데이터베이스 관리
- DBA 관점의 품질관리

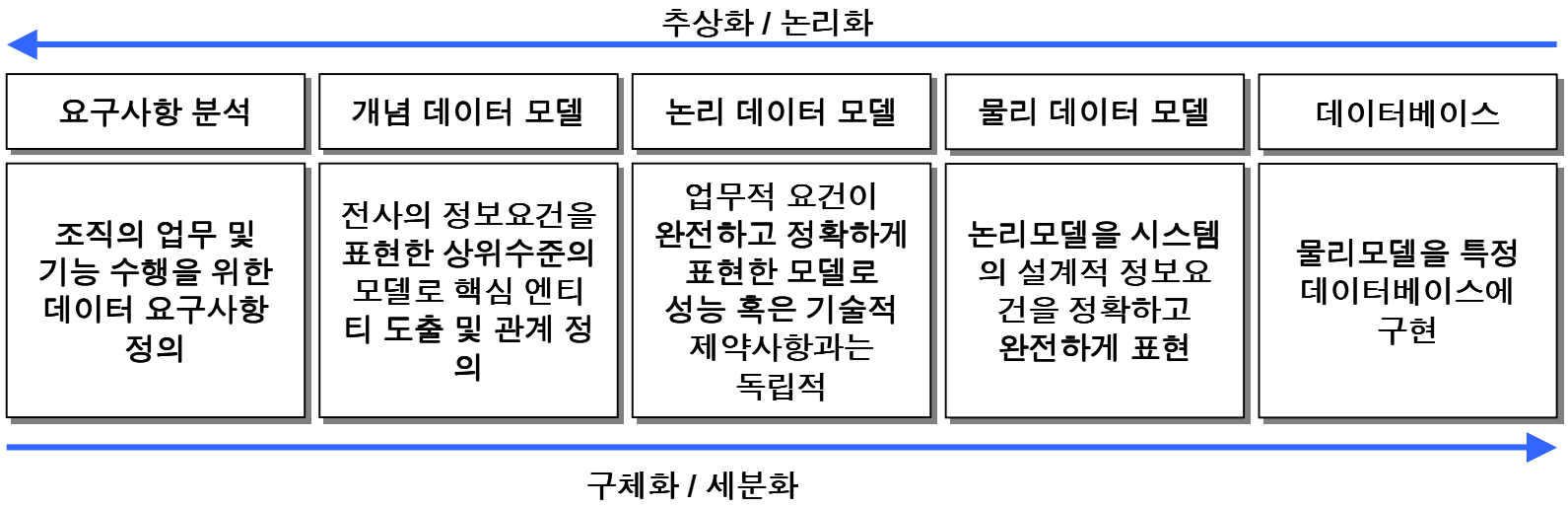
물리 데이터 모델 및 데이터베이스 관리

김문영 수석 컨설턴트
비투엔 컨설팅

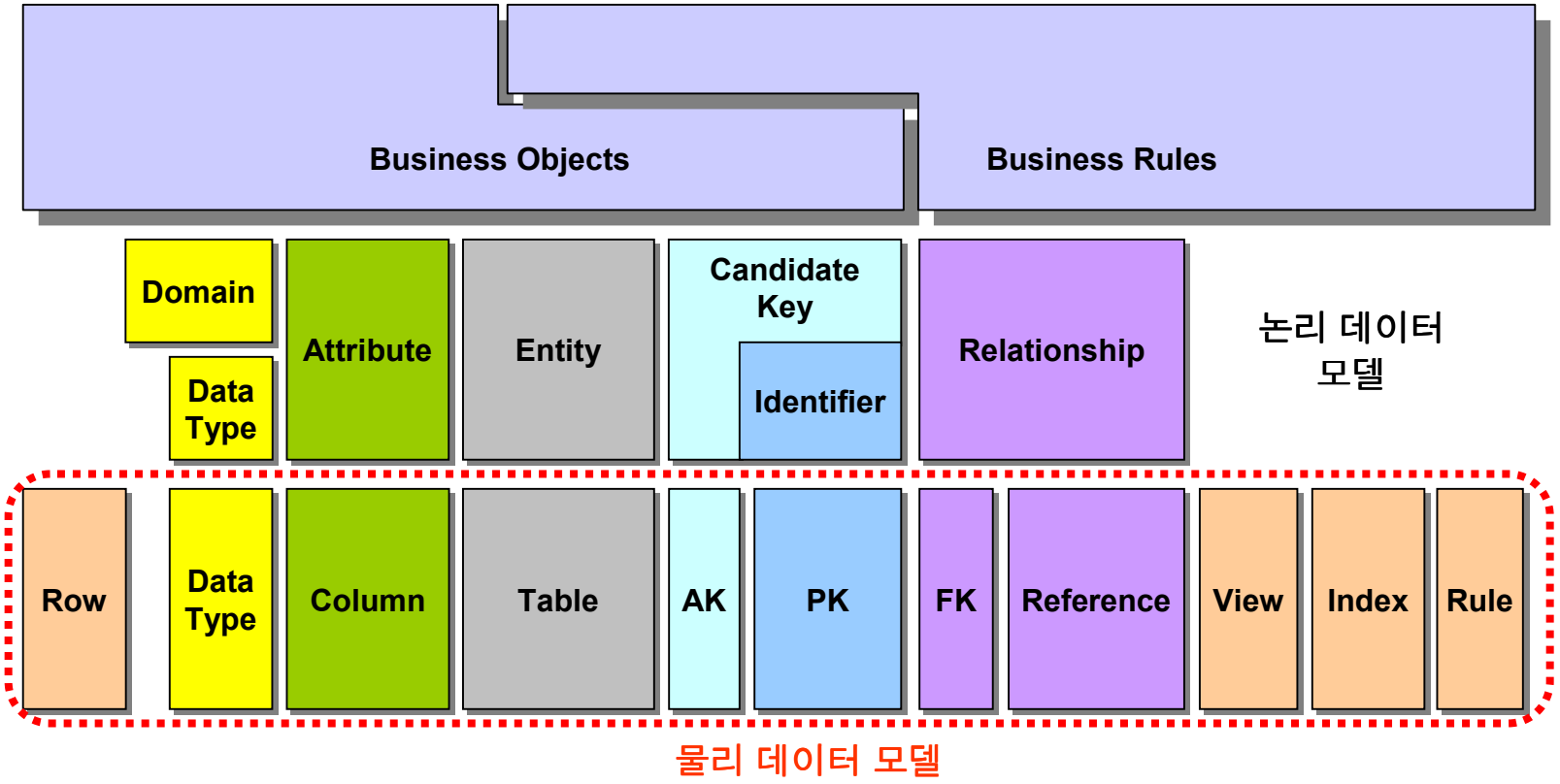
목차

■ 데이터 모델

- 사람이 인식하는 현실세계를 단순화하고 추상화시킨 모형을 데이터 구조로 표현한 것
- 정해진 기호와 규칙을 이용하여 조직의 업무 및 기능이 어떻게 운영되고 목적을 달성하기 위하여 어떤 데이터가 필요한지를 공식적이고 구조적으로 분석하여 복잡성을 제거하고 시각화한 것
- 고객과의 의사교환을 용이하게 하고 조직의 정보 요구에 대한 정확한 이해를 제공하여 시스템 개발의 기초자료로 활용
- **Business to Engineering, Data Modeling 작업의 결과로 Data Model이 도출됨**



■ 단계별 데이터 모델


Business Objects
Business Rules
Domain
Data Type
Attribute
Entity
Candidate Key
Identifier
Relationship
논리 데이터 모델
Row
Data Type
Column
Table
AK
PK
FK
Reference
View
Index
Rule
물리 데이터 모델

Ⅰ. 물리 데이터 모델의 품질

■ 물리 데이터 모델의 정의

- 논리 데이터 모델을 **DBMS**의 특성 및 성능을 고려하여 구체화 시킨 모델
- 데이터 모델을 적용할 데이터베이스관리시스템(**DBMS**)에 맞게 논리 데이터 모델을 변환한 것으로 관계형 데이타베이스의 경우 **ER** 데이터 모델링 기법으로 작성된 논리 데이타 모델링의 결과를 일관성있게 이행할 수 있으며 계층형이나 네트워크형 데이타베이스의 경우는 추가적인 작업 필요

■ 관리목적

- 물리 데이터 모델은 데이터베이스관리시스템(**DBMS**) 의 유형 선정 이후에 해당 **DBMS**에서 최적의 성능을 보장하도록 논리 데이터 모델에서 저장하는 데이터의 물리적 특성을 최대한 반영하여 설계하고 이를 관리한다.
- 물리 데이터 모델은 실제 데이터 객체가 아니므로 소홀히 인식될 수 있으나 논리 데이터 모델이 **1:1**로 데이터베이스 객체로 선언되는 것이 아니며 실제 사용자의 **DBMS** 만족도에 가장 큰 영향을 미치는 처리 속도를 고려한 설계 방안을 정의하는 것이므로 중요성을 가지고 관리되어야 한다.

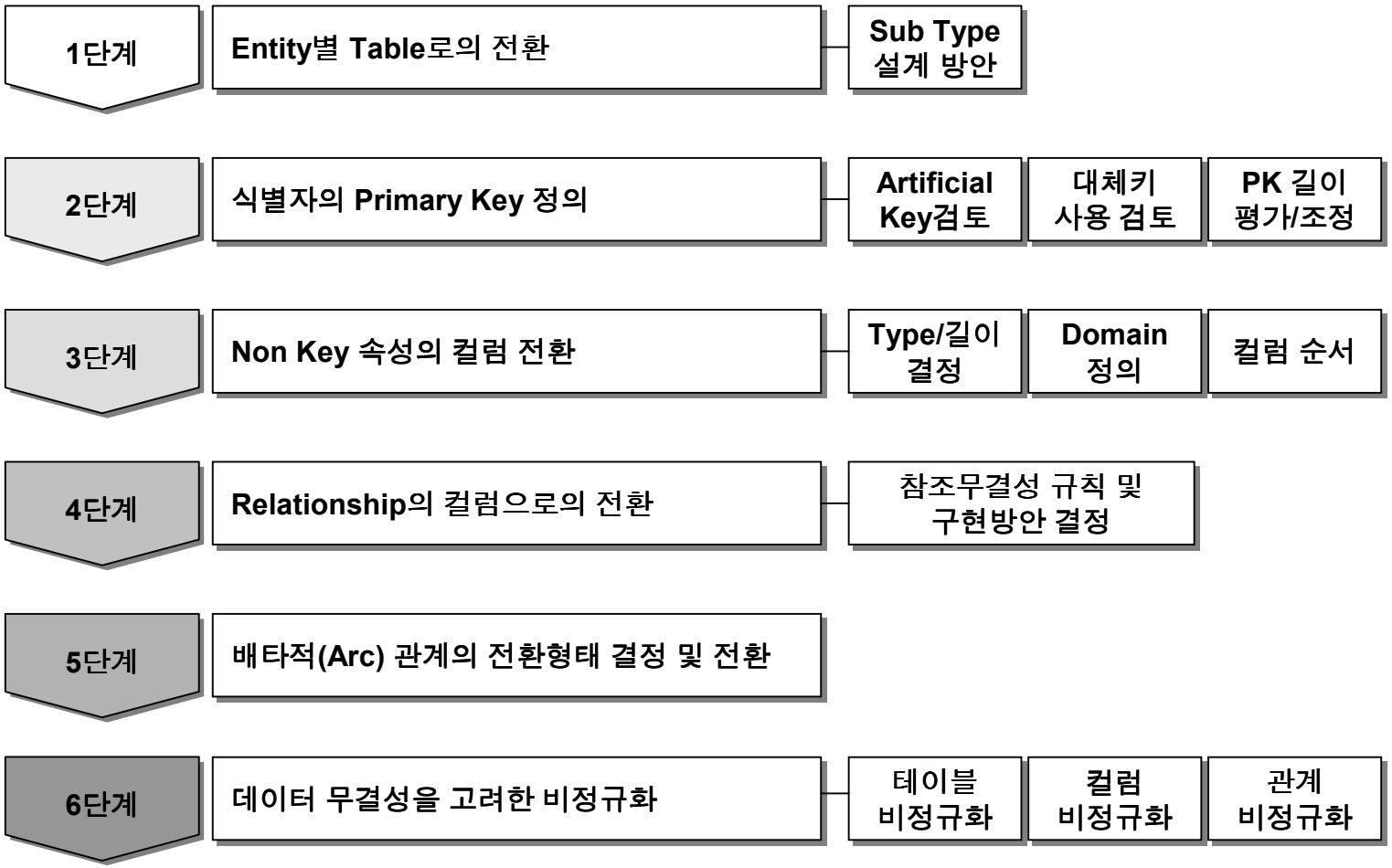
■ 참고) 논리 데이터 모델의 목적

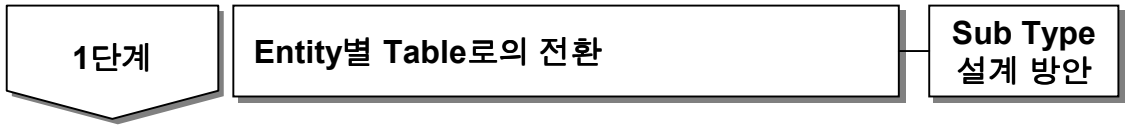
- 비즈니스 영역의 데이터 요구사항을 파악하여 특정 **Application**이나 정보기술 환경에 종속되지 않고 조직 전체의 정보 요구를 체계적으로 수용할수 있으며 향후 변화에 능동적으로 대처 가능한 데이터 구조의 설계

■ 물리 데이터 모델의 설계를 위한 준비

- **최적의 물리 데이터 모델의 설계를 위하여 아키텍처에 대한 전략 및 설계가 선행되어야 한다**
- **플랫폼 및 기술 아키텍처**
  - ✓ 하드웨어, 시스템 소프트웨어, 네트워크 전략 검토
  - ✓ 물리 데이터 모델은 특정 하드웨어 플랫폼과 데이타베이스 관리시스템에 의존하므로 선택된 특정 제품의 기능과 제약에 맞도록 조정
- **참조 무결성 구축방안**
  - ✓ DBMS에 의해 적용
  - ✓ 응용에 의해 적용
  - ✓ 양자 모두 사용여부
- **과거 데이타에 대한 조회 및 감사 추적 요건 확인 및 필요 데이터 구조 결정**
- **백업 및 복구 처리 전략 정의**
- **위치 유형/각 플랫폼 유형을 고려한 데이타 및 프로세스 분산전략**
- **타 시스템에 대한 고려**
- **기존 DB의 설계 검토 및 이행방안 고려**
- **물리 데이터 모델 설계에 필요한 표준 및 원칙 정의**

■ 물리 데이터 모델의 설계의 일반적인 과정





- **데이터 구조 품질 관점에서의 고려 사항**
  - 데이터베이스의 물리적 특성을 고려하여 논리 데이터 모델의 엔티티가 모두 테이블로 전환되었는가?
- **Sub-Type의 데이터 구조 품질적 의미**
  - 데이터 모델이 표현하는 업무 규칙을 보다 명확하게 표현하여 관리 대상 데이터와 업무를 정확하게 이해할 수 있도록 한다
  - 엔티티의 정의를 보다 규체화
  - 속성의 선택성/ 관계의 선택성 제거
  - 논리 데이터 모델링 과정에서 사용자의 데이터 모델에 대한 이해도를 증진시킬 수 있으며 물리 데이터 모델의 설계하는 과정에서 여러가지 형태로 테이블로 전환된다.
- **Sub-Type의 테이블 전환 방안**
  - 하나의 테이블로 통합
  - 각각의 Sub-Type별 테이블로 분리
  - Super-Type 및 Sub-Type 테이블 생성

하나의 테이블로 통합

- **Sub Type**을 **Super Type**에 통합하여 하나의 테이블로 생성하며 통합된 테이블에는 **Super Type**의 데이터는 물론 모든 **Sub Type**의 데이터를 포함한다

- 전환 절차
  - **Super Type**으로 테이블 명칭 정의
  - **Sub Type**을 구분할 수 있는 설계 컬럼 추가
  - **Super Type** 및 **Sub Type**의 속성을 컬럼으로 전환
  - **Super Type** 및 **Sub Type**의 관계를 **Foreign Key**로 전환

- 장점
  - **Super-Type** 및 **Sub-Type**의 정보를 동시에 이용하는 경우에 적절
  - 개별 **Sub Type**만을 참조할 경우 구분속성을 이용하여 가능 (**View** 설계 검토)
  - 복잡한 처리를 하나의 **SQL**로 통합하기가 유리

- 단점
  - **Sub-Type**별 사용 속성 가독성 저하 및 **DBMS**에 의한 **Not NULL** 제한 불가
  - **Sub-Type**에 국한된 정보를 추출 시 성능 감소
  - 테이블의 컬럼 수 및 로우 수 증가, 인덱스 크기 증가 (저장공간의 낭비 발생)
  - **SQL** 사용 시 **Sub-Type**구분자 필요한 경우 다수

Sub Type별 테이블 생성

- 각각의 **Sub Type**별로 테이블을 생성하는 것으로 주로 **Sub-Type**별 다수의 속성이나 관계를 가진 경우에 적용
- 선택적 관계가 존재하는 경우 **Sub-Type**을 분할함으로써 관계가 필수적으로 변하는지 확인

- 전환 절차
  - **Sub Type**마다 테이블 명 부여
  - **Sub Type**별 속성을 컬럼으로 전환, **Sub Type**별 관계를 **FK**로 전환,
  - **Super Type**의 속성을 **Sub-Type** 전환 테이블의 속성으로 전환
  - **Super Type**의 관계를 **Sub-Type** 전환 테이블의 **FK**로 전환

- 장점
  - **Sub-Type**별 속성 가독성 증가, 선택사양의 명확한 정의 가능
  - **SQL** 사용 시 **Sub-Type** 구분자 불필요
  - 하나의 테이블로 통합하는 것과 비교하여 테이블 크기 감소

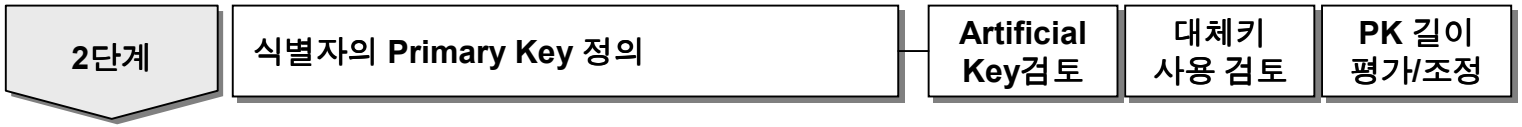
- 단점
  - **Sub-Type** 구분없이 데이터를 처리하는 경우 **UNION (UNION ALL)** 발생
  - 업무 요건이 변경 또는 확장되는 경우 지속적으로 테이블의 수가 증가할 가능성
  - 데이터 통합이 요구되는 업무의 경우 트랜잭션 처리의 복잡도 증가 및 데이터 무결성 유지 어려움
  - **Super Type**의 속성 및 관계가 각 **Sub Type**별로 중복되므로 데이터 무결성 유지를 위한 처리 필요

Super Type 및 Sub Type 테이블 생성

- **Super-Type**과 **Sub-Type**을 각각의 테이블로 변환

- 장점
  - 전체 즉, **Super-Type**을 대상으로 하는 처리와 **Sub-Type**별 독립적인 업무처리가 모두 많은 경우 적합
  - 하나의 테이블로 통합할 경우 컬럼의 수가 너무 많아지는 경우 고려
  - **Super-Type** 및 **Sub-Type**을 대상으로 하는 트랜잭션의 구분이 명확하고 개체의 통합이 필요한 경우 적합

- 고려사항
  - 원칙적으로 **Super-Type**이 전환되는 테이블은 공통 속성이나 관계만을 포함하지만 **Super-Type**과 **Sub-Type**을 **Join**하는 업무가 빈번할 경우 **Access Path**가 되는 컬럼이나 관계가 분산되지 않도록 **Super-Type**으로 이전 또는 중복을 고려



■ 데이터 구조 품질 관점에서의 고려 사항

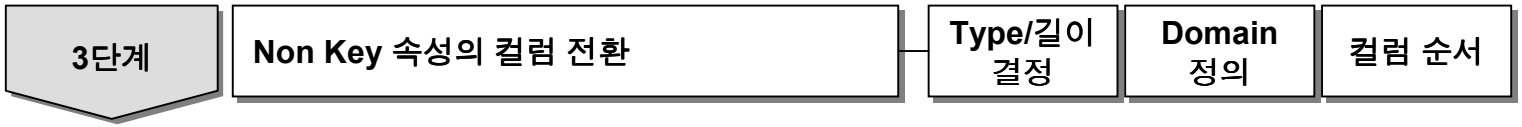
- 논리 데이터 모델의 식별자를 기본으로 하여 성능을 고려한 **Primary Key**를 고려하였는가?
- **Primary Key Constraint**가 **DBMS**내에 명시적으로 선언되어 있는가?
- 식별자는 가능한 업무적으로 의미있는 속성으로 구성되는 것이 바람직하며 무의미한 일련번호나 기타 설계자가 추가하는 식별자는 가능하면 피하는 것이 바람직하다. 특히, **Primary Key Constraint**로 인하여 생성되는 **Index**를 활용하고 무의미한 속성을 **Primary Key**에 추가하는 것은 피해야 한다.
- **Primary Key**를 구성하는 컬럼의 순서는 데이터를 활용하는 **Pattern**을 고려하여 결정한다

■ Artificial Key 검토

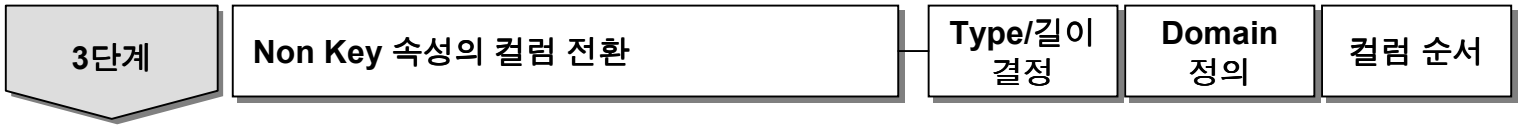
- 테이블의 **Primary Key**는 논리 모델의 식별자 정의 원칙과 마찬가지로 데이터를 유일하게 식별할 수 있는 후보들 중 최소의 개수로 정의된다.
- 일반적으로 **5**개 이상의 속성으로 **Primary Key**가 구성되고 많은 하위 엔티티에 **Primary Key**를 상속하는 경우 **Artificial Key**의 사용을 적극 검토한다.

■ 대체키(Surrogate Key) 사용 검토

- 논리 데이터 모델에서는 도출되지 않는 **Key**로 추가적으로 부여된 설계 속성
- 현업의 업무와는 독립적이므로 사용자에게는 보이지 않음
- 값을 어떻게 할당할 것인가에 대한 고려 필요 (예:**Primary Key**의 변형 등)
- **HW/SW** 특설 활용 (예:**Timestamp**)



- **데이터 구조 품질 관점에서의 고려 사항**
  - 속성이 모두 컬럼으로 전환되었으며 컬럼의 데이터 타입 및 길이, 도메인에 대한 정의가 적절한가
  - 데이터 추적 또는 감사를 위한 시스템 속성의 추가 검토
- **Data Type 및 길이의 결정**
  - 데이터 타입이 문자인 컬럼인 경우 고정길이, 가변길이 결정 필요, 일관된 데이터 타입을 사용하는 것이 바람직
  - 긴 문자열 처리를 위한 데이터 타입의 정의 (Long, CLOB)
  - 숫자 컬럼의 경우 DBMS의 종류에 따라 다양한 데이터 타입이 존재함
- **Domain 정의**
  - 컬럼의 허용값 정의 및 논리데이터 모델의 속성 선택성을 물리 데이터 모델에서 NULL로 처리할 것인지 특정 값으로 처리할 것인지 결정
  - Default값의 정의 (일관된 Default값 정의 원칙 필요, 예: 이력데이터 모델의 유효종료일의 Default값) 및 DBMS의 Default Constraint 사용 여부 결정
  - Domain 정의 시 데이터 타입, 길이 (숫자의 경우 소수점 이하 자릿수도 함께 정의), 표시형식, 허용되는 제약 조건 (상한 값, 하한 값, 범위), 의미 (컬럼의 정의 및 선택성 조건), 유일성, NULL 허용여부, 초기값, 별명 등을 정의
  - 논리 모델에서 정의된 추출 속성일 경우 추출 알고리즘을 제약 조건에 명시



- 컬럼의 순서는 **DBMS** 성능에 영향을 주지 않으나 가독성 등을 고려하여 순서결정
  - **PK** 우선, 업무적 연관성이 높은 컬럼끼리 그룹핑하여 위치, 빈번하게 사용되는 컬럼을 먼저 위치
  - 메모 컬럼과 같이 길이가 긴 텍스트성 속성 또는 특수한 데이터 타입의 컬럼은 후분부에 위치시킴

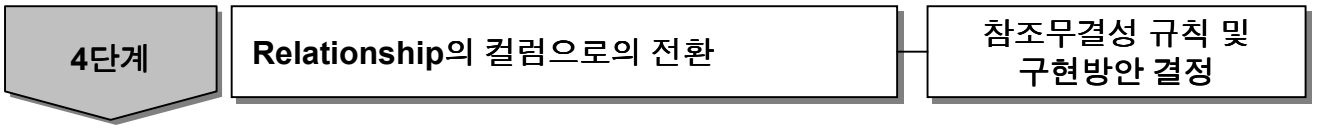
4단계 | Relationship의 컬럼으로의 전환 | 참조무결성 규칙 및 구현방안 결정

■ 데이터 구조 품질 관점에서의 고려 사항

- 논리 데이터 모델에서 정의된 참조 무결성 규칙의 구현방안이 정의되었는가?
- **Relationship이 모두 Column으로 정의되었는가?**

■ 참조 무결성 규칙 (Referential Integrity)

- **삭제 및 수정 규칙 (부모 실체의 건이 삭제/수정될 때 적용한다)**
  - ✓ Restrict : 대응되는 자식 실체가 없는 경우 부모 실체의 삭제 허용
  - ✓ Cascade : 부모실체의 삭제 허용 및 해당 자식 실체의 모든 건을 자동 삭제
  - ✓ Nullify : 부모 실체의 삭제 허용 및 해당 자식 실체의 외부 식별자 Null 처리
  - ✓ Default : 부모 실체의 삭제 허용 및 해당 자식 실체의 외부 식별자 기본 값 처리
  - ✓ No Effect : 조건없이 허용
- **입력 규칙 (자식 실체의 입력/수정 허용 및 대응되는 부모 실체의 자동 생성)**
  - ✓ Dependent : 대응되는 부모 실체가 있는 경우 자식 실체의 입력 허용
  - ✓ Automatic : 부모실체의 자동 생성
  - ✓ Nullify : 부모 실체가 없는 경우 대응되는 FK Null 처리
  - ✓ Default : 부모 실체가 없는 경우 대응되는 FK 기본 값 처리
  - ✓ No Effect : 조건없이 허용



■ 참조 무결성 규칙 (Referential Integrity) 적용 방안

- 논리 데이터 모델의 관계를 **FK Constraint**로 정의할 것인지 결정 필요 (**FK Constraint** 선언 시 대량의 배치 작업에서 성능상의 문제를 발생시킬 수 있음)
- **Foreign Key Constraint**로 선언되지 않은 참조 무결성은 **Trigger**, 응용 프로그램에 의하여 보장
- 조건없이 허용되는 경우는 엄격하게 제한되어야 함
- **Nullify** 보다 **Default** 규칙 적용
- **Sub-Type**에 의하여 분리된 테이블의 경우는 입력 시 **Dependent/Automatic**, 삭제 시 **Restrict/Cascade** 규칙을 적용하도록 할 것
- 코드 테이블의 참조 무결성은 **Foreign Key Constraint** 보다 응용 프로그램에 의한 무결성 보장
- **DBMS**별로 지원하는 참조 무결성 규칙에는 한계가 있으므로 미리 대상 **DBMS**의 기능을 파악한 후 **FK**를 정의한다

5단계 배타적(arc) 관계의 전환 형태 결정 및 전환

■ 배타적 관계의 정의

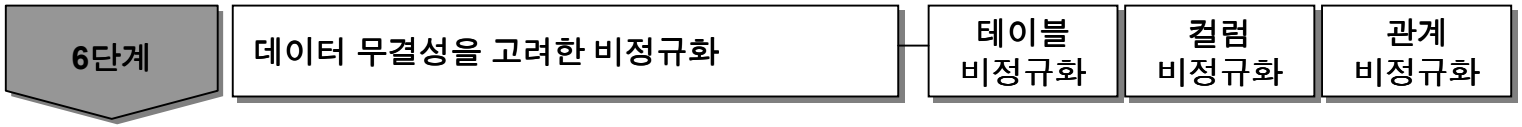
- 어떤 실체가 두개 이상의 다른 실체의 합집합과 관계를 가지는 것을 배타적 **(Exclusive)** 또는 **Arc** 관계라고 하며 배타적 관계의 **Optionality** 는 동일하다.

■ 배타적 관계의 전환형태 1 - 관계통합

- 배타적 관계를 물리적으로 컬럼으로 전환 시 1개의 관계처럼 전환하는 것을 의미
- 배타적 관계를 형성하는 각 관계 컬럼의 개수, 데이터 타입, 길이가 동일할 경우 적용
- 컬럼의 숫자 감소 및 인덱스 구성 및 **SQL** 적용 시 유리함
- 관계를 형성하는 속성의 의미가 상이하므로 컬럼명 정의 시 주의필요 (예:주민번호, 법인번호 통합 시 주민법인번호 사용)

■ 배타적 관계의 전환형태 2 – 개별적으로 관계를 컬럼으로 전환

- 배타적 관계를 형성하는 각 관계 속성의 개수, 데이터 타입, 길이가 상이할 경우 적용
- 각 컬럼의 의미가 명확해지는 장점 존재
- 배타적 관계 모두 컬럼으로 전환되므로 컬럼의 수가 증가하고 인덱스 구성 및 **SQL** 적용 시 불리함



■ 비정규화의 의의

- 정규화된 논리 데이터 모델을 물리적 데이터 구조로 전환된 것을 기반으로 데이터베이스에 대한 성능 향상을 고려하여 테이블에 대한 비정규화 수행
- 비정규화 작업은 이미 정규화된 모델을 시스템의 성능 향상을 위해 재조정하는 작업이므로 반드시 정규화된 모델에서 출발해야 하고 비정규화 작업을 할 경우에는 데이터의 무결성을 보장하기 위한 조치가 선행되어야 한다.
- 비정규화의 방법은 컬럼을 중복시켜 하나의 질의가 접근하는 테이블 개수를 줄이는 방법
- 파생 또는 요약 데이터를 사용하여 접근 테이블 또는 접근 레코드의 개수를 줄이는 방법
- 대체키를 사용하는 방법, 반복 그룹을 허용하는 방법 등

■ 비정규화의 적정성 판단 원칙

- 정규화와 비정규화의 균형 유지 필요
- 조회 성능과 데이터 중복으로 인한 불이익간의 **Trade off** 판단
- 데이터의 의미가 변형되거나 업무 규칙이 감추어져서는 안됨
- 정규화될수 록 **Update**가 많은 경우에 유리, 비정규화는 **Query** 성능 향상 효과
- 단순히 **Join**을 피하기 위한 비정규화 지양

- 테이블의 분할
- 수평적 분할(Horizontal Fragmentation)
  - 한 테이블내 로우(Rows)의 수가 많고, 각기 다른 사용자 그룹이 일정한 형태로 구분되는 특정 Row만을 사용하는 경우 수평 분할 고려
  - 테이블의 레코드를 기준으로 분할하는 것으로 테이블의 내용은 PK 값에 따라 다르며, 테이블의 PK와 Non-Key 간의 관계는 분할 전후가 동일하다.
  - 분할된 데이터를 동시에 접근해야 하는 경우에는 여러 개의 테이블을 접근해야 하므로 성능저하 요인이 발생할 수 있음.
  - 각 Row는 분할된 한 테이블에만 존재하여야 하며 부분범위 처리 및 PK의 유일성 보장이 어려움.
  - 테이블의 분할 시에는 발생 가능한 상황 즉, 테이블 운영의 복잡도 증가, 테이블 관리의 위험도 증가, 성능 증가 등의 상황을 고려하여 결정
  - 테이블의 수평 분할이 아닌 Partition 기능을 이용하는 것이 바람직
- 수직적 분할(Vertical Fragmentation)
  - 서로 다른 사용자 그룹이 특정 컬럼의 집합만을 자주 접근하는 경우에는 수직 분할을 고려한다.
  - Column의 수가 너무 많고 특정 컬럼만을 조회하는 Critical Access Program이 존재하는 경우 고려 (예:Login Check)
  - 컬럼들이 분할되어 있으므로 갱신이 복잡하고, 분할된 테이블들 간의 입력 및 삭제 규칙에 대한 추가적인 정의가 필요하다. 분할된 각 테이블마다 주키가 포함되므로 주키의 중복에 따른 Space의 낭비가 발생한다
  - 분할된 테이블간의 Join이 많이 발생하는지 검증할 것 (많이 발생할 경우 부적절한 분할)

Ⅰ. 물리 데이터 모델의 품질

■ 테이블의 통합

- **1 : 1**의 관계를 갖는 **Table**을 통합 **(1:1 Mandantory**의 경우 반드시 통합**)**
- 같이 처리되는 경우가 빈번한 **Sub-Type**의 통합 고려
- **Index**가 많아질 수 있음
- 공유되지 않는 문자 형태의 **Column**은 가변 길이 이용
- **Row**수가 증가하여 **Full Table Scan**시 처리량 증가

■ 요약 테이블의 생성

- **많은 엔티티를 참고하고 방대한 양의 데이터를 빈번하게 집계하여 처리하는 경우 요약 테이블 생성 고려**
- **통계정보를 용이하게 도출할 수 있는 구조 반영 (예:Star Schema)**
- 레코드의 수정이나 삭제를 엄격하게 제한할 것
- **요약 테이블의 데이터 생성 시점 고려**
  - ✓ 원천 데이터 발생 시 실시간으로 반영 (성능 고려)
  - ✓ 특정 주기 (일,주,월,분기 등)를 설정하여 일괄 작업 수행

■ 컬럼의 중복

- **빈번하게 Join을 발생하거나 조회 조건으로 자주 사용되는 컬럼**
- **중복된 컬럼의 무결성 보장을 위한 방안 확보 (연쇄 작용으로 정의할 것)**
- **추출 컬럼의 추가**
  - ✓ AVG, SUM 등의 계산결과.
  - ✓ Indicator : Relation의 Optionality가 높은 경우 효과적.
  - ✓ System 일자, 입력자 ID 등.
  - ✓ Data의 성격과 Access Pattern을 고려하여 적용.

View의 정의

- View는 원칙적으로 하나 이상의 기본 테이블로부터 유도된 이름을 가진 가상 테이블로 사용자에게는 column과 row로 구성된 테이블로 보이지만 SQL문에 의해 생성된 질의 결과
- View의 장점
  - ✓ 사용자들이 필요에 따라 데이터베이스를 각기 다른 관점에서 볼 수 있도록 허용한다.
  - ✓ 사용자들이 테이블의 특정 열이나 행만을 볼 수 있도록 함으로써 사용자의 데이터 접근을 제한한다.
  - ✓ 각각의 사용자에 대해 관심있는 데이터만을 보여줌으로써 사용자의 데이터 접근을 단순화한다.
  - ✓ 테이블과 달리 물리적인 저장공간을 가지지 않으며 (Materialized View 제외) 관련정보가 단지 Data Dictionary에 저장될 뿐이나 테이블과 거의 동등하게 취급될 수 있는 논리적인 집합으로써 정규화 규칙을 무시하고 목적에 따라 자유롭게 사용할 수 있다.
  - ✓ 따라서, 뷰의 설계는 다수의 사용자들이 자주 동일하게 사용하는 뷰에 대해서만 미리 조사하여 설계단계에 반영할 수 있도록 한다

분산 DB에서의 무결성 설계

- 테이블에 대한 분산 설계(테이블 분할, 테이블 요약, 테이블 복제 등) 작업에 따른 추가 또는 조정된 테이블에 대한 추가적인 무결성 설계 부문을 보완
- 테이블 복제(Replication) 및 분할(Fragmentation)에 따른 데이터 무결성 손상, 즉 데이터베이스의 의미적 변경이 없다는 것을 보장
- 완전성의 원칙: 테이블의 분할 전의 모든 데이터 항목은 분할 후의 테이블에서도 모두 표현되어야 한다.
- 재구성의 원칙: 테이블 분할 전에 제공 가능했던 모든 정보는 분할 후에도 모두 가능해야 한다.
- 유일성의 원칙: 분할하기 이전의 모든 Non-Key 속성은 단지 하나의 테이블에만 존재해야 한다.

Index의 생성 기준

- Primary Key는 DBMS에 의하여 자동적으로 Index 정의됨 (Unique Index)
- Foreign Key의 경우 대부분 Index 정의 필요 (Join 시 필요)
- 좁은 범위 (예:10 ~ 15%)를 접근하는 Query문의 경우 주로 활용
- 단일 컬럼 인덱스 보다 복합 컬럼 인덱스 우선 고려
- 가능한 수정이 빈번하지 않은 컬럼을 대상으로 하며 한 컬럼이 여러 인덱스에 포함되지 않도록 할 것 (특히 동일 컬럼이 여러 인덱스의 선두 컬럼으로 나오는 경우 재검토 필요)
- 결합 인덱스의 경우 컬럼 순서 선정에 주의 (사용빈도, 유일성, 분포도, 정렬순서, 부분범위처리 등을 고려)
- 특별한 경우를 제외하고는 1개의 테이블을 기준으로 인덱스 수가 5개 이하가 바람직
- 새롭게 추가된 인덱스는 기존 Access Path에 영향을 미치므로 주기적인 모니터링과 검증 필요하며 인덱스를 고려한 SQL 작성이 요구됨
- 인덱스와 테이블은 서로 다른 디스크에 저장
- 온라인 응용 프로그램의 성능 향상은 인덱스 활용, 일괄 프로그램의 성능향상은 Cluster 또는 Partition 활용
- 필요 시 Index Organized Table 고려

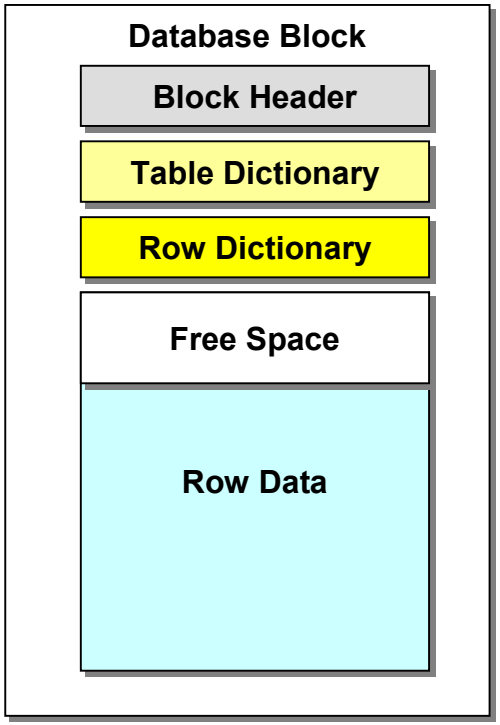
■ 데이터베이스의 **Object**

- **물리 데이터 모델을 적용하여 구축된 실제 데이터가 저장되는 데이터 저장소에 존재**
- **최적의 성능 및 효율적인 저장공간의 활용을 고려한 Object 설계 필요**
- **Schema Objects (Oracle 9i 기준)**
  - ✓ 데이터의 논리적인 구조의 집합을 의미하며 데이터베이스 사용자에 의해 소유되며 SQL에 의해 생성되고 활용된다
  - ✓ Clusters
  - ✓ Constraints
  - ✓ Database links
  - ✓ Database triggers
  - ✓ Dimensions
  - ✓ Index-organized Tables
  - ✓ Indexes
  - ✓ Indextypes
  - ✓ Java classes, Java resources, Java sources
  - ✓ Materialized views
  - ✓ Object tables, Object types, Object views
  - ✓ Packages
  - ✓ Sequences
  - ✓ Stored functions, stored procedures
  - ✓ Synonyms
  - ✓ Tables
  - ✓ Views

- **Non Schema Objects (Oracle 9i 기준)**
- 데이터베이스내에 저장되고 SQL에 의해 생성되고 이용되나 Schema가 아닌 Object
  - ✓ Contexts
  - ✓ Directories
  - ✓ Parameter files (PFILEs) and server parameter files (SPFILEs)
  - ✓ Profiles
  - ✓ Roles
  - ✓ Rollback segments
  - ✓ Tablespaces
  - ✓ Users

■ 데이터베이스 설계의 품질 – 테이블 생성 시 고려사항

- **저장될 테이블 스페이스를 반드시 지정**
- **데이터 Block의 할당을 위하여 저장 파라메터 (Initial, Next, Pctfree, Pctused) 를 고려 (Fragmentation 제거, 공간 관리, 성능 향상)**
- **테이블의 Partition / Cluster 여부 결정**
- **테이블의 크기 계산 방법**
  - ✓ Block Header 크기 계산
  - ✓ Block 당 사용 가능한 데이터 공간 계산
  - ✓ 컬럼 길이의 평균을 구하여 합산
  - ✓ Row의 평균 길이 계산
  - ✓ Block 당 저장 가능한 Row 수 계산
  - ✓ 테이블 생성 시 정의되는 초기 저장공간 (Initial) 크기 계산
  - ✓ 다른 Storage Parameter 결정

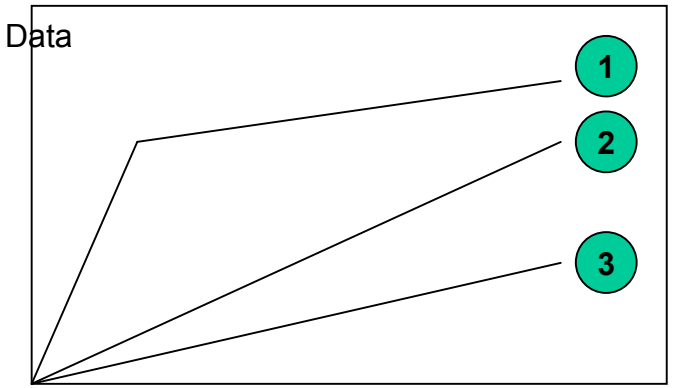


■ 테이블의 크기 계산 방법

- **STEP1-Block Header 크기 계산**
  - ✓ Fixed Header + Variable Transaction Header + Table Dictionary + Row Dictionary
  - ✓ Fixed Header : 57 Byte (기본적으로 요구되는 Block Header 크기)
  - ✓ Variable Transaction Header : INITRANS 파라메터의 수 마다 23 Bytes
  - ✓ Table Dictionary : 4 Bytes (기본 값)
  - ✓ Row Dictionary : 2*Block내의 Row 수
- **STEP2-Block 당 사용 가능한 데이터 공간 계산**
  - ✓ (DB_ Block_Size - [Step-1의 결과]) - (1- PCTFREE / 100)
  - ✓ PCTFREE : 테이블 생성 시 기본 값 10
  - ✓ 많은 NULL 컬럼을 가지거나 길이의 증가가 많이 발생하는 경우 PCTFREE 증가 고려
  - ✓ 길이의 증가가 적고 NULL 컬럼이 적은 경우 PCTFREE 감소 고려 (인덱스의 경우 5이하)
- **STEP3-컬럼 길이의 평균을 구하여 1개 Row의 길이 계산**
  - ✓ Char : 정의된 길이
  - ✓ Varchar2 : 정의된 길이
  - ✓ Date : 7 Byte
  - ✓ Number : 정수길이/2 + 1

■ 테이블의 크기 계산 방법

- **STEP4-Row의 평균 길이 계산**
  - ✓ Row Header (3 Bytes) + (250 Bytes 이하 컬럼 수) + (3* 250 Bytes 이상 컬럼 수 + Step3의 결과
- **STEP5-Block 당 저장 가능한 Row 수 계산**
  - ✓ 한 개 Block에 저장하는 평균 Row 수 = Step2 / Step4
- **STEP6-테이블 생성 시 정의되는 초기 저장공간 (Initial) 크기 계산** 1
  - ✓ 테이블을 생성하기 위해 요구되는 Block 수 : 예상 총 Row 수/Step5 2
  - ✓ 테이블을 생성하기 위해 요구되는 Initial Bytes : 요구되는 Block수 * DB_Block_Size
- **STEP7-다른 Storage Parameter 결정** 3
  - ✓ Next, PCTINCREASE Parameter의 결정 (지나친 Extents가 발생하지 않도록 정의)



① **초기 급속한 증가 후 점차 완만한 증가 (Initial 크게, PCTINCREASE 0 ~ 10%**
② **다량의 데이터가 규칙적으로 증가 (Next 상대적으로 크게, PCTINCREASE 10 ~ 40%)**
③ **소량의 데이터가 점차적으로 증가 (Initial 적제, Next 상대적으로 크게, PCTINCREASE 0~5%)**

- **기준 Extent 회수를 정의(예:20)하여 계산하고 테이블의 특성 및 시스템의 상황을 고려하여 Next 최종 결정**

■ 데이터베이스 설계의 품질 – 인덱스 생성 시 고려사항

- **저장될 테이블 스페이스를 반드시 지정**
- **테이블을 위한 테이블 스페이스와 인덱스 저장을 위한 테이블 스페이스 물리적으로 분리**
- **인덱스의 크기 계산 방법**
  - ✓ Block Header 크기 계산
    - – Fixed Header(113) + Variable Transaction Header (INITRANS 파라메터의 수 마다 23 Bytes)
  - ✓ Block 당 사용 가능한 공간 계산
    - – (DB_ Block_Size - [Step-1의 결과]) - (1- PCTFREE / 100)
  - ✓ 인덱스 컬럼 길이의 평균을 구하여 합산
    - – Not NULL 컬럼 : AVG(NVL(VSIZE(인덱스 컬럼)
    - – NULL 컬럼 : 1
  - ✓ 인덱스 Entry의 크기 계산
    - – Entry Header(2) + (128 Byte 이하 컬럼 수) + (3* 128 Byte 이상 컬럼 수)+평균 인덱스컬럼길이의 합+ROWID
  - ✓ 인덱스 총 소요량 계산
    - – 한 블럭에 저장할 수 있는 인덱스 수 (A)= STEP2 / STEP4
    - – 인덱스 총 소요량 : 1.05 *(NOT NULL Row 수 / A))*2048

Q & A

Contact Info : mykim@b2en.com

데이터베이스관리
- DBA관점의 품질관리

관리데이터

김창갑이사
소프트로드(주)

목차

- 관리데이터의 정의
  - 데이터를 효과적으로 관리, 유지하기 위한 프로세스에서 파생하는 데이터
- 데이터베이스 관리의 다양한 측면

데이터 사용	추세분석	만족도파악	문제해결
데이터 작업	DB 구축	DB 변경	DB이관
데이터 흐름	소스 DB	타겟 DB	매핑
장애	재해	데이터 복구	보안
성능	품질		

- 기대효과
  - 비효율적 사용 예방
  - 정상적 상태 유지

Ⅱ. 사용관리데이터

■ 사용관리데이터의 정의

- 사용자가 데이터를 효과적으로 사용할 수 있도록 지원하고 문제를 해결하는데 필요한 관리 데이터

■ 사용관리데이터의 관리기준

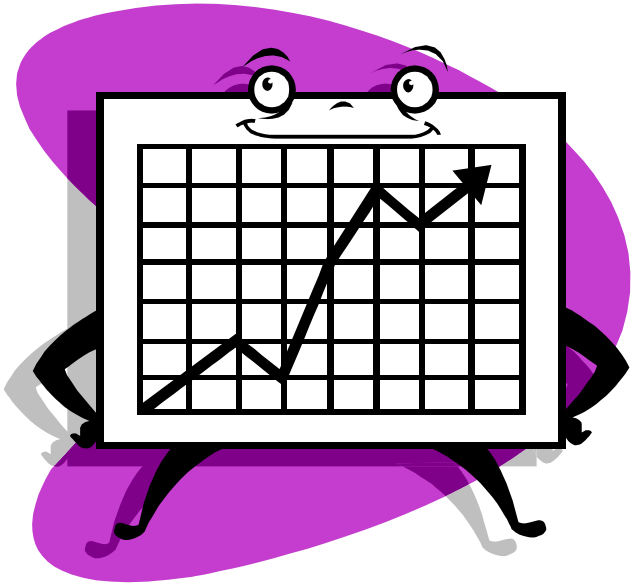
데이터활용도	일별	주별	월별
사용자만족도	불평	방치	
문제해결소요기간	접수	분석	처리

■ 관리방법

- 주기적인 데이터 사용의 집계 및 추세 분석
  - ✓ 급격한 변화와 원인 파악
  - ✓ 예상되는 문제점에 대한 대책 수립
- 사용자 불평 불만에 대한 수집
  - ✓ 내용 분석과 해결
- 문제해결 과정의 관리

Ⅱ. 사용관리데이터 – 사용추세분석

- 추세 분석의 목적
  - 미래의 데이터베이스 시스템 사용 상황에 대하여 예측
  - 안정적이고 효과적인 데이터베이스 시스템을 유지하기 위한 계획 수립
- 데이터 사용의 의의
  - 데이터베이스 시스템 접근
    - ✓ 데이터 검색
    - ✓ 데이터 생성, 변경, 삭제
- 데이터사용에 대한 통계
  - 피크타임의 데이터 접근 ➔ 데이터베이스 성능
    - ✓ 하루 중 피크 타임
    - ✓ 주별, 월별, 계절별 피크 타임
  - 누적 사용량 ➔ 데이터베이스 활용도
    - ✓ 일일, 주간, 월간, 연간 사용량



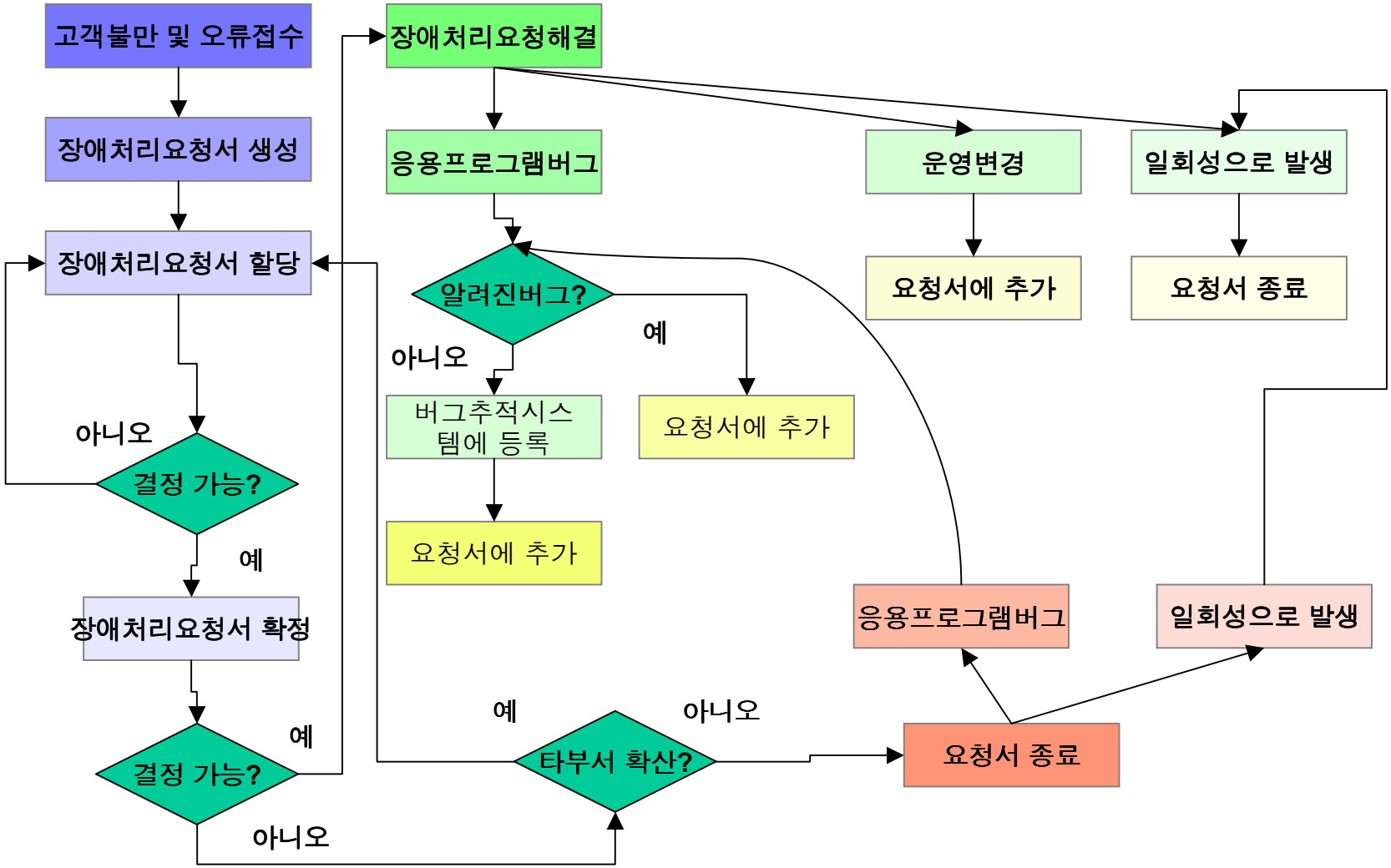
Ⅱ. 사용관리데이터 – 사용자 만족도 제고

- **사용자 불평 불만에 대한 수집**
  - **내용 분석과 해결**
- **문제해결 과정**
  - **체계적인 관리 필요 ➔ SLA 개념 도입**
- **Service Level Agreement (출처, MS SQL Server 2000 Operation Guide)**
  - **데이터베이스 시스템관리자가 제공하리라고 사용자가 기대하는 서비스 수준**
    - ✓ 24 X 7
    - ✓ zero defects
  - **SLA 작성의 고려사항**
    - ✓ 장기적으로 수준 향상
    - ✓ 투입되는 인력의 수 (DBA, 문제 추적 전문가, 중요한 버그 해결사, 버그 테스트 요원, 고객 대변인)
    - ✓ (다양한 고객을 위한 일반적 서비스 수준) + (부가 사항, 예외조항, 특이 상황)
  - **SLA 소요 리소스**
    - ✓ 소프트웨어 ➔ 장애처리시스템, 버그 추적 시스템
    - ✓ DBA 역할 외에 추가로 요구되는 역할 (혹은 인력) ➔ 헬프데스크 관리자, 헬프데스크 요원, 응용프로그램 담당자, 응용프로그램 개발자, 응용프로그램 관리팀, 응용프로그램 테스터, 버그 툴 관리자, 네트워크 관리자, 모니터링 요원

Ⅱ. 사용관리데이터 – SLA 구성요소

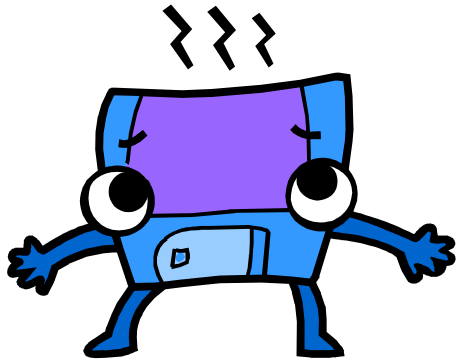
- 유지의 범위
  - 하드웨어, 네트워크, 서버, 응용프로그램
- 제공서비스 수준
  - 응용프로그램, 서비스, 서버들의 정상/비정상 상태, 상태별 조치, 조치에 대한 타이 프레임
- 수준 부합 실패시 확산 절차
  - 자체적으로 문제 해결하지 못한 경우
  - 외부 인력 투입
  - 서비스 수준 인상
- 대응조치의 갱신
  - 기존 대응조치가 불충분한 경우
  - 서비스 수준 유지를 갱신
- 장애처리요청 프로세스
  - 버그추적 프로세스
- 변경관리 프로세스

Ⅱ. 사용관리데이터 – SLA 작성 예시 : 장애처리요청지원



Ⅱ. 사용관리데이터 – 서비스 수준

- 운영상의 상태 (대응조치의 조건)
  - 서버 셧다운
  - 서비스 셧다운
  - 데이터베이스 다운
  - 데이터베이스 객체 다운
  - 복수 또는 단수의 **jobs/tasks/packages** 실패
  - 복수 또는 단일 **transactions** 실패
  - 부정적 지표 발생
    - ✓ UI
    - ✓ back-end 접속 포인트
    - ✓ multi-tier COM 객체
    - ✓ 데이터베이스
    - ✓ 응용프로그램 외부로의 프로세싱 담당 응용프로그램
  - 부정적 지표 없음
- 타임 프레임
  - 실시간
  - 근사 실시간
  - 지연



Ⅲ. 작업관리데이터

■ 작업관리데이터의 정의

- 데이터베이스를 구축하거나 변경하는 작업을 모니터링하는데 필요한 데이터

■ 작업관리데이터의 관리기준

투명성	작업자	작업내용	진행현황
영향도	내용의 중첩	공동자원 사용	네트워크

■ 관리방법

- **프로젝트 관리 데이터베이스**
  - ✓ 기본 정보 추출
  - ✓ 현재 상태 반영 여부 확인
- **현행 데이터베이스 분석**
  - ✓ 주기적인 상태 점검 및 로그
- **영향도 분석과 대책 수립**
  - ✓ 내용상의 중첩으로 데이터 또는 메타데이터의 복사 작업 발생 주기
  - ✓ 하드 디스크, 메모리 등의 자원을 공동으로 사용하는 경우 피크타임 발생 주기
  - ✓ 네트워크 사용의 병목 현상 발생 주기

Ⅲ. 작업관리데이터 – 프로젝트관리 데이터베이스

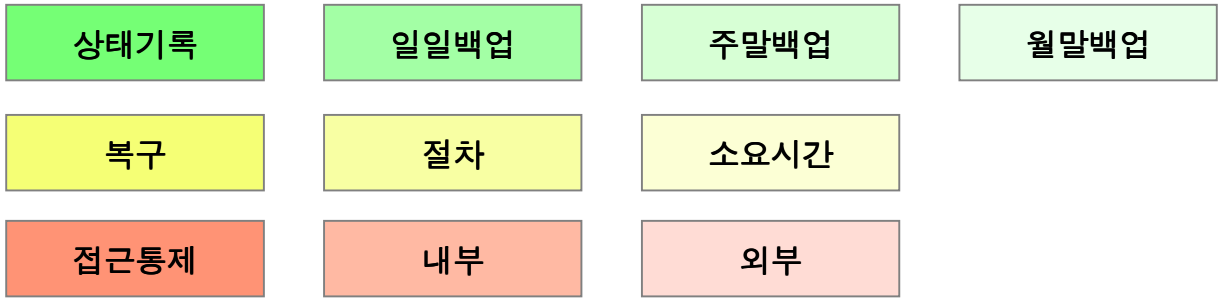
- 프로젝트관리 데이터베이스의 목적
  - 운영계 시스템 또는 분석계 시스템을 개발하는 경우 데이터베이스에 대한 요구사항 분석, 설계, 구현, 테스트 등의 제반 정보를 생성, 유지보수하여 효과적인 데이터베이스 구축에 사용
- 프로젝트관리 데이터베이스의 내용
  - 요구사항 정의서
  - 코드 설계서
  - 데이터베이스 설계서
  - 입력, 출력 설계서
  - 프로그램 명세서
  - 테스트 시나리오
  - 테스트 결과 분석서

Ⅳ. 장해 및 보안관리데이터

■ 장해 및 보안관리데이터의 정의

- 데이터베이스의 정상적인 상태유지와 효과적인 사용을 방해하는 사건을 예방하고, 이러한 사건이 발생할 경우 신속히 복구하는데 필요한 데이터

■ 장해 및 보안관리데이터의 관리기준



■ 관리방법

- 데이터베이스의 중요도 분석
  - ✓ 일일백업 대상, 주말백업대상, 월말 백업 대상 분류
- 백업 절차와 복구 절차
  - ✓ 백업 당시의 상태와 최종 사용 상태 사이의 갭
  - ✓ 복구 절차 (프로세스와 규칙)
- 접근통제

Ⅴ. 성능관리데이터

■ 성능관리데이터의 정의

- 데이터베이스의 성능을 향상하는데 필요한 데이터

■ 성능관리데이터의 관리기준

성능점검	일별	주별	월별
성능향상수단	프로세스	규칙	상용도구
	질의어 최적화	DB 구조 변경	정렬과 재구성

■ 관리방법

- **성능측정기준의 정립**
  - ✓ 측정 방법의 체계화와 측정 주기의 확립
  - ✓ 기준의 정량화
- **성능향상 절차와 규칙**
  - ✓ 프로세스의 정립
  - ✓ 적용규칙의 확립
  - ✓ 상용도구의 사용

Ⅴ. 성능관리데이터 – 성능측정기준

- 성능최적화의 목적
  - OLTP 위주의 운영계 데이터베이스 시스템 : 응답시간(Response Time)의 최소화
  - Batch 위주 의 분석계 데이터베이스 시스템 : 처리시간 (Total Throughput Time) 최소화
- 데이터베이스의 성능최적화 의의
  - 리소스(I/O, LATCH, LOCK) 사용 및 경합 최소화
  - (예시) 자원사용량

LATCH	LIBRARY CACHE	83,786
	ROW CACHE OBJECTS	2,436
	SHARED POOL	41,932

  - library cache : Shared Pool에서 공유 SQL 문에 접근하기 위해 라이브러리 캐시에 액세스하 latch 접근 횟수
  - row cache objects : Shared Pool의 data dictionary에 액세스한 latch 접근 횟수
  - shared pool : Shared pool의 메모리 영역 할달을위해 핵세스한 latch 접근 횟수

Ⅴ. 성능관리데이터 – 성능향상 절차와 규칙

■ 성능향상을 위한 Approach

- 데이터 아키텍처
- 데이터 모델
- 물리설계 (옵티마이징 팩터로 인덱스, 파티션, 클러스터, IOT 등의 설계)
- SQL문 최적화

■ SQL문 최적화를 위한 성능 가이드 (출처, Fujitsu Technology Journal, 2003.11.)

- **절차형 처리방식에서 집합적 처리방식으로 전환하라.**
  - ✓ 과도한 DBMS Call (커서선언, 커서열기, 커서페치, 커서닫기)
  - ✓ loop query
- **리터럴 쿼리(Literal Query)에 의한 파싱 오버헤드(Parsing Overhead)를 줄여라.**
  - ✓ 과도한 파싱의 문제
  - ✓ Softer Parsing, Soft Parsing, Hard Parsing
- **부분범위 처리를 하라.**
  - ✓ 게시글 형태의 경우 사용자 액션(Page Click)이 들어올 때마다 해당 테이블 전체를 읽거나 조건절을 만족하는 모든 대상을 읽고 나서 필요한 부분만 잘라서 보여줌 ➔ 대상 건수가 증가하면 증가할수록 응답속도 저하
- **대량의 데이터에 대한 오퍼레이션(해시, 정렬, 조인)을 위해 적절한 메모리 크기를 사용하라.**
  - ✓ 처리속도의 관점에서 본 하드웨어 자원 : CPU ➔ 메모리 ➔ 디스크 ➔ 네트워크
- **비즈니스 규칙을 로직으로 처리하지 말고 Value화 하라.**

Ⅴ. 성능관리데이터 – 성능튜닝의 각종 이슈

(출처, Productivity Point International)

- alert log , trace file, events의 이해
- index와 SQL tuning 사용
- optimizer의 사용
- SQL문의 대안 작성
- explain plan 사용
- shared pool의 크기 조정
- buffer cache의 크기 조절
- redo log buffer, contention, Java use 등을 위한 SGA 조절
- configuration of databases와 I/O issues
- rollback segments tuning과 redo log
- lock, latch contention의 모니터링
- tuning sort의 이해
- shared server의 tuning
- database storage의 이해와 block의 효과적인 사용
- OS와 resource manager tuning (virtual memory, paging, swaping,thread, ..., etc)
- tuning package 사용

Ⅴ. 성능관리데이터 – 상용도구의 사용

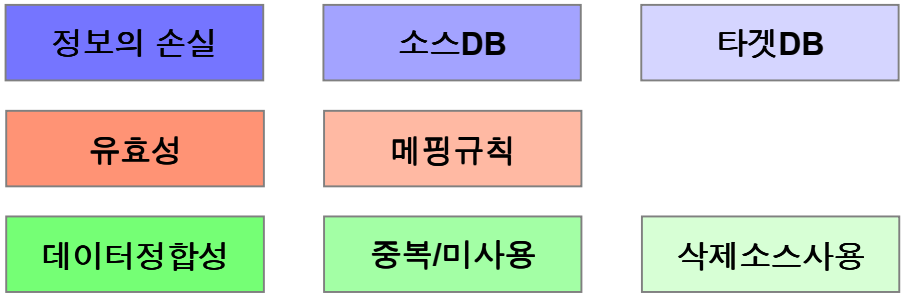
- 상용도구의 기대효과
  - 인력과 시간 절약
  - 관리데이터의 수집과 보관 및 활용
- 상용도구의 적용 분야
  - 데이터 아키텍처 설계
  - 데이터 모델링
  - 데이터베이스 설계
  - 데이터베이스 구현
  - 데이터베이스 시스템 모니터링
  - 역공학
  - 데이터표준화
- 현실적인 문제
  - 가격이 비싸다.
  - 익숙하지 않다.
  - 정확한 용도를 모른다.
    - ✓ 상용도구의 적절한 Mix-n-Match 전략 부재

Ⅵ. 흐름관리데이터

■ 흐름관리데이터의 정의

- 하나의 정보시스템에서 생성되어 사용되는 데이터가 다른 정보시스템으로 이동하여 사용될 때 이 데이터의 소스와 타겟 사이의 매핑을 관리하기 위한 데이터

■ 흐름관리데이터의 관리기준



■ 관리방법

- **정보손실에 대한 확인**
  - ✓ 타겟에서 요구하는 모든 데이터소스의 포함여부
  - ✓ 중복 또는 미사용 정보의 이동
  - ✓ 데이터소스의 존재 여부
- **유효성과 정합성**
  - ✓ 소스와 타겟의 데이터구조 비교
  - ✓ 동일한 변환규칙의 적용
  - ✓ 변환규칙의 데이터 무결성 보장여부

Ⅶ. 품질관리데이터

■ 품질관리데이터의 정의

- 데이터의 전반적인 정합성을 관리하고 데이터 품질을 유지하고 개선하기 위해 지속적으로 수행하는 관리 활동에서 발생하는 데이터

■ 품질관리데이터의 관리기준

주기적 품질검증	측정기준	측정주기	
품질검증 절차	품질검증 규칙		
품질향상수단	품질평가기준	품질향상절차	품질향상규칙

■ 관리방법

- **품질관리에 대한 항목 도출**
  - ✓ Entity integrity
  - ✓ Reference integrity
  - ✓ Domain Integrity
  - ✓ 속성, 컬럼의 비즈니스 룰 적용
  - ✓ 엔터티, 테이블 정의에 따른 데이터 생성, 변경, 삭제 규칙 확인
  - ✓ 트리거를 포함한 사용자정의 DBMS 객체의 작동 여부 확인

참고. 데이터의 구분에 따른 관리 초점

■ 데이터의 구분에 따른 관리 초점

- **운영계 시스템의 데이터베이스**
  - ✓ OLTP에서 발생하는 데이터

사용관리	작업관리	장해 및 보안관리	성능관리
품질관리	흐름관리	…	

- **분석계 시스템의 데이터베이스**
  - ✓ DW, DM에서 발생하는 데이터

흐름관리	사용관리	작업관리	장해 및 보안관리
품질관리	성능관리	…	

Q & A

연락처 : nibble1101@empal.com