| **개**념적 설계  (정보 모델링, 개념화) | 독립적인 개념 스키마 모델링, 트랜잭션 모델링  # E-R 다이어그램 모델 **★** |
| **논**리적 설계  (데이터 모델링) | 목표 DBMS에 맞는(종속적인) 논리 스키마 설계  # 트랜잭션 인터페이스 설계, 테이블 설계(RDB),  논리적 데이터베이스 구조로 매핑(Mapping),  스키마의 평가 및 정제 **★** |
| **물**리적 설계  (데이터 구조화) | 목표 DBMS에 맞는(종속적인) 물리적 구조의 데이터로 변환  # 저장 레코드, 접근 경로 설계 **★** |
| **구**현 | 목표 DBMS의 DDL(데이터 정의어)로 데이터베이스 생성,  트랜잭션 작성 |

#**요개논물구**

**데이터 모델 ★★**

**p.302, 3-59**

**1) 데이터 모델의 구성 요소**

- **개**체(Entity): 데이터베이스에 표현하려는 것으로 사람이 생각하는 개념이나 정보 단위 같은 현실 세계의 대상체

- **속**성(Attribute): 데이터의 가장 작은 논리적 단위로서 파일 구조상의 데이터 항목 또는 데이터 필드에 해당

- **관**계(Relationship): 개체 간의 관계 또는 속성 간의 논리적인 연결을 의미

#**개속관**

**2) 개념적 데이터 모델**

- 현실 세계에 대한 인간의 이해를 돕기 위해 현실 세계에 대한 인식을 추상적 개념으로 표현하는 과정  
# E-R(Entity-Relation) 모델

**3) 논리적 데이터 모델 \_\_ 3-62**

- 개념적 모델링 과정에서 얻은 개념적 구조를 컴퓨터가 이해하고 처리할 수 있는 컴퓨터 세계의 환경에 맞도록 변환하는 과정

- 단순히 데이터 모델이라고 하면 논리적 데이터 모델을 의미  
# 관계 모델, 계층 모델, 네트워크 모델

**4) 데이터 모델에 표시할 요소 ★**

- **구**조(Structure): 논리적인 개체 타입들 간의 관계, 데이터 구조 및 정적 성질을 표현

- **연**산(Operation): 실제 데이터를 처리하는 작업에 대한 명세로, 조작하는 기본 도구

- **제**약 조건(Constraint): DB에 저장될 수 있는 실제 데이터의 논리적인 제약 조건  
#**구연제**

**개체(Entity) ★**

**p.305**

**1) 개체의 정의 및 특징**

- 실세계에 독립적으로 존재하는 유형, 무형의 정보로 서로 연관된 몇 개의 속성으로 구성됨

- 데이터베이스에 표현하려는 것으로 사람이 생각하는 개념이나 정보 단위 같은 현실 세계의 대상체

- 독립적으로 존재하거나 그 자체로서도 구별 가능

- 유일한 식별자(Unique Identifier)에 의해 식별 가능

- 다른 개체와 하나 이상의 관계(Relationship)가 있음

**2) 개체 선정 방법**

- 실제 업무를 담당하고 있는 담당자와 인터뷰를 함

- 실제 업무에 사용되고 있는 장부와 전표를 이용

- 자료 흐름도(DFD; Data Flow Diagram)를 통해 업무 분석을 수행했을 경우 자료 흐름도의 자료 저장소(Data Store)를 이용함

- BPR(Business Process Reengineering, 업무 프로세스 재설계)에 의해 업무를 재정의한 경우 관련 개체를 찾음

**3) 개체명 지정 방법**

- 일반적으로 해당 업무에서 사용하는 용어로 지정

- 약어 사용은 되도록 제한

- 가능하면 단수 명사 사용

- 모든 개체명은 유일해야 함

- 가능하면 개체가 생성되는 의미에 따라 이름 부여  
ex) 교수, 고객, 주문, 도시 등

**속성(Attribute) ★**

**p.307**

**1) 속성의 정의 및 특징**

- 데이터베이스를 구성하는 가장 작은 논리적 단위

- 파일 구조상의 데이터 항목 또는 데이터 필드

- 개체를 구성하는 항목 및 개체의 특성을 기술

- **속성의 수**를 디그리(Degree) 또는 차수라고 함 ★

cf) **튜플(Tuple)의 수**는 카디널리티(Cardinality) ★

**2) 속성의 특성에 따른 분류**

|  |  |
| --- | --- |
| **종류** | **설명** |
| **기**본 속성  (Basic Attribute) | 업무 분석을 통해 정의한 속성  ex) 자동차명, 제조일, 연비 |
| **설**계 속성  (Designed Attribute) | 원래 업무상 존재하지 않고 설계 과정에서 도출해낸 속성  ex) 자동차 코드 |
| **파**생 속성  (Derived Attribute) | 다른 속성으로부터 영향을 받아 발생하는 속성  ex) 계산 값 |

#**기설파**

**3) 개체 구성 방식에 따른 분류**

|  |  |
| --- | --- |
| **종류** | **설명** |
| **기본 키** 속성  (Primary Key Attribute) | 개체를 식별할 수 있는 속성 |
| **외래 키** 속성  (Foreign Key Attribute) | 다른 개체와의 관계에서 포함된 속성 |
| 일반 속성 | 개체에 포함되어 있고 기본 키, 외래 키에 포함되지 않은 속성 |

**4) 속성명 지정 원칙**

- 해당 업무에서 사용하는 용어 지정

- 서술형으로 지정하지 않음

- 가급적 약어의 사용은 제한

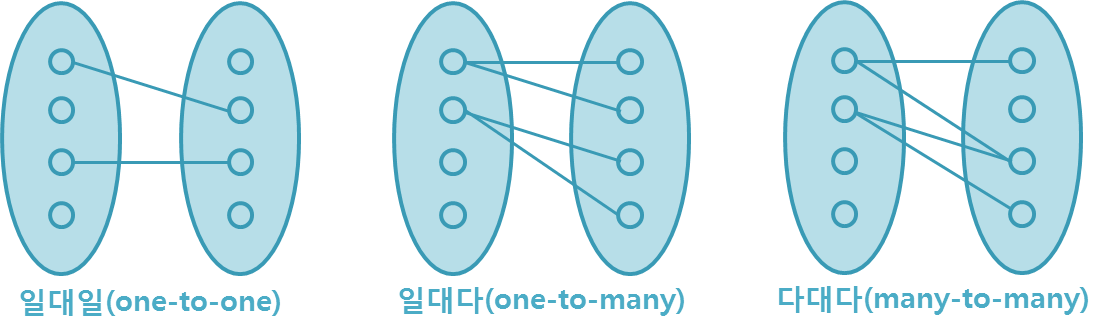
- 개체명은 속성명으로 사용할 수 없음

- 개체에서 유일하게 식별 가능하도록 지정

**관계(Relationship) ★**

**p.310**

**1) 관계의 형태**



- 일 대 일(1:1): 개체 집합 A의 각 원소가 개체 집합 B의 원소 한 개와 대응하는 관계

- 일 대 다(1:N): 개체 집합 A의 각 원소는 개체 집합 B의 원소 여러 개와 대응하고 있지만, 개체 집합 B의 각 원소는 개체 집합 A의 원소 한 개와 대응하는 관계

- 다 대 다(N:M): 개체 집합 A의 각 원소는 개체 집합 B의 원소 여러 개와 대응하고, 개체 집합 B의 각 원소도 개체 집합 A의 원소 여러 개와 대응하는 관계

**2) 관계의 종류**

- **종**속(Dependant) 관계, **중**복(Redundant) 관계, **재**귀(Recursive) 관계, **배**타(Exclusive) 관계

#**종중재배**

**식별자(Identifier) ★**

**p.313**

- 하나의 개체 내에서 각각의 인스턴스를 유일(Unique)하게 구분할 수 있는 구분자

- 모든 개체는 한 개 이상의 식별자를 반드시 가져야 함

|  |  |  |
| --- | --- | --- |
| **분류** | **식별자** | **설명** |
| 대표성 여부 | 주 식별자  (Primary Identifier) | 개체를 대표하는 유일한 식별자,  하나의 개체에 한 개만 존재 ★ |
| 보조 식별자  (Alternate Identifier) | 주 식별자를 대신해 개체를 식별할 수 있는 속성,  하나의 개체에 한 개 이상이 존재 |
| 스스로 생성 여부 | 내부 식별자  (Internal Identifier) | 개체 내에서 스스로 만들어지는 식별자 |
| 외부 식별자  (Foreign Identifier) | 다른 개체와의 관계(Relationship)에 의해 외부 개체의 식별자를 가져와 사용하는 식별자,  자신의 개체에서 다른 개체를 찾아가는 연결자 역할 |
| 단일 속성 여부 | 단일 식별자  (Single Identifier) | 주 식별자가 한 가지 속성으로만 구성된 식별자 |
| 복합 식별자  (Composite Identifier) | 주 식별자가 두 개 이상의 속성으로 구성된 식별자 |
| 대체 여부 | 원조 식별자  (Original Identifier) | 업무에 의해 만들어지는 가공되지 않은 원래의 식별자  (본질 식별자) |
| 대리 식별자  (Surrogate Identifier) | 주 식별자의 속성이 두 개 이상인 경우 속성들을  하나의 속성으로 묶어 사용하는 식별자  (인조 식별자) ★ |

**E-R(개체-관계) 모델 ★★**

**p.319, 3-60**

**1) 개요**

- 개념적 데이터 모델의 가장 대표적인 것

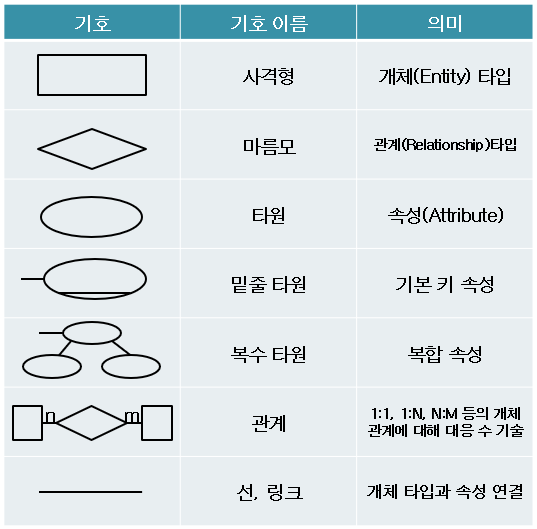
- 1976년 피터 첸(Peter Chen)에 의해 제안되고 기본적인 구성 요소가 정립됨

- 데이터를 **개**체(Entity), **속**성(AttributE), **관**계(Relationship)으로 묘사

- 특정 DBMS를 고려한 것은 아님

- E-R 다이어그램으로 1:1, 1:N, N:M 등의 관계 유형을 제한 없이 나타냄

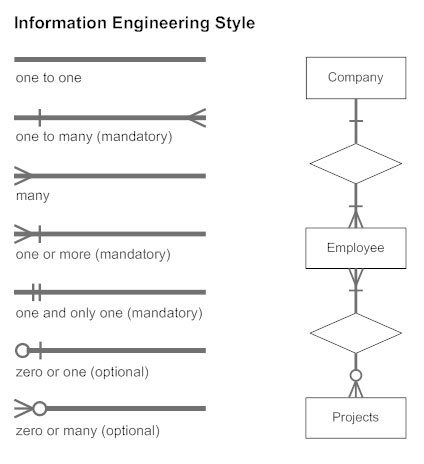
**2) 피터 첸 표기법 ★ \_\_ 20년 1, 2회 기출문제**



이중 타원 다중 값 속성(복합속성)

**3) 정보 공학 표기법(Information Engineering Notation, 크로우즈 핏)**

- 1981년 클리프 핀켈쉬타인(Clive Finkelstein)과 제임스 마틴(James Maritn)이 공동 개발



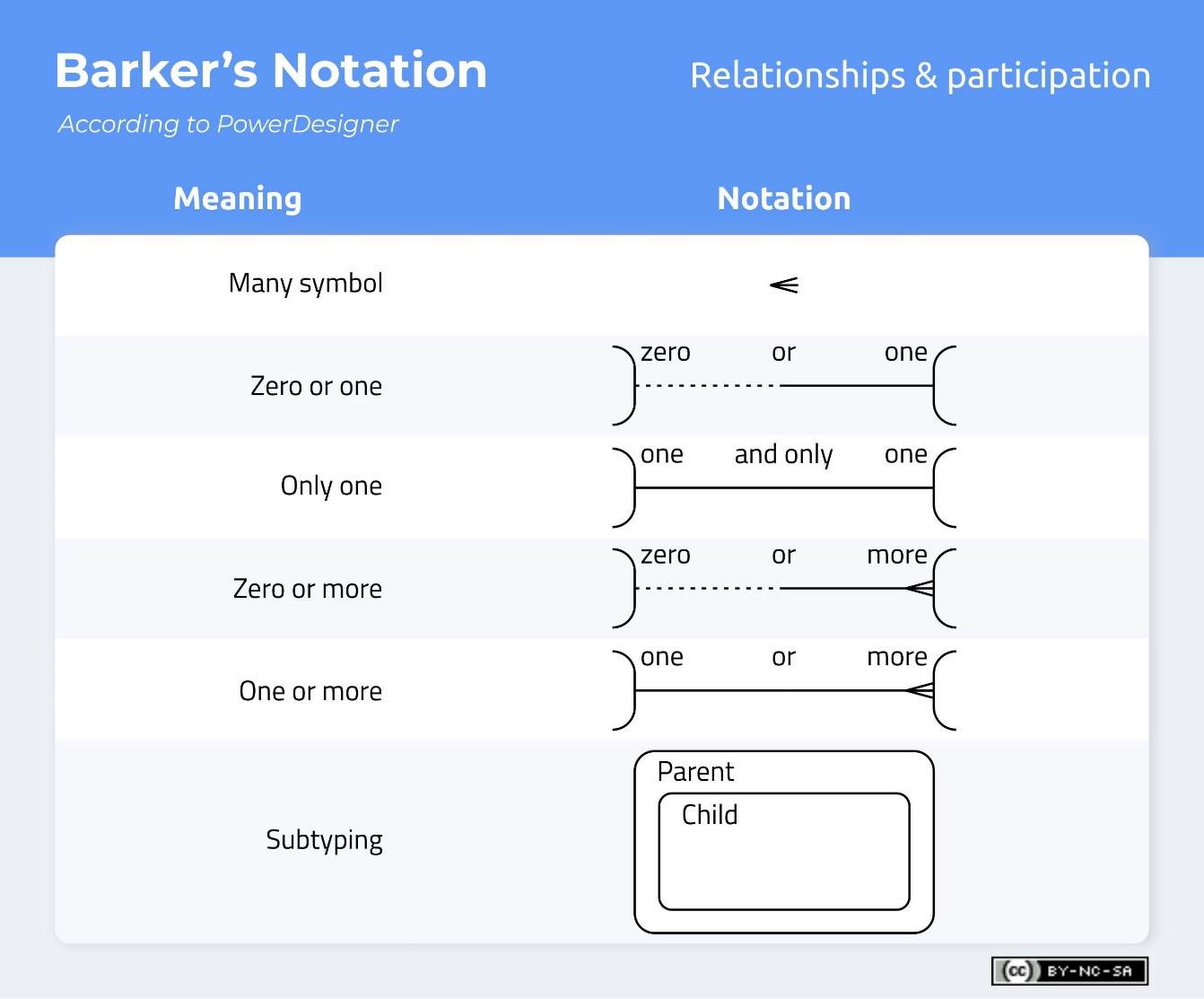
|  |  |
| --- | --- |
| **기호** | **의미** |
| I | 필수(Mandatory) |
| O | 선택적(Optional) |
| < | 다중(Multiple) |

- 실선은 1개를 의미, 까마귀 발은 N개를 의미함

- 원형 표시는 선택적 의미를 지니는데 관계가 있을 수도, 없을 수도 있다는 것

**4) 바커 표기법(Barker Notation)**

- 영국 컨설팅 회사 CACI에서 개발, 리차드 바커(Richard Barker)에 의해 정립



|  |  |
| --- | --- |
| **기호** | **의미** |
|  | 필수(Mandatory) |
| ----------- | 선택적(Optional) |
| < | 다중(Multiple) |

**관계형 데이터 모델 ★★**

**p.322, 3-28**

**1) 개요 ★**

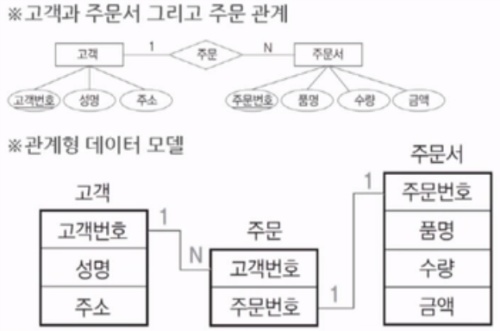
- 2차원적인 표(Table)를 이용해 데이터 상호 관계를 정의하는 DB 구조

- 기본 키(Primary Key)와 이를 참조하는 외래 키(Foreign Key)로 데이터 간의 관계를 표현

- 계층 모델과 망 모델의 복잡한 구조를 단순화시킨 모델

- 관계형 모델의 대표적인 언어는 SQL이고 1:1, 1:N, N:M 관계를 자유롭게 표현

**2) 관계형 데이터 모델(Relational Data Model)의 구성**



- <고객> 테이블에 있는 ‘고객번호’와 <주문서>테이블에 있는 ‘주문번호’는 “기본 키”

- <주문> 테이블에 있는 ‘고객번호’, ‘주문번호’는 기본 키를 참조하고 있는 “외래 키”

- <고객> 테이블과 <주문> 테이블의 관계는 1:N  
# 즉 한 명의 고객은 여러 개의 주문을 신청할 수 있음

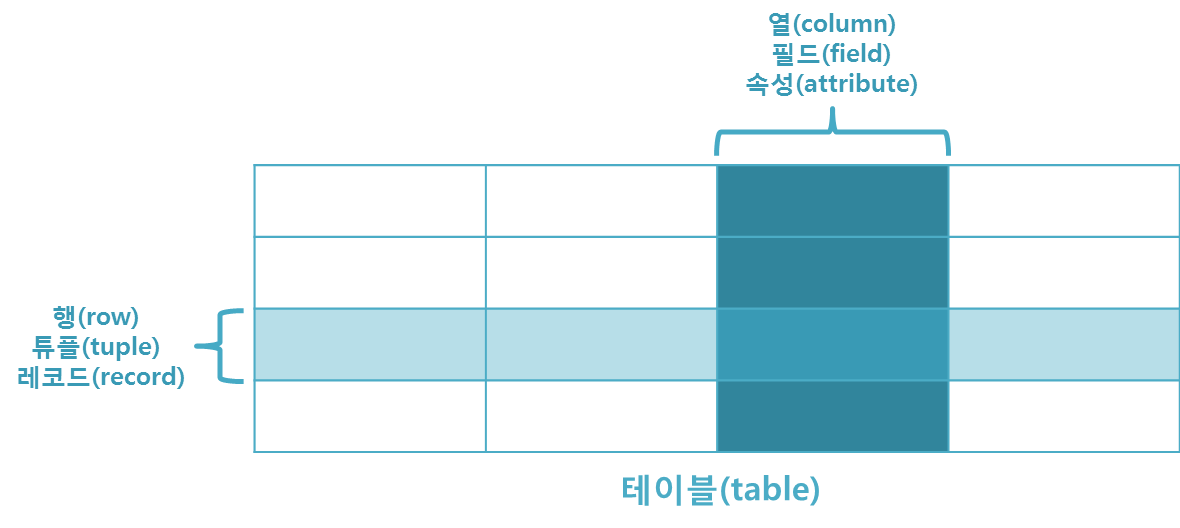
- <주문서> 테이블과 <주문> 테이블의 관계는 1:1  
# 즉 주문서는 주문번호 1개에 대한 정보만을 가짐

**관계형 데이터베이스의 구조 ★★**

**p.325, 3-51**

**1) 관계형 데이터베이스의 Relation 구조 ★★**

- 1970년 IBM에 근무하던 코드(E. F. Codd)에 의해 처음 제안됨



▶ 튜플(Tuple), 행(Row), 레코드(Record)

-속성의 모임으로 구성됨

-파일 구조상 레코드(실제 데이터)와 같은 의미

-**튜플의 수** = 카디널리티(Cardinality) 또는 기수, 대응수 **★**

▶ 속성(Attribute), 열(Column), 필드(Field)

-데이터베이스를 구성하는 가장 작은 논리적 단위

-파일 구조상의 데이터 항목 또는 데이터 필드에 해당

-개체의 특성을 기술

-**속성의 수** = 디그리(Degree) 또는 차수 **★**

▶ 도메인(Domain) **★ \_\_ 20년 1, 2, 3회 기출문제**

-하나의 속성(Attribute, 애트리뷰트)이 가질 수 있는 같은 타입 원자(Atomic)값들의 집합ex) 성별 속성(Attribute)의 도메인은 ‘남’, ‘여’로 그 외의 값은 입력될 수 없음(일반적)

**2) 릴레이션의 특징 ★ \_\_ 20년 3회 기출문제**

- 한 릴레이션(테이블)에 포함된 튜플(행)들은 모두 상이함, 즉 서로 다른 값을 갖음

- 한 릴레이션(테이블)에 포함된 튜플(행) 사이에는 순서가 없음

- 릴레이션 스키마를 구성하는 애트리뷰트(열) 간의 순서는 중요하지 않음

- 각 애트리뷰트는 식별을 위해 릴레이션 내에서 유일한 이름을 가짐, 그러나 그에 해당하는 도메인(애트리뷰트를 구성하는 값)에는 동일한 값이 있을 수 있음  
ex) ‘학년’ 속성에는 1, 2, 3, 4 값이 중복될 수 있음

- 애트리뷰트는 논리적으로 더 이상 쪼갤 수 없는 원자 값으로 저장함

▶ **정리**

1. **튜플**은 서로 상이한 값을 갖고, 순서가 없음

2. **애트리뷰트**는 원자 값을 가지고, 순서가 중요하지 않으며, 유일한 이름을 가짐

**키(Key) ★★**

**p.328**

- 데이터베이스에서 튜플들을 서로 구분할 수 있는 기준이 되는 속성(Attribute)

**1) 후보키(Candidate Key) ★ \_\_ 20년 1, 2회 기출문제**

- 튜플을 유일하게 식별하기 위해 사용하는 속성들의 부분집합, 즉 기본키로 사용할 수 있는 속성들, 모든 릴레이션에는 반드시 하나 이상의 후보키가 존재

- 릴레이션에 있는 모든 튜플에 대해 **유일성**과 **최소성**을 만족시켜야 함 **★**

▶ **유일성**(Unique): 하나의 키 값으로 하나의 튜플만을 유일하게 식별할 수 있어야 함

▶ **최소성**(Minimality): 모든 레코드들을 유일하게 식별하는 데 꼭 필요한 속성으로만 구성되어야 함

**2) 기본키(Primary Key)**

- 후보키 중에서 특별히 선정된 주키(Main Key)로, 중복된 값과 NULL값을 가질 수 없음

- 후보키의 성질인 유일성과 최소성을 가지며 튜플을 식별하기 위해 반드시 필요한 키

**3) 대체키(Alternate Key)**

- 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키를 의미

**4) 슈퍼키(Super Key) ★**

- 한 릴레이션 내에 있는 속성들의 집합으로 구성된 키

- 모든 튜플에 대해 유일성은 만족시키지만, 최소성은 만족시키지 못함

**5) 외래키(Foreign Key) ★ \_\_ 20년 1, 2회 기출문제**

- 다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합

- 참조되는 릴레이션의 기본키와 대응돼 릴레이션 간의 참조 관계를 표현

**무결성(Integrity) ★★**

**p.331, 3-94**

- 데이터베이스에 저장된 데이터 값과 그것이 표현하는 현실 세계의 실제 값이 일치하는 정확성을 의미

**1) 개체 무결성(Entity Integrity, 실체 무결성) ★ \_\_ 20년 1, 2, 3회 기출문제**

- 테이블의 기본키를 구성하는 어떤 속성(Attribute)도 널(NULL)값이나 중복 값을 가질 수 없음

- 기본키의 속성 값이 널(NULL)값이 아닌 원자 값을 갖는 성질

**2) 도메인 무결성(Domain Integrity, 영역 무결성)**

- 릴레이션 내의 튜플들이 각 속성(Attribute)의 도메인에 지정된 값 만을 가져야 함

**3) 참조 무결성(Referential Integrity) ★**

- 외래키 값은 NULL이거나 참조 릴레이션의 기본키 값과 동일해야 함

- 릴레이션은 참조할 수 없는 외래키 값을 가질 수 없다는 규정

**4) 사용자 정의 무결성(User-Defined Integrity)**

- 속성 값들이 사용자가 정의한 제약 조건에 만족해야 함

**5) 데이터 무결성 강화**

- 애플리케이션: 데이터 생성, 수정, 삭제 시 무결성 조건을 검증하는 코드를 데이터를 조작하는 프로그램 내에 추가

- 데이터베이스 트리거: 트리거 이벤트에 무결성 조건을 실행하는 절차형 SQL을 추가

- 제약 조건: 데이터베이스에 제약 조건을 설정해 무결성을 유지

**관계대수 및 관계해석 ★★★**

**p.334, 3-52**

**1) 관계대수 ★★ \_\_ 20년 1, 2, 3회 기출문제**

- 관계형 데이터베이스에서 원하는 정보와 그 정보를 검색하기 위해서 어떻게(How) 유도하는가를 기술하는 절차적인 언어

▶ 순수관계 연산자

|  |  |  |
| --- | --- | --- |
| **연산자** | **기호** | **의미** |
| **Sel**ect  (선택) | σ | 조건(Predicate)을 만족하는 튜플들의 부분 집합  (수평 연산) **★** |
| **P**roject  (추출) | π | 속성들의 부분 집합, 중복은 제거됨  (수직 연산) **★** |
| **Jo**in  (조인) | ▷◁  ex) R ▷◁ S | 두 개의 릴레이션이 공통으로 가지고 있는 속성을 이용하여 두 개의 릴레이션을 하나로 합쳐서 새로운 릴레이션을 만드는 연산 **★**  두 개의 릴레이션의 연관된 튜플들을 결합 |
| **D**ivision  (나누기) | ÷  ex) R ÷ S | R릴레이션에서 S릴레이션의 속성 도메인 값과 일치하는 R릴레이션의 튜플들을 찾아내는 연산 **★** |

#**셀프조디**

▶ 일반집합 연산자

|  |  |  |
| --- | --- | --- |
| **연산자** | **기호** | **의미** |
| Union  (**합**집합) | **∪**  ex) R**∪**S | 두 개의 릴레이션의 합이 추출되고, 중복은 제거됨 |
| Intersection  (**교**집합) | **∩**  ex) R**∩**S | R릴레이션과 S릴레이션의 중복되는 값들만 추출 |
| Difference  (**차**집합) | **—**  ex) R - S | R릴레이션에서 S릴레이션에 중복되지 않는 값들만 추출 |
| **Car**tesian Product  (교차곱) | **Х**  ex) R x S | 두 릴레이션의 가능한 모든 튜플들의 집합,  차수(Degree)는 더하고, 카디널리티(Cardinality)는 곱해서 값을 구함 **★** |

#**합교차카**

**2) 관계해석(Relational Calculus) \_\_ 3-54**

- 관계 데이터 모델의 제안자인 코드(E. F. Codd)가 수학의 Predicate Calculus(술어 해석)에 기반을 두고 관계 데이터베이스를 위해 제안

- 원하는 정보가 무엇(What)이라는 것만 정의하는 비절차적 특성

- 튜플 관계해석, 도메인 관계해석

- 기본적으로 관계해석과 관계대수는 관계 데이터베이스를 처리하는 기능과 능력면에서 동등

- 관계대수로 표현한 식은 관계해석으로 표현할 수 있음

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **구성요소** | **기호** | **설명** |
| 연산자 | OR 연산 | V | 원자식 간 “또는”이라는 관계로 연결 |
| AND 연산 | ∧ | 원자식 간 “그리고”라는 관계로 연결 |
| NOT 연산 | ㄱ | 원자식에 대해 부정 |
| 정량자 | 전칭 정량자  (Universal Quantifier | ∀ | 모든 가능한 튜플 “For All”  # All의 ‘A’를 뒤집은 형태 **★** |
| 존재 정량자  (Existential Quantifier) | ∃ | 어떤 튜플 하나라도 존재 “There Exists”  # Exists의 ‘E’를 뒤집은 형태 **★** |

**3) 관계대수와 관계해석 비교 ★**

|  |  |  |
| --- | --- | --- |
| **구분** | **관계대수** | **관계해석** |
| 특징 | 절차적 언어(순서 명시) | 비절차적 언어(계산 수식의 유연적 사용),  프레디킷 해석(Predicate Calculus) 기반 |
| 목적 | 어떻게 유도하는가?(How) | 무엇을 얻을 것인가?(What) |
| 종류 | 순수관계 연산자, 일반집합 연산자 | 튜플 관계 해석, 도메인 관계 해석 |

**정규화(Normalization), 반정규화(Denormalization) ★★**

**p.341, 3-63, 3-100**

- 하나의 종속성이 하나의 릴레이션에 표현될 수 있도록 분해해가는 과정

- 데이터베이스의 논리적 설계 단계에서 수행

**1) 정규화의 목적 ★ \_\_ 20년 3회 기출문제**

- 데이터 구조의 안정성 및 무결성을 유지

- 어떠한 릴레이션이라도 데이터베이스 내에서 표현 가능하게 만듦

- 효과적인 검색 알고리즘 생성 가능

- 데이터 중복을 배제해 **이상(Anomaly)**의 발생 방지 및 자료 저장 공간의 최소화

- 개체와 속성의 누락 여부 확인 가능

- 데이터 삽입 시 릴레이션을 재구성할 필요성을 줄임

**2) 이상(Anomaly)의 개념 및 종류 ★ \_\_ 20년 3회 기출문제**

- 정규화를 거치지 않아 데이터베이스 내에 데이터들이 불필요하게 중복되어 릴레이션 조작 시 예기치 못하게 발생하는 곤란한 현상

▶ **삽**입 이상(Insertion Anomaly): 릴레이션에 데이터를 삽입할 때 의도와 상관없이 원하지 않은 값들도 함께 삽입되는 현상

▶ **삭**제 이상(Deletion Anomaly): 릴레이션에서 한 튜플을 삭제할 때 의도와는 상관없는 값들도 함께 삭제되는 현상

▶ **갱**신 이상(Update Anomaly): 릴레이션에서 튜플에 있는 속성 값을 갱신할 때 일부 튜플의 정보만 갱신되어 정보에 모순이 생기는 현상

#**삽삭갱**

**3) 정규화의 원칙**

- 정보의 무손실, 분리의 원칙, 데이터의 중복성 감소

**4) 정규화 과정 ★★ \_\_ 20년 1, 2, 3회 기출문제**

|  |  |
| --- | --- |
| **정규형** | **설명** |
| 1NF  (제1정규형) | 릴레이션에 속한 모든 도메인(Domain)이  **원**자 값(Atomic Value)만으로 되어 있는 정규형 |
| 2NF  (제2정규형) | 릴레이션 R이 1NF고, 기본키가 아닌 모든 속성이 기본키에 대해  완전 함수적 종속을 만족하는, **부**분적 함수 종속을 제거한 정규형 **★** |
| 3NF  (제3정규형) | 릴레이션 R이 2NF고, 기본키가 아닌 모든 속성이 기본키에 대해 \***이**행적 함수 종속 관계를 만족하지 않는 정규형  : A → B이고 B → C일 때 A → C를 만족하는 관계(이행 규칙) **★** |
| BCNF  (Boyce-Codd 정규형) | 릴레이션 R에서 모든 **결**정자가 후보키인 정규형,  모든 BCNF가 종속성을 보존하는 것은 아님  (강한 제3정규형, 보이스/코드 정규형) |
| 4NF  (제4정규형) | 릴레이션 R에 **다**치 종속이 성립하는 경우  R의 모든 속성이 A에 함수적 종속 관계를 만족하는 정규형 |
| 5NF  (제5정규형) | 릴레이션 R의 모든 **조**인 종속이  R의 후보키를 통해서만 성립되는 정규형 |

#**원부이결다조**

**5) 반정규화 개념 ★**

- 시스템의 성능 향상, 개발 및 운영의 편의성 등을 위해 정규화된 데이터 모델을 통합, 중복, 분리하는 과정으로 의도적으로 정규화 원칙을 위배하는 행위 **★★**

- 그러나 데이터의 일관성 및 정합성이 저하될 수 있음

**6) 반정규화 방법 \_\_ 20년 1, 2회 기출문제**

- 테이블 통합: 1:1 관계 | 1:N 관계 | 슈퍼타입/서브타입 {테이블 통합}

- 테이블 분할: 수평 분할, 수직 분할 → 기본키의 유일성 관리가 어려워짐

- 중복 테이블 추가: **집**계 테이블 | **진**행 테이블 | **특**정 부분만을 포함하는 테이블 {추가}  
#**집진특 ★**

- 중복 속성 추가: 자주 사용하는 속성을 하나 더 추가하는 것

**시스템 카탈로그(System Catalog) ★★**

**p.346, 3-32, 3-54**

**1) 시스템 카탈로그의 의미 ★**

- 사용자를 포함해 DBMS에서 지원하는 모든 데이터 객체에 대한 정의나 명세에 관한 정보를 유지 관리하는 시스템 테이블

- 좁은 의미로는 카탈로그를 데이터 사전(Data Dictionary)이라고도 함

- 시스템 카탈로그에 저장된 정보를 메타 데이터(Meta-Data)라고 함

**2) 카탈로그의 특징**

- 일반 이용자도 SQL을 이용해 내용을 검색할 수 있음

- INSERT, DELETE, UPDATE문으로 카탈로그를 갱신할 수 없음 **★**

- DBMS에 따라 상이한 구조를 갖음

- 카탈로그는 DBMS가 스스로 생성하고 유지함

- 사용자가 SQL문을 실행시켜 변화를 주면 시스템이 자동으로 갱신함

**3) 데이터 디렉터리(Data Directory, 사전 관리기)**

- 데이터 사전(Data Dictionary)에 수록된 데이터를 실제로 접근하는 데 필요한 정보를 관리 유지하는 시스템

- 시스템만 접근할 수 있음

cf) 시스템 카탈로그(데이터 사전): 사용자와 시스템 모두 접근할 수 있음

**데이터베이스 저장 공간 설계 ★**

**p.366**

**1) 테이블(Table) \_\_ 3-31**

- 데이터베이스의 가장 기본적인 객체로 행(Row, 튜플), 열(Column, 컬럼)로 구성

- 논리 설계 단계의 개체(Entity)에 대응하는 객체

**2) 클러스터드 인덱스 테이블(Clustered Index Table)**

- 기본키나 인덱스키의 순서에 따라 데이터가 저장되는 테이블

- 일반적인 인덱스를 사용하는 테이블에 비해 접근 경로가 단축됨

**3) 파티셔닝 테이블(Partitioning Table) \_\_ 20년 3회 기출문제**

- 대용량의 테이블을 작은 논리적 단위인 파티션으로 나는 테이블

- 파티션 키를 잘못 구성하면 성능 저하 등의 역효과 초래

|  |  |
| --- | --- |
| **종류** | **설명** |
| **레**인지 파티셔닝  (Range Partitioning) | 지정한 열의 값을 기준으로 분할 (**범위분할**)  ex) 일별, 월별, 분기별 등 |
| **해**시 파티셔닝  (Hash Partitioning) | 해시 함수에 따라 데이터 분할 (**해시분할**) |
| **리**스트 파티셔닝  (List Partitioning) | 미리 정해진 그룹핑 기준에 따라 분할 |
| **컴**포지트 파티셔닝  (Composite Partitioning) | 레인지 파티셔닝 이후 해시 함수를 적용 (**조합분할**)  ex) 범위분할 + 해시분할 |

#**레해리컴**

▶ 파티션의 장점

-**성**능 향상, **가**용성 향상, **백**업 가능, 경**합** 감소  
#**성가백합**

**4) 외부 테이블(External Table)**

- 데이터베이스에서 일반 테이블처럼 이용할 수 있는 외부 파일  
# 데이터 웨어하우스(Data Warehouse), ETL(Extraction, Transformation, Loading)

**5) 임시 테이블(Temporary Table)**

- 트랜잭션이나 세션별로 데이터를 저장하고 처리할 수 있는 테이블

- 임시테이블에 저장된 데이터는 트랜잭션이 종료되면 삭제됨

- 절차적인 처리를 위해 임시로 사용하는 테이블

**6) 컬럼(Column, 열)**

- 가변 길이 데이터 타입: 예상되는 최대 길이로 정의

- 고정 길이 데이터 타입: 최소 길이로 지정

- 소수점 이하 자릿수: 소수점 이하 자릿수는 반올림되어 저장

- 고정 길이 컬럼이고 NOT NULL인 컬럼: 앞 쪽

- 가변 길이 컬럼, NULL값이 많을 것으로 예상되는 컬럼: 뒤 쪽

**7) 테이블스페이스(Tablespace)**

- 테이블이 저장되는 논리적인 영역

- 테이블을 저장하면 논리적으로는 테이블스페이스에 저장되고, 물리적으로는 해당 테이블스페이스와 연관된 데이터 파일(Data File)에 저장됨

▶ 테이블스페이스 설계 시 고려사항

-업무별로 구분해 지정하고, 테이블과 인덱스는 분리해 저장함

-대용량 테이블은 하나의 테이블스페이스에 독립적으로 저장함

-LOB(Large Object) 타입의 데이터는 독립적인 공간으로 지정함

**트랜잭션(Transaction) ★★★**

**p.371, 3-29**

**1) 트랜잭션의 정의 ★ \_\_ 20년 3회 기출문제**

- 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위

- 한꺼번에 모두 수행되어야 할 일련의 연산들

▶ **COMMIT**: 트랜잭션 처리가 정상적으로 종료되어 수행한 변경 내용을 DB에 반영하는 명령어

▶ **ROLLBACK**: 트랜잭션 처리가 비정상으로 종료되어 DB의 일관성이 깨졌을 때 트랜잭션이 행한 모든 변경 작업을 취소하고 이전 상태로 되돌리는 연산

▶ SAVEPOINT(=CHECKPOINT): 트랜잭션 내에서 ROLLBACK할 위치인 저장점을 지정하는 명령어, 여러 개의 SAVEPOINT 지정 가능

\* COMMIT과 ROLLBACK 명령어에 의해 보장 받는 트랜잭션 특징 = **원자성 ★**

**2) 트랜잭션의 특성 ★★ \_\_ 20년 1, 2회 기출문제**

|  |  |
| --- | --- |
| **원리** | **특징** |
| 원자성  (**A**tomicity) | 트랜잭션 연산을 데이터베이스 모두에 반영되든지 아니면 전혀 반영되지 않아야 함(All or Nothing) **★** |
| 일관성  (**C**onsistency) | 트랜잭션이 실행을 성공적으로 완료할 시 일관성 있는 데이터베이스 상태를 유지 |
| 독립성  (**I**solation, 격리성) | 둘 이상 트랜잭션 동시 실행 시 한 개의 트랜잭션만 접근이 가능하여 간섭 불가 |
| 영속성  (**D**urability) | 성공적으로 완료된 트랜잭션 결과는 영구적으로 반영됨 |

#**ACID**

**3) CRUD 매트릭스 \_\_ 3-104**

- Create, Read, Update, Delete, ‘C > D > U > R’의 우선순위 적용

- 테이블, 프로세스에 C, R, U, D가 모두 없는 경우

- 테이블에 C 또는 R이 없는 경우 (프로세스는 하나만 있어도 돌아감)

**인덱스(Index) ★**

**p.375, 2-15, 3-38**

**1) 인덱스의 개념 및 선정기준, 고려사항**

- 데이터 레코드를 빠르게 접근하기 위해 <키 값, 포인터>쌍으로 구성된 데이터 구조

▶ 인덱스 컬럼 선정 **★**

-인덱스 컬럼의 분포도(Selectivity)가 10~15% 이내인 “컬럼”

-가능한 한 수정이 빈번하지 않는 “컬럼”

-ORDER BY, GROUP BY, UNION이 빈번한 “컬럼”

-분포도가 좋은 컬럼은 단독 인덱스로 생성

-인덱스들이 자주 조합되어 사용되는 컬럼은 결합 인덱스로 생성

▶ 설계 시 고려사항 **★**

-새로 추가되는 인덱스는 기존 엑세스 경로에 영향을 미칠 수 있음

-지나치게 많은 인덱스는 오버헤드(Overhead) 발생

-넓은 범위 인덱스 처리 시 오히려 전체 처리보다 많은 오버헤드를 발생시킴

-인덱스만의 추가적인 저장 공간이 필요

-인덱스와 테이블 데이터의 저장 공간이 분리되도록 설계

**2) 인덱스 종류**

- **클러스터드 인덱스**(Clustered Index) / **넌클러스터드 인덱스**(Non-Clustered Index)

- **트리 기반 인덱스**: 인덱스를 저장하는 블록들이 트리 구조를 이루고 있는 것

- **비트맵 인덱스**: 인덱스 컬럼의 데이터를 Bit 값인 0, 1로 변환해 인덱스 키 사용

- **함수 기반 인덱스**: 컬럼에 특정 함수나 수식을 적용해 산출된 값을 사용하는 것

- **비트맵 조인 인덱스**: 다수의 조인된 객체로 구성된 인덱스

- **도메인 인덱스**: 개발자가 필요한 인덱스를 직접 만들어 사용하는 것 (확장형 인덱스)

**뷰(View) ★★**

**p.380, 2-16, 3-36, 3-55**

**1) 뷰의 개요 및 특징**

- 기본 테이블로부터 유도된, 이름을 가지는 가상 테이블로 기본 테이블과 같은 형태의 구조를 사용하며, 조작도 기본 테이블과 거의 같음

- 가상 테이블이기 때문에 물리적으로 구현되어 있지 않지만 사용자에게 있는 것처럼 간주됨 → 저장장치 내에 논리적으로 존재

- 정의된 뷰로 다른 뷰를 정의할 수 있음

- 뷰가 정의된 기본 테이블이나 뷰를 삭제하면 그 테이블이나 뷰를 기초로 정의된 다른 뷰도 자동으로 삭제됨

|  |  |
| --- | --- |
| **속성** | **설명** |
| REPLACE | 뷰가 이미 존재하는 경우 재생성 |
| FORCE | 본 테이블의 존재 여부에 관계 없이 뷰 생성 |
| NOFORCE | 기본 테이블이 존재할 때만 뷰 생성 |
| WITH CHECK OPTION | 서브 쿼리 내의 조건을 만족하는 행만 변경 |
| WITH READ ONLY | 데이터 조작어(DML) 작업 불가 |

**2) 뷰의 장, 단점 ★ \_\_ 20년 1, 2, 3회 기출문제**

▶ 장점

-논리적 데이터 독립성 제공

-접근 제어를 통한 자동 보안 제공

-사용자 데이터 관리 용이

▶ 단점

-독립적인 인덱스를 가질 수 없음

-뷰의 정의를 ALTER로 변경할 수 없음 → DROP하고 새로 CREATE해야 함

-뷰로 구성된 내용에 대한 삽입, 삭제, 갱신, 연산에 제약이 따름

**클러스터(Cluster) ★**

**p.383, 2-16, 3-87**

**1) 클러스터의 개요 및 특징 ★**

- 데이터 저장 시 데이터 액세스 효율을 향상시키기 위해 동일한 성격의 데이터를 동일한 데이터 블록에 저장하는 물리적 저장 방법

- 인덱스의 단점을 해결한 기법 → 분포도(Selectivity)가 넓을수록 오히려 유리함

- 분포도가 넓은 “테이블”의 클러스터링은 저장 공간의 절약이 가능

- 대량의 범위를 자주 액세스(조회)하는 경우 적용

- 인덱스를 사용한 처리 부담이 되는 넓은 분포도에 활용

**2) 클러스터의 선정기준 및 고려사항**

▶ 클러스터 테이블 선정

-수정이 빈번하지 않는 “테이블”

-ORDER BY, GROUP BY, UNION이 빈번한 “테이블”

-처리 범위가 넓어 문제가 발생하는 경우 단일 테이블 클러스터링 사용

-조인이 많아 문제가 발생되는 경우는 다중 테이블 클러스터링 사용

▶ 설계 시 고려사항

-클러스터링 된 테이블은 조회 속도를 향상시켜주지만 입력, 수정, 삭제 시 성능이 저하됨(부하가 증가)

-대용량을 처리하는 트랜잭션은 전체 테이블을 스캔하는 일이 자주 발생하므로 클러스터링을 하지 않는 것이 좋음

-클러스터링 된 테이블에 클러스터드 인덱스를 생성하면 접근 성능이 향상됨

**분산 데이터베이스 설계 ★★**

**p.390, 3-77**

**1) 분산 데이터베이스 정의**

- 논리적으로는 하나의 시스템에 속하지만 물리적으로는 네트워크를 통해 연결된 여러 개의 컴퓨터 사이트(Site)에 분산돼 있는 데이터베이스

**2) 분산 데이터베이스의 구성 요소**

|  |  |
| --- | --- |
| **구성 요소** | **설명** |
| 분산 처리기 | 자체적으로 처리 능력을 가지며,  지리적으로 분산되어 있는 컴퓨터 시스템 |
| 분산 데이터베이스 | 지리적으로 분산되어 있는 데이터 베이스,  해당 지역의 특성에 맞게 구성된 데이터 베이스 |
| 통신 네트워크 | 분산 처리기들을 통신망으로 연결해 논리적으로 하나의 시스템처럼 작동할 수 있도록 하는 통신 네트워크 |

**3) 분산 데이터베이스의 목표 ★★ \_\_ 20년 1, 2, 3회 기출문제**

|  |  |
| --- | --- |
| **목표** | **설명** |
| **위**치 투명성  (Location Transparency) | 데이터베이스의 실제 위치를 알 필요 없이 단지 데이터베이스의 논리적인 명칭만으로 액세스할 수 있음 |
| 중**복** 투명성  (Replication Transparency, 복제 투명성) | 동일 데이터가 여러 곳에 중복되어 있더라도 사용자는 마치 하나의 데이터만 존재하는 것처럼 사용하고, 시스템은 자동으로 여러 자료에 대한 작업을 수행 |
| **병**행 투명성  (Concurrency Transparency) | 다수의 트랜잭션들이 동시에 실현되더라도 그 트랜잭션의 결과는 영향을 받지 않음 |
| **분**할 투명성  (Division Transparency) | 하나의 논리적 릴레이션이 여러 단편으로 분할되어 각 단편의 사본이 여려 시스템에 저장되어 있음을 인식할 필요가 없음 |
| **장**애 투명성  (Failure Transparency) | 트랜잭션, DBMS, 네트워크, 컴퓨터 장애가 발생해도 트랜잭션을 정확하게 처리하고 데이터 무결성을 보장함 |

#**위복병분장**

**4) 분산 데이터베이스의 장, 단점**

|  |  |
| --- | --- |
| **장점** | **단점** |
| - 지역 자치성이 높음  - 자료의 공유성 향상  - 분산 제어 가능  - 시스템 성능 향상  - 중앙 컴퓨터의 장애가 전체 시스템에 영향을 끼치지 않음  - 효용성과 융통성이 높음  - 신뢰성 및 가용성이 높음  - 점진적 시스템 용량 확장이 용이 | - DBMS가 수행할 기능이 복잡  - 데이터베이스 설계가 어려움  - 소프트웨어 개발 비용 증가  - 처리 비용 증가  - 잠재적 오류 증가 (사이트 간의 오류 발생률 높음) → 보안의 어려움 **★** |

**5) 분산 데이터베이스 설계**

- 애플리케이션이나 사용자가 분산되어 저장된 데이터에 접근하게 하는 것을 목적

▶ 분산 설계 방법

-테이블 위치 분산: 테이블을 각기 다른 서버에 분산시켜 배치하는 방법

-**분할**(Fragmentation): 테이블의 데이터를 분할하여 분산시키는 것

-**할당**(Allocation): 동일한 분할을 여러 개의 서버에 생성하는 방법  
# 중복이 없는 할당, 중복이 있는 할당

**데이터베이스 이중화 / 서버 클러스터링 ★**

**p.394, 3-78**

**1) 데이터베이스 이중화(Database Replication)**

- 시스템 오류로 인한 데이터베이스 서비스 중단이나 물리적 손상 발생 시 이를 복구하기 위해 동일한 데이터베이스를 복제해 관리하는 것

**2) 데이터베이스 이중화의 분류**

|  |  |
| --- | --- |
| **기법** | **설명** |
| **Eager** 기법 | 트랜잭션 수행 중 데이터 변경이 발생하면 이중화 된 모든 데이터베이스에 즉신 전달해 변경 내용이 즉시 적용되도록 하는 기법 |
| **Lazy** 기법 | 트랜잭션의 수행이 종료되면 변경 사실을 새로운 트랜잭션에 작성해 각 데이터베이스에 전달되는 기법  → 데이터베이스마다 새로운 트랜잭션이 수행되는 것으로 간주됨 |

**3) 데이터베이스 이중화 구성 방법**

|  |  |
| --- | --- |
| **방법** | **설명** |
| 활동-대기  (Active-Standby) | 한 DB가 활동 상태로 서비스하고 있으면 다른 DB는 대기하고 있다가 활동 DB에 장애가 발생하면 대기 상태에 있던 DB가 자동으로 모든 서비스를 대신 수행  → 구성 방법 및 관리가 쉬워 많은 기업에서 이용함 |
| 활동-활동  (Active-Active) | 두 개의 DB가 서로 다른 서비스를 제공하다가 둘 중 한쪽 DB에 문제가 발생하면 나머지 다른 DB가 서비스를 제공  → 두 DB 모두 처리를 하기 때문에 처리율이 높지만 구성 방법 및 설정이 복잡함 |

**4) 서버 클러스터링(Server Clustering)**

- 두 대 이상의 서버를 하나의 서버처럼 운영하는 기술

**▶** **고가용성 클러스터링**: 하나의 서버에 장애 발생 → 다른 서버가 대신 처리

**▶** **병렬 처리 클러스터링**: 하나의 작업을 여러 개의 서버에 분산해 처리

**데이터베이스 보안 / 스토리지 ★★**

**p.397~407, 3-79**

**1) 데이터베이스 보안의 개요**

- 데이터베이스 일부분 또는 전체에 대해서 권한이 없는 사용자가 액세스하는 것을 금지하기 위해 사용되는 기술

- 데이터베이스 사용자들은 일반적으로 서로 다른 객체에 대해 다른 접근 권리 또는 권한을 가짐

**2) 암호화(Encryption)**

▶ 암호화(Encryption) 과정

-암호화되지 않은 평문을 정보 보호를 위해 암호문으로 바꾸는 과정  
 # 개인키 암호 방식(대칭키), 공개키 암호 방식(비대칭키)

▶ 복호화(Decryption) 과정

-암호문을 원래의 평문으로 바꾸는 과정

**3) 암호화 방식**

|  |  |  |
| --- | --- | --- |
| **방식** | **특징** | **종류** |
| **개인키 암호 방식**  (Private Key Encryption, 비밀키 암호 방식, 대칭키) | 동일한 키로 데이터를 암호화하고 복호화 함,  비밀키는 DB 사용 권한이 있는 사용자만 나눠 가짐 | **DES, AES, SEED, ARIA** |
| **공개키 암호방식**  (Public Key Encryption, 비대칭키) | 데이터를 암호화할 때 사용하는 키(공개키)는 DB 사용자에게 공개하고, 복호화할 때의 키(비밀키)는 관리자가 관리하는 방법 | **RSA** (Rivest Shamir Adleman)  **Diffie Hellman** Algorithm |

**4) 접근통제 \_\_ 5-86**

- 데이터가 저장된 객체와 이를 사용하려는 주체 사이의 정보 흐름을 제한하는 것

- 접근통제 3요소: 접근통제 **정**책, 접근통제 **보**안모델, 접근통제 메**커**니즘  
#**정보커**

▶ 임의 접근통제(DAC; Discretionary Access Control)

-데이터에 접근하는 사용자의 신원에 따라 접근 권한 부여  
# 접근통제 권한=주체

▶ 강제 접근통제(MAC; Mandatory Access Control)

-주체와 객체의 등급을 비교해 접근 권한 부여  
# 접근통제 권한=제3자

**5) 접근통제 정책**

|  |  |
| --- | --- |
| **정책** | **설명** |
| 신분 기반 정책  (**DAC**) | 주체나 그룹의 신분에 근거해 객체의 접근을 제한하는 방법 # IBP(Individual-Based Policy), GBP(Group-Based Policy) |
| 규칙 기반 정책  (**MAC**) | 주체가 갖는 권한에 근거해 객체의 접근을 제한하는 방법 # MLP(Multi-Level Policy), CBP(Compartment-Based Policy) |
| 역할 기반 정책  (**RBAC**) | 주체가 맡은 역할에 근거해 객체의 접근을 제한하는 방법 # 인사 담당자, DBA(Database Administration) |

**6) 접근통제 메커니즘 ★**

▶ 접근통제 목록(**ACL**; Access Control List): 객체를 기준으로 특정 객체에 대해 어떤 주체가 어떤 행위를 할 수 있는지를 기록한 목록

▶ 능력 리스트(**CL**; Capability List): 주체를 기준으로 주체에게 허가된 자원 및 권한을 기록한 목록

▶ 보안 등급(Security Label), 패스워드, 암호화

**7) 접근통제 보안 모델**

- 기밀성 모델: 군사적인 목적으로 개발된 최초의 수학적 모델, 기밀성 보장 최우선  
# 벨라파듈라 모델: No Read UP(기밀성), No Write Down

- 무결성 모델: 불법적인 정보 변경을 방지하기 위해 무결성을 기반으로 개발된 모델  
# 비바 모델: No Read Down, No Write Up(무결성)

- 접근통제 모델: 접근통제 메커니즘을 보안 모델로 발전시킨 것  
# 접근통제 행렬(Access Control Matrix): 행=주체, 열=객체

**8) 데이터베이스 백업 종류 \_\_ 3-88, 5-51**

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **복구 수준** |
| 물리 백업 | 로그 파일 백업 실시 | 완전 복구 |
| 로그 파일 백업 없음 | 백업 시점까지 복구 |
| 논리 백업 | DBMS 유틸리티 |

\* **로그 파일**: 데이터베이스의 상태 변화를 시간의 흐름에 따라 모두 기록한 파일

**9) 스토리지(Storage) \_\_ 3-75, 5-49**

|  |  |  |  |
| --- | --- | --- | --- |
| **종류** | **설명** | **장점** | **단점** |
| **DA**S  (Direct Attached Storage) | 서버와 저장장치를 전용 케이블로 직접 연결하는 방식 | 속도가 빠르고 설치 및 운영이 쉬움, 초기 구축 및 유지보수 비용 저렴 | 파일 공유 불가, 확장성 및 유연성이 떨어짐 |
| **NA**S  (Network Attached Storage) | 서버와 저장장치를 네트워크를 통해 연결하는 방식 | 장소에 구애받지 않고 저장장치에 쉽게 접근, 확장성 및 유연성 우수 | 접속 증가 시 성능 저하 |
| **S**AN  (Storage Area Network) | 서버와 저장장치를 연결하는 전용 네트워크를 별도로 구성한 방식 | 파이버 채널 스위치로 네트워크 구성, 광케이블로 처리속도 빠름, 확장성, 유연성, 가용성 뛰어남 | 설치 비용이 많이 듦 |

#**다나쓰**

**논리 데이터 모델의 물리 데이터 모델 변환 및 품질 검토 ★**

**p.410~418, 2-13**

**1) 일반적인 변환 절차**

▶ 단위 개체를 테이블로 변환 → 속성을 컬럼으로 변환 → UID(Unique Identifier)를 기본 키(Primary Key)로 변환 → 관계를 외래 키(Foreign Key)로 변환 → 컬럼 유형(Type)과 길이(Length) 정의 → 반정규화(De-normalization) 수행

**2) 슈퍼타입/서브타입을 테이블로 변환**

-슈퍼타입 기준 테이블 변환: 서브타입을 슈퍼타입에 통합해 하나의 테이블로 만드는 것

-서브타입 기준 테이블 변환: 슈퍼타입 속성들을 각각의 서브타입에 추가해 서브타입들을 개별적인 테이블로 만드는 것

-개별타입 기준 테이블 변환: 슈퍼타입과 서브타입들을 각각의 개별적인 테이블로 변환하는 것

**3) 물리 데이터 모델 품질 기준 (=논리 데이터 모델 품질 기준) \_\_ 3-70, 3-102**

|  |  |
| --- | --- |
| **기준** | **설명** |
| **정**확성 | 요구사항이나 업무 규칙, 표기법에 따라 정확하게 표현됨 |
| **완**전성 | 데이터 모델의 구성 요소를 누락 없이 정의하고  요구사항이나 업무 영역을 누락 없이 반영함 |
| **준**거성 | 데이터 표준, 표준화 규칙, 법적 요건 등을 정확하게 준수함 |
| **최**신성 | 최근의 이슈나 현행 시스템을 반영함 |
| **일**관성 | 표현상의 일관성을 유지함 |
| **활**용성 | 업무 변화에 따른 데이터 구조의 변경이 최소화될 수 있도록 설계됨 |

#**정완준 최일활**

**SQL 응용 ★★★**

**p.426~4, 3-8**

- 1974년 IBM 연구소에서 개발한 SEQUEL에서 유래함

- 관계대수와 관계해석을 기초로 한 혼합 데이터 언어

**1) SQL(Structured Query Language)의 분류 ★★ \_\_ 20년 1, 2회 기출문제**

▶ **DDL**(Data Define Language, 데이터 정의어) **\_\_ 3-26**

-DOMAIN(**도**메인), SCHEMA(**스**키마), TABLE(**테**이블), VIEW(**뷰**), INDEX(**인**덱스)를 **정의**하거나 **변경** 또는 **삭제**할 때 사용하는 언어

#**도스테뷰인**

|  |  |
| --- | --- |
| **명령어** | **기능** |
| **CREATE** | DOMAIN, SCHEMA, TABLE, VIEW, INDEX 정의  → CREATE DOMAIN, SCHEMA, TABLE, VIEW, INDEX 도스테뷰인명; |
| **ALTER** | TABLE에 대한 정의 변경  → ALTER TABLE 테이블명; |
| **DROP** | DOMAIN, SCHEMA, TABLE, VIEW, INDEX 삭제  → DROP DOMAIN, SCHEMA, TABLE, VIEW, INDEX 도스테뷰인명;  \* **CASCADE**: 참조하는 모든 개체 함께 제거 ★  \***RESTRICTED**: 다른 개체가 제거할 요소를 참조 중이면 제거 취소 |

▶ **DML**(Data Manipulation Language, 데이터 조작어) ★ **\_\_ 3-13, 1, 2회 기출문제**

-데이터베이스 사용자가 응용 프로그램이나 질의어를 통해 저장된 데이터를 실질적으로 처리하는 데 사용하는 언어

|  |  |
| --- | --- |
| **명령어** | **기능** |
| **SELECT** | 테이블에서 조건에 맞는 튜플 검색  → **SELECT FROM** 테이블명 [WHERE 조건]; |
| **INSERT** | 테이블에 새로운 튜플 삽입  → **INSERT INTO** 테이블명 **VALUES** 데이터; |
| **DELETE** | 테이블에서 조건에 맞는 튜플 삭제  → **DELETE FROM** 테이블명 [WHERE 조건]; |
| **UPDATE** | 테이블에서 조건에 맞는 튜플의 내용 갱신(변경)  → **UPDATE** 테이블명 **SET** 속성명=데이터 [WHERE 조건]; |

▶ **DCL**(Data Control Language, 데이터 제어어) **\_\_ 3-15, 20년 1, 2, 3회 기출문제**

-데이터의 무결성, 보안, 회복, 병행수행 제어 ★ 등을 정의하는 데 사용되는 언어

-데이터베이스 관리자(DBA)가 데이터 관리를 목적으로 사용

|  |  |
| --- | --- |
| **명령어** | **기능** |
| **COMMIT** | 명령에 의해 수행된 결과를 실제 물리적 디스크로 저장하고,  데이터베이스 조작 작업이 정상적으로 완료됐음을 알려주는 명령어 |
| **ROLLBACK** | 아직 COMMIT되지 않은 변경된 모든 내용들을 취소하고,  데이터베이스를 이전 상태로 되돌리는 명령어  \*SAVEPOINT: 트랜잭션 내에 ROLLBACK 할 위치인 저장점을 지정하는 명령어 |
| **GRANT** | 데이터베이스 사용자에게 사용 권한 부여  → GRANT 권한 리스트 ON 개체 TO 사용자 [WITH GRANT OPTION];  \*WITH GRANT OPTION: 부여받은 권한을 다른 사용자에게 다시 부여할 수 있는 권한 |
| **REVOKE** | 데이터베이스 사용자의 사용 권한 취소  → REVOKE [GRANT OPTION FOR] 권한 리스트 ON 개체 FROM 사용자 [CASCADE];  \*GRANT OPTION FOR: 다른 사용자에게 권한을 부여할 수 있는 권한을 취소 |

ex) GRANT UPDATE ON 고객(테이블) TO 홍길동 WITH GRANT OPTION;

ex) REVOKE GRANT OPTION FOR UPDATE ON 고객(테이블) FROM 홍길동 CASCADE;

**2) SELECT \_\_ p.444~453 ★ \_\_ 20년 1, 2, 3회 기출문제**

▶ **WHERE**절: 검색할 조건을 기술

▶ **ORDER BY**절: 특정 속성을 기준으로 정렬해 검색할 때 사용  
# ASC(오름차순), DESC(내림차순) – 따로 설정이 없을 때는 기본적으로 ASC 사용

▶ **GROUP BY**절: 특정 속성을 기준으로 그룹화해 검색할 때 사용, 일반적으로 그룹 함수와 함께 사용

▶ **HAVING**절: GROUP BY와 함께 사용되며, 그룹에 대한 조건 지정

\* **DISTINCT**: 중복 튜플 제거 **★**

▶ **집계/그룹함수**: GROUP BY절에 지정된 그룹별로 속성의 값을 집계할 함수를 기술함

|  |  |
| --- | --- |
| **종류** | **설명** |
| **COUNT**(속성명) | 그룹별 튜플 수를 구하는 함수 |
| **SUM**(속성명) | 그룹별 합계를 구하는 함수 |
| **AVG**(속성명) | 그룹별 평균을 구하는 함수 |
| **MAX**(속성명) | 그룹별 최대값을 구하는 함수 |
| **MIN**(속성명) | 그룹별 최소값을 구하는 함수 |
| **STDDEV**(속성명) | 그룹별 표준편차를 구하는 함수 |
| **VARIANCE**(속성명) | 그룹별 분산을 구하는 함수 |
| **ROLLUP**(속성명, 속성명, …) | 인수로 주어진 속성을 대상으로 그룹별 소계를 구하는 함수,  속성의 개수가 n개면, n+1레벨까지,  하위 레벨에서 상위 레벨 순으로 데이터 집계 |
| **CUBE**(속성명, 속성명, …) | 인수로 주어진 속성을 대상으로 모든 조합의 그룹별 소계를 구하는 함수,  속성의 개수가 n개면, n2 레벨까지,  상위 레벨에서 하위 레벨 순으로 데이터 집계 |

▶ **윈도우 함수**: GROUP BY절을 이용하지 않고 속성의 값을 집계할 함수를 기술함

-함수의 인수로 지정한 속성이 대상 레코드의 범위가 되는데, 이를 WINDOW라 함

# PARTITION BY: 윈도우 함수가 적용될 범위로 사용할 속성 지정  
→ WINDOW 함수 OVER (PARTITION BY 속성 ORDER BY 속성) [AS 바꾸고 싶은 이름]

|  |  |
| --- | --- |
| **종류** | **설명** |
| **ROW NUMBER**( ) | WINDOW별로 각 레코드에 대한 일련 번호를 반환함  ex) 1, 2, 3, 4, 5 |
| **RANK**( ) | WINDOW 별로 순위를 반환하며, 공동 순위를 반영함  ex) 1, 1, 1, 4, 5 |
| **DENSE\_RANK**( ) | WINDOW 별로 순위를 반환하며,  공동 순위를 무시하고 순위를 부여함  ex) 1, 1, 1, 2, 3 |

**3) 조인(JOIN) \_\_ p.461, 3-43**

- 결합을 의미하며, 관계형 데이터베이스에서의 조인은 교집합 결과를 가지는 결합 방법을 의미

- 두 릴레이션으로부터 연관된 튜플들을 결합해, 하나의 새로운 릴레이션을 반환

▶ 논리적 조인

|  |  |  |
| --- | --- | --- |
| **구분** | **조인 유형** | **설명** |
| **INNER JOIN**  (내부 조인) | EQUI JOIN  (**동**등 조인) | 공통 존재 컬럼의 값이 같은 경우를 추출 |
| NATURAL JOIN  (**자**연 조인) | 두 테이블의 모든 컬럼을 비교해,  같은 컬럼 명을 가진 값이 같은 경우를 추출 |
| CROSS JOIN  (**교**차 조인) | 조인 조건이 없는 모든 데이터 조합을 추출 **★** |
| **OUTER JOIN**  (외부 조인) | LEFT OUTER JOIN  (왼쪽 외부 조인) | 왼쪽 테이블의 모든 데이터와  오른쪽 테이블의 동일 데이터를 추출 |
| RIGHT OUTER JOIN  (오른쪽 외부 조인) | 오른쪽 테이블의 모든 데이터와  왼쪽 테이블의 동일 데이터를 추출 |
| FULL OUTER JOIN  (완전 외부 조인) | 양쪽의 모든 데이터를 추출 |

#**동자교**

▶ 물리적 조인

|  |  |
| --- | --- |
| **종류** | **설명** |
| **NE**STED-LOOP JOIN  (중첩 반복 조인) | 2개 이상의 테이블에서 하나의 집합을 기준으로 순차적으로 상대방 Row를 결합해 원하는 결과를 조합하는 방식 |
| **SO**RT-MERGE JOIN  (정렬 합병 조인) | 양쪽 테이블의 정렬한 결과를 차례로 검색하면서  연결고리 형태로 합병하는 방식 |
| **HA**SH JOIN  (해시 조인) | 해싱 함수 기법을 활용하여 조인을 수행하는 방식 |

#**네소해**

**SQL 활용 ★★**

**p.476~484**

**1) 절차형 SQL**

- C, JAVA 등의 프로그래밍 언어와 같이 연속적인 실행이나 분기, 반복 등의 제어가 가능한 SQL

- 일반적인 프로그래밍 언어에 비해 효율이 떨어지지만, 연속적인 작업 처리 적합

- BEGIN ~ END 형식으로 작성되는 블록(Block) 구조로 기능별 모듈화 가능

**2) 프로시저(Procedure) #디비컨SET**

- 호출을 통해 실행되어 미리 저장해 놓은 SQL 작업 수행, 처리 결과는 한 개 이상의 값 혹은 반환을 아예 하지 않음

- 시스템의 일일 마감 작업, 일괄(Batch) 작업 등에 주로 사용됨

**DECLARE**(필수): 프로시저의 명칭, 변수, 인수, 데이터 타입을 정의하는 **선**언부

**BEGIN**(필수): 프로시저의 시작을 의미, **실**행부

**CON**TROL: 조건문 또는 반복문이 삽입되어 순차적으로 처리됨

**S**QL: DML, DCL이 삽입되어 데이터 관리를 위한 작업 수행 ★

**E**XCEPTION: BEGIN ~ END 안의 구문 실행 시 **예**외가 발생하면 이를 처리

**T**RANSACTION: 수행된 데이터 작업들을 DB에 적용할지 말지 결정하는 처리부

**END**(필수): 프로시저의 종료를 의미, BEGIN/END는 함께 다님

→ **CREATE** [OR REPLACE] **PROCEDURE** 프로시저명(파라미터) [지역변수 선언]

**BEGIN**

프로시저 BODY;

**END**;

\* OR REPLACE: 선택적인 예약어, 동일한 프로시저 이름이 이미 존재하는 경우 기존의 프로시저를 대체할 수 있음

▶ **EXECUTE, EXEC, CALL** 프로시저명; / **DROP** PROCEDURE 프로시저명;

**3) 트리거(Trigger) \_\_ 3-2, 20년 1, 2회 기출문제 #디이비컨SE**

- 삽입, 갱신, 삭제 등의 이벤트가 발생할 때마다 관력 작업을 자동 수행 ★

- 데이터베이스에 저장되며, 데이터 변경 및 무결성 유지, 로그 메시지 출력 등의 목적으로 사용됨

- DCL(데이터 제어어)을 사용할 수 없으며, DCL이 포함된 프로시저나 함수를 호출하는 경우에도 오류 발생

- 트리거에 오류가 있는 경우 트리거가 처리하는 데이터에도 영향을 미치므로 트리거를 생성할 때 세심한 주의가 필요

**DECLARE**(필수): 트리거의 명칭, 변수 및 상수, 데이터 타입을 정의하는 **선**언부

**EVENT**(필수): 트리거가 실행되는 조건을 명시

**BEGIN**(필수): 트리거의 시작을 의미, **실**행부

**CON**TROL: 조건문 또는 반복문이 삽입되어 순차적으로 처리됨

**S**QL: DML문이 삽입되어 데이터 관리를 위한 작업 수행 ★

**E**XCEPTION: BEGIN ~ END 안의 구문 실행 시 **예**외가 발생하면 이를 처리

**END**(필수): 트리거의 종료를 의미, BEGIN/END는 함께 다님

→ **CREATE** [OR REPLACE] **TRIGGER** 트리거명 [동작시기 옵션][동작 옵션] ON 테이블명

**REFERENCING**[NEW | OLD] AS 테이블명

**FOR EACH ROW** [WHEN 조건식]

**BEGIN**

트리거 BODY;

**END**;

\* 동작시기 옵션: AFTER(테이블이 변경된 후 트리거 실행), BEFORE(변경되기 전 실행)

\* NEW | OLD: NEW(추가되거나 수정에 참여할 테이블), OLD(수정되거나 삭제 전 테이블)

\* FOR EACH ROW: 각 튜플마다 트리거 적용

▶ **DROP** TRIGGER 트리거명;

**3) 사용자 정의 함수 \_\_ 3-5 #디비컨SER**

- 프로시저와 유사하게 SQL을 사용해 일련의 작업을 연속적으로 처리

- 종료 시 예약어 RETURN을 사용해 처리 결과를 단일값으로 반환

- DML문(SELECT, INSERT, DELETE, UPDATE)의 호출에 의해 실행됨

- RETURN을 통해 값을 반환해, 출력(OUT) 파라미터가 없음

- INSERT, DELETE, UPDATE로 테이블 조작은 할 수 없고, SELECT로 조회만 할 수 있음

- 프로시저를 호출해 사용할 수 없음 ★

|  |  |  |
| --- | --- | --- |
| **구분** | **프로시저** | **사용자 정의 함수** |
| 반환값 | 없거나 1개 이상 가능 | 1개(단일값) |
| 파라미터 | 입, 출력 가능(IN, OUT) | 입력만 가능(IN) |
| 사용 가능 명령문 | DML, DCL | SELECT |
| 호출 | 프로시저, 사용자 정의 함수 | 사용자 정의 함수 |
| 사용 방법 | 실행문 | DML에 포함해서 사용 |

**DECLARE**(필수): 사용자 정의 함수의 명칭, 변수, 인수, 데이터 타입을 정의하는 **선**언부

**BEGIN**(필수): 사용자 정의 함수의 시작을 의미, **실**행부

**CON**TROL: 조건문 또는 반복문이 삽입되어 순차적으로 처리됨

**S**QL: SELECT문이 삽입되어 데이터 관리를 위한 작업 수행 ★

**E**XCEPTION: BEGIN ~ END 안의 구문 실행 시 **예**외가 발생하면 이를 처리

**RETURN**(필수): 호출 프로그램에 반활할 값이나 변수를 정의 ★

**END**(필수): 사용자 정의 함수의 종료를 의미, BEGIN/END는 함께 다님

→ **CREATE** [OR REPLACE] **FUNCTION** 사용자 정의 함수명(파라미터) [지역변수 선언]

**BEGIN**

사용자 정의 함수 BODY;

**RETURN** 반환값;

**END**;

**DMBS 접속 기술 ★**

**p.489**

**1) 웹 응용 시스템의 구조**

- 사용자 **←→** 웹 서버 **←→** WAS **←→** DBMS

# 사용자는 웹 서버에 접속해 데이터를 주고 받고, 웹 서버는 WAS에게 해당 요청을 전달함, 그 다음 WAS는 수신한 요청을 트랜잭션 언어로 변환한 후 DBMS에 전달해 데이터를 받으면, 이 데이터를 다시 웹 서버로 전달해 사용자에게 도달하게 함

**2) DBMS 접속 기술**

▶ JDBC(Java Database Connectivity)

-1997년 썬 마이크로시스템에서 출시, JAVA 언어로 다양한 종류의 데이터베이스에 접속하고 SQL문을 수행할 때 사용되는 표준 API

-접속하려는 DBMS에 대한 드라이버가 필요

▶ ODBC(Open Database Connectivity)

-1992년 마이크로소프트에서 출시, 데이터베이스에 접근하기 위한 표준 개방형 API로 개발 언어에 관계없이 사용 가능

-ODBC도 접속하려는 DMBS에 맞는 드라이버가 필요하지만, 접속하려는 DBMS의 인터페이스를 알지 못하더라도 ODBC 문장을 사용해 SQL을 작성하면 ODBC에 포함된 드라이버 관리자가 해당 DBMS의 인터페이스에 맞게 연결해줌

→ DBMS의 종류를 몰라도 됨

**3) 정적 SQL vs 동적 SQL**

|  |  |  |
| --- | --- | --- |
|  | **정적 SQL(Static SQL)** | **동적 SQL(Dynamic SQL)** |
| **SQL 구성** | 커서(Cursor)를 통한 정적 처리 | 문자열(String) 변수에 담아 동적 처리 |
| **개발 패턴** | 커서의 범위 안에서 반복문을 활용해 SQL 작성 | NVL 함수를 사용할 필요없이 로직을 통해 SQL 작성 |
| **실행 속도** | 빠름 | 느림 |
| **사전 검사** | 가능 | 불가능 → SQL 변형 위험 |

**ORM(Object-Relational Mapping) ★**

**p.496**

**1) ORM의 개요**

- 객체(Object)와 관계형데이터베이스(RDB)의 데이터를 연결(Mapping)하는 기술 ★

- ORM으로 생성된 가상의 객체지향 데이터베이스는 프로그래밍 코드 또는 데이터베이스와 독립적이므로 재사용 및 유지보수 용이

- 직관적이고 간단하게 데이터 조작 가능

**2) ORM 프레임워크**

|  |  |
| --- | --- |
| **언어** | **프레임워크** |
| JAVA | JPA, Hibernate, Eclipse Link, Data Nucleus, Ebean 등 |
| C++ | ODB, QxOrm 등 |
| Python | Django, SQL Alchemy, Storm 등 |
| iOS | Core Date, Database Objects 등 |
| .NET | NHibernate, Database Objects, Dapper 등 |
| PHP | Doctrine, Propel, RedBean 등 |

**3) ORM의 한계**

- 프레임워크가 자동으로 SQL을 작성하기 때문에 의도대로 작성되었는지 확인해야 함

- 객체지향적인 사용 고려와 프로젝트가 크고 복잡해질수록 적용하기 어려워짐

- 기존의 기업들은 ORM을 고려하지 않은 데이터베이스를 사용하고 있기 때문에, ORM에 적합하게 변환하려면 많은 시간과 노력 필요

**쿼리 성능 최적화 ★**

**p.498, 3-107**

- 데이터 입, 출력 애플리케이션의 성능 향상을 위해 SQL 코드를 최적화하는 것

- 쿼리 성능 최적화하기 전, 성능 측정 도구인 APM(Application Performance Management)을 사용해 최적화 할 쿼리를 선정해야 함

- 최적화 할 쿼리에 대해 옵티마이저가 수립한 실행 계획(Execution Plan)을 EXPLAIN 명령어를 통해 검토하고, SQL 코드와 인덱스 재구성

\***옵티마이저(Optimizer)**: 작성된 SQL이 가장 효율적으로 수행되도록 최적의 경로를 찾아 주는 모듈

**1) RBO(Rule Based Optimizer) vs CBO(Cost Based Optimizer)**

|  |  |  |
| --- | --- | --- |
|  | **RBO** | **CBO** |
| **최적화 기준** | 규칙에 정의된 우선순위 | 액세스 비용 |
| **성능 기준** | 개발자의 SQL 숙련도 | 옵티마이저 알고리즘의 예측 성능 |
| **특징** | 실행 계획 예측이 쉬움 | 성능 통계치 정보 활용, 예측이 복잡함 |
| **고려사항** | 개발자의 규칙 이해도, 규칙의 효율성 | 비용 산출 공식의 정확성 |

**2) SQL 코드 및 인덱스 재구성**

▶ **SQL 코드 재구성**

-서브 쿼리에 특정 데이터가 존재하는지 확인 할 때는 IN보다 EXISTS 활용

-실행 계획이 잘못되었다고 판단되는 경우 힌트(Hint)를 활용해 변경

▶ **인덱스 재구성**

-인덱스의 추가 및 변경은 해당 테이블을 참조하는 다른 SQL문에도 영향을 줄 수 있으므로 신중히 결정

-단일 인덱스로 쓰기나 수정 없이 일기로만 사용되는 테이블의 경우 IOT(Index-Organized Table) 구성 고려

**데이터 전환 ★**

**p.508, 3-109**

**1) 데이터 전환의 정의**

운영 중인 기존 정보 시스템에 축적되어 있는 데이터를 추출(Extraction)하여 새로 개발할 정보 시스템에서 운영 가능하도록 변환(Transformation) 후, 적재(Loading)하는 일련의 과정  
# **ETL**(Extraction, Transformation, Loading): 추출, 변환, 적재 과정

# **데이터 이행**(Data Migration), 데이터 이관이라고도 함

**2) 데이터 전환 계획서**

|  |  |
| --- | --- |
| **항목** | **세부 항목** |
| 데이터 전환 개요 | 데이터 전환 목표  주요 성공 요인  전제조건 및 제약 조건 |
| 데이터 전환 대상 및 범위 |  |
| 데이터 전환 환경 구성 | 원천 시스템 구성도(As-Is 시스템)  목적 시스템 구성도(To-Be 시스템)  전환 단계별 DISK 사용량 |
| 데이터 전환 조직 및 역할 | 데이터 전환 조직도  조직별 역할 |
| 데이터 전환 일정 |  |
| 데이터 전환 방안 | 데이터 전환 규칙  데이터 전환 절차  데이터 전환 방법  데이터 전환 설계  전환 프로그램 개발 및 테스트 계획  데이터 전환 계획  데이터 검증 방안 |
| 데이터 정비 방안 | 데이터 정비 대상 및 방법  데이터 정비 일정 및 조직 |
| 비상 계획 | 종합상황실 및 의사소통 체계 |
| 데이터 복구 대책 |  |

**추가 정리, 수제비 및 기출문제** ★★★

**1) WHERE 조건 ★ \_\_ 3-8, 20년 3회 기출문제**

|  |  |  |
| --- | --- | --- |
| **구분** | **연산자** | **사례** |
| 비교 | =, <>, <, <=, >, >=  \*<>: 다름을 의미 | 가격(PRICE)이 50000 미만  → PRICE < 50000 |
| 범위 | BETWEEN | 가격(PRICE)이 50000 보다 크거나 같고 80000 보다 작거나 같음  → PRICE **BETWEEN** 50000 **AND** 80000 ★  = PRICE >= 50000 AND PRICE <=80000 |
| 집합 | IN, NOT IN | 가격(PRICE)이 40000 또는 50000 또는 60000  → PRICE IN (40000, 50000, 60000) ★ |
| 패턴 | LIKE | 이름(NAME)이 ‘정보’로 시작되는 문자열  → NAME LIKE ‘정보%’ ★★ |
| NULL | IS NULL, IS NOT NULL | 가격(PRICE)이 NULL 값인 경우  → PRICE IS NULL ★ |
| 복합조건 | AND, OR, NOT | 가격(PRICE)이 50000 미만이고 이름(NAME)이 ‘정보’로 시작되는 문자열  → PRICE < 50000 AND NAME LIKE ‘정보%’ |

**2) LIKE와 같이 사용하는 와일드 문자 ★ \_\_ 3-9**

|  |  |  |
| --- | --- | --- |
| **문자** | **설명** | **사례** |
| + | 문자열을 연결 | → ‘축구’ + ‘감독’ = “축구 감독” |
| % | 0개 이상의 문자열과 일치 | → LIKE ‘%구’ = 축’구’, 농’구’, 배’구’ **★**  → LIKE ‘%구%’ = 축’구’선수, 농’구’코트, ‘구’심력, 피’구’ |
| [ ] | 1개의 문자와 일치 | → ‘[0-8]%’ = 0-8 사이 숫자로 시작하는 문자열 |
| [ ^ ] | 1개의 문자와 불일치 | → ‘[^0-8]%’ = 0-8 사이 숫자로 시작하지 않는 문자열 |
| \_ | 특정 위치의 1개의 문자와 일치 | → ‘\_구%’ = 축’구’, 축’구’선수 |

**3) 주석 처리 \_\_ 3-9**

|  |  |
| --- | --- |
| **주석 기호** | **설명** |
| -- | ‘—‘이 시작하는 위치부터 해당 라인 끝까지 실행이 되지 않도록 함 |
| /\* 문장 \*/ | ‘/\*’이 시작되는 부분부터 ‘\*/’이 나타날 때까지의 여러 라인을 실행 되지 않도록 함 **★** |

**4) 힌트의 사용 \_\_ 3-9**

- SQL 문에 사전 정보를 줘서, SQL문 실행에 빠른 결과를 가져오는 효과를 만드는 문법

|  |  |
| --- | --- |
| **힌트** | **설명** |
| --+ 힌트명(파라미터) | ‘--+’이 시작되는 위치부터 힌트로 인식 |
| /\*+ 힌트명(파라미터) \*/ | ‘/\*+’이 시작되는 부분부터 ‘\*/’ 사이를 힌트로 인식 |

**5) 집합 연산자 \_\_ 3-41**

- 테이블을 집합 개념으로 보고, 두 테이블 연산에 집합 연산자를 사용하는 방식

- 여러 질의 결과를 연결해 하나로 결합하는 방식을 사용

|  |  |
| --- | --- |
| **집합 연산자** | **설명** |
| UNION | 중복 행이 제거된 쿼리 결과 집합(합집합) |
| UNION ALL | 중복 행이 제거되지 않은 쿼리 결과 집합 |
| INTERSECTION | 두 쿼리 결과에 공통적으로 존재하는 집합(교집합) |
| MINUS | 첫 쿼리에 있고 두 번째 쿼리에는 없는 집합(차집합) |

**6) 서브쿼리(Sub-Query) \_\_ 3-46**

- SQL문 안에 포함된 또 다른 SQL문

|  |  |
| --- | --- |
| **종류** | **설명** |
| 단일 행(Single Row) 서브쿼리 | 결과가 항상 1건 이하인 서브쿼리 |
| 다중 행(Multiple Row) 서브쿼리 | 실행 결과가 여러 건인 서브쿼리 |
| 다중 컬럼(Multiple Column) 서브쿼리 | 결과가 여러 컬럼으로 반환되는 서브쿼리 |

**7) 데이터 지역화(Data Locality) ★ \_\_ 3-91**

- 데이터베이스의 저장 데이터를 효율적으로 이용할 수 있도록 저장하는 방법

- **구역성(Locality)**라고도 함

▶ 데이터 지역화의 종류

|  |  |  |
| --- | --- | --- |
| **종류** | **설명** | **활용** |
| **시**간적 구역성 | 최근에 참조된 기억장소가 가까운 장래에 계속 참조될 가능성이 높은 특성  # Stack(스택), Subroutine(서브루틴), Loop(루프),  Counting(카운팅), Totaling(집계) **★** | for, while 같은 반복문에 사용하는 조건 변수 |
| **공**간적 구역성 | 최근에 참조된 기억장소와 가까운 기억정보가 가까운 장래에 계속 참조될 가능성이 높은 특성  # Array(배열), Sequential Code(순차적 코드) **★** | A[0], A[1] 같은 배열에 연속 접근 |
| **순**차적 구역성 | 별도의 분기가 없는 한, 데이터가 기억장치에 저장된 순서대로 순차적으로 인출되고 실행될 가능성이 높은 특성 | 1:1, 1:N, N:M 관계 존재 |

#**시공순**

▶ 데이터 지역화를 활용한 관리 기법

|  |  |
| --- | --- |
| **종류** | **설명** |
| 기억장치 계층구조(Hierarchy) | CPU → 캐시 메모리 → 메인 메모리 순서로 접근시간(Access Time)을 효과적으로 단축 |
| 캐시 접근시간 단축 | 캐시 적중률(Cache Hit Ratio)의 극대화 가능 |
| 워킹세트(Working Set) | 하나의 페이지(Page)가 자주 접근하는 페이지들의 집합,  페이지 폴트(Page Fault)를 줄여 스레싱(Thrashing) 감소 **★** |

**8) 병행제어의 로킹(Locking) 단위 ★★ \_\_ 개정 전 / 20\_1, 2, 3회 / 21년 1회 기출문제**

- 한번에 한 명만 사용할 수 있게 잠그는(Locking) 단위

- 로킹의 대상이 되는 객체의 크기를 로킹 단위라고 함

- 데이터베이스, 파일, 레코드 등은 로킹 단위가 될 수 있음

- 한꺼번에 로킹할 수 있는 객체의 크기를 로킹 단위라고 함

▶ **로킹 단위가 작으면 小**

- 로킹 오버헤드가 증가함

- 데이터베이스 공유도가 증가함 (= 병행성 수준이 높아짐)

▶ **로킹 단위가 크면 大**

- 로킹 오버헤드가 감소함

- 데이터베이스 공유도가 감소함 (= 병행성 수준이 낮아짐)

**9) 데이터베이스 로그(log)를 필요로 하는 회복기법 \_\_ 20년 3회 기출문제**

**#지RE 즉UN**

▶ **지연 갱신 기법(Deferred Update)**

-트랜잭션이 부분 완료 상태에 이르기까지 발생한 모든 변경 내용을 로그 파일에만 저장하고, 데이터베이스에는 **COMMIT**이 발생할 때까지 **저장을 지연**하는 기법

-트랜잭션이 실패할 경우 UNDO없이 로그 단순 폐기

**# REDO**

▶ **즉시 갱신 기법(Immediate Update)**

-트랜잭션 수행 도중 데이터를 변경하면 변경 정보를 로그 파일에 저장하고, 부분 완료되기 전이라도 모든 변경 내용을 **즉시 데이터베이스에 반영**하는 기법

-로그 파일을 참조해 **미완료된 변경**에 대해 **UNDO를 우선 실행**한 후, 완료된 변경에 대해 REDO 실행 (UNDO는 **COMMIT된 지점이 없음**)

**# UNDO**

**\* 정보처리기사 실기, REDO/UNDO 기출문제**

