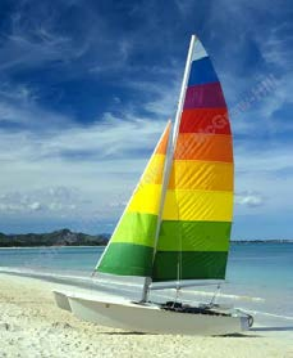


Chapter 22: 객체 기반 데이터베이스

Database System Concepts, 6th Ed.

Chapter 22: Object-Based Databases

- 복합 데이터 타입과 객체 지향성
- SQL에서의 구조 데이터 타입과 상속
- 테이블 상속
- SQL의 배열과 다중 집합 타입
- SQL의 객체 식별자와 참조형
- 객체 지향과 객체 관계형 데이터베이스 비교

SQL에서의 Complex Types

- SQL:1999 에서 다음과 같은 complex type을 지원하도록 확장/도입되었다:
  - 집합(collection type)과 large object types
    - Nested relations은 collection type의 예로 볼 수 있다.
  - 구조체 (structured types)
    - composite attributes와 같은 nested record structures
  - 상속 (Inheritance)
  - 객체 지향성
    - 객체 식별자(object identifiers)와 참조형(references)

- 현재 이 들 기능을 완전히 서포트하고 있는 DB 시스템은 없다.
  - 그러나 몇몇 대표적 상용 시스템은 이 들 기능 중 일부분을 지원하고 있다.
    - 각 데이터베이스 시스템의 매뉴얼을 참조하여야 한다.

SQL의 배열과 다중집합 타입

- 예제:

```
create type Publisher as
    (name          varchar(20),
     branch        varchar(20));
create type Book as
    (title         varchar(20),
     author_array  varchar(20) array [10],
     publisher     Publisher,
     keyword-set   varchar(20) multiset);
create table books of Book;
```

books

title	*author_array*	*publisher*	*keyword_set*
		(*name*, *branch*)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

집단(collection) 값의 생성

- Array 생성

  array [‘Silberschatz’,`Korth’,`Sudarshan’]

- Multisets

  multiset [‘computer’, ‘database’, ‘SQL’]

- Books 릴레이션에 의하여 정의된 튜플 생성법:

  (‘Compilers’, **array**[`Smith’,`Jones’],
  new *Publisher* (`McGraw-Hill’,`New York’),
  multiset [`parsing’,`analysis’])

- Books 릴레이션에 튜플 삽입 방법:

  insert into *books*
  values
  (‘Compilers’, **array**[`Smith’,`Jones’],
  new *Publisher* (`McGraw-Hill’,`New York’),
  multiset [`parsing’,`analysis’]);

```
create type Publisher as
        (name       varchar(20),
         branch     varchar(20));
    create type Book as
        (title      varchar(20),
         author_array  varchar(20) array [10],
         publisher  Publisher,
         keyword-set  varchar(20) multiset);
    create table books of Book;
```

집단 값을 갖는 속성들에 대한 질의

```
create type Publisher as
        (name          varchar(20),
         branch        varchar(20));
    create type Book as
        (title         varchar(20),
         author_array  varchar(20) array [10],
         publisher     Publisher,
         keyword-set   varchar(20) multiset);
    create table books of Book;
```

books

title	*author_array*	*publisher*	*keyword_set*
		(name, branch)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

unnest

parsing
analysis

- 키워드중의 하나에 “parsing”을 가진 모든 책을 찾아라.

  select *title*
  from *books*
  where ‘parsing’ **in** (**unnest**(*keyword-set*))

- 배열의 각 요소를 인덱스를 붙여 액세스할 수 있다:

  select *author_array*[1], *author_array*[2], *author_array*[3]
  from *books*
  where *title* = `Compliers’

집단 값을 갖는 속성들에 대한 질의

```
create type Publisher as
        (name          varchar(20),
         branch        varchar(20));
    create type Book as
        (title         varchar(20),
         author_array  varchar(20) array [10],
         publisher     Publisher,
         keyword-set   varchar(20) multiset);
    create table books of Book;
```

books

title	*author_array*	*publisher*	*keyword_set*
		(name, branch)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

- "title, author_name" 형태의 릴레이션을 구하기 위한 질의:

  select *B.title, A.author*
  from *books* **as** *B*, **unnest** (*B.author_array*) **as** *A* (*author*)

A

author
Smith
Jones

- 배열의 순서화 정보를 유지하기 "**with ordinality" 구문을 이용한다.**

  select *B.title, A.author, A.position*
  from *books* **as** *B*, **unnest** (*B.author_array*) **with ordinality as**
  A (*author, position*)

A

author	*position*
Smith	1
Jones	2

비중첩(Unnesting)

- 중첩 릴레이션을 변환하여 릴레이션 속성(relation-valued attributes)을 대부분 혹은 전부 소거하는 과정을 비중첩 **(unnesting)** 이라고 한다..

```
create type Publisher as
        (name          varchar(20),
         branch        varchar(20));
      create type Book as
        (title         varchar(20),
         author_array  varchar(20) array [10],
         publisher     Publisher,
         keyword-set   varchar(20) multiset);
      create table books of Book;
```

books

title	*author_array*	*publisher*	*keyword_set*
		(*name*, *branch*)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

- **select** *title*, *A* **as** *author*, *publisher.name* **as** *pub_name*, *publisher.branch* **as** *pub_branch*, *K.keyword*
  from *books* **as** *B*, **unnest**(*B.author_array*) **as** *A* (*author*),
  unnest (*B.keyword_set*) **as** *K* (*keyword*)

- 결과 릴레이션 *flat_books*

title	*author*	*pub_name*	*pub_branch*	*keyword*
Compilers	Smith	McGraw-Hill	New York	parsing
Compilers	Jones	McGraw-Hill	New York	parsing
Compilers	Smith	McGraw-Hill	New York	analysis
Compilers	Jones	McGraw-Hill	New York	analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web

중첩 (Nesting)

- **중첩 (Nesting)**은 비중첩의 반대 연산으로 집단 값을 갖는 릴레이션을 생성하는 과정을 말한다. 중첩 연산은 집단 (aggregation) 연산 과정과 유사하나, 집단 연산자 대신에 **collect**() 의 함수를 사용하여 다중집합(multiset)을 생성한다.

title	*author*	*pub_name*	*pub_branch*	*keyword*
Compilers	Smith	McGraw-Hill	New York	parsing
Compilers	Jones	McGraw-Hill	New York	parsing
Compilers	Smith	McGraw-Hill	New York	analysis
Compilers	Jones	McGraw-Hill	New York	analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web

flat_books

books

title	*author_array*	*publisher*	*keyword_set*
		(*name, branch*)	
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

- *flat_books* 릴레이션의 *keyword* 속성을 중첩 시키기:

select *title*, *author*, *Publisher* (*pub_name, pub_branch*) **as** *publisher*,
collect (*keyword*) **as** *keyword_set*
from *flat_books*
groupby *title, author, publisher*

- authors 와 keywords 속성을 중첩 시키기:

select *title*, **collect** (*author*) **as** *author_set*,
Publisher (*pub_name, pub_branch*) **as** *publisher*, **collect** (*keyword*) **as** *keyword_set*
from *flat_books*
group by *title*, *publisher*

Nesting (Cont.)

- 중첩 릴레이션을 생성하기 위한 또 하나의 다른 방법: 4NF 형태의 *books4* 릴레이션에서 출발하여 **select** 절에서 부질의를 이용한다.

```
select title,
        array (select author
                from authors as A
                where A.title = B.title
                order by A.position) as author_array,
        Publisher (pub-name, pub-branch) as publisher,
        multiset (select keyword
                  from keywords as K
                  where K.title = B.title) as keyword_set
from books4 as B
```

authors

title	author	position
Compilers	Smith	1
Compilers	Jones	2
Networks	Jones	1
Networks	Frick	2

keywords

title	keyword
Compilers	parsing
Compilers	analysis
Networks	Internet
Networks	Web

books4

title	pub_name	pub_branch
Compilers	McGraw-Hill	New York
Networks	Oxford	London

books

title	author_array	publisher (name, branch)	keyword_set
Compilers	[Smith, Jones]	(McGraw-Hill, NewYork)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

객체 식별자와 참조형

- 객체 지향 언어에서는 객체를 참조하는 능력을 제공한다.
- *name* 필드와 *Person* 타입에 대한 참조인 *head* 필드를 가진 *Department* 타입을 정의한다. 단, 여기에서 참조는 *people* 테이블의 튜플로 제한한다 (테이블의 튜플에 대한 참조 범위의 제한은 SQL에서 필수 사항이고, 이는 참조를 외래 키처럼 사용하도록 한다):

```
create type Department (
        name varchar (20),
        head ref (Person) scope people)
```

```
create type Person as (
    name Name,
    address Address,
    dateOfBirth date)
    not final
```

- 테이블 정의:

```
create table departments of Department
```

- 타입 정의에 기술된 **scope** 정의를 사용하지 않고, 다음과 같이 테이블 정의문에 **scope** 정의문을 추가할 수 있다:

```
create table departments of Department
    (head with options scope people)
```

- 참조되는 테이블은 각 튜플의 식별자를 저장하는 속성을 가져야 한다. **ref is** 절을 추가함으로써 자기 참조 속성(**self-referential attribute**)이라고 부르는 속성을 선언할 수 있다.

```
create table people of Person
ref is person_id system generated;
```

참조 타입 속성의 초기화

```
create type Person as (
    name Name,
    address Address,
    dateOfBirth date)
    not final
```

```
create type Department (
        name varchar (20),
        head ref (Person) scope people)
```

```
create table departments of Department
```

```
create table people of Person
    ref is person_id system generated;
```

- 참조 값을 가진 튜플을 생성하려면, 우선 null 값을 가진 참조 타입을 생성하고 이후에 질의를 이용하여 참조 값을 설정한다 :

departments

name	head
CS	sys_gen

people

person_id	name	address	dateOf Birth
sys_gen	John	NY	200101

```
insert into departments
        values (`CS', null)
    update departments
        set head = (select p.person_id
                    from people as p
                    where name = `John')
        where name = `CS'
```

사용자 생성 식별자 (User Generated Identifiers)

- 시스템이 생성하는 식별자에 대한 대안으로 사용자가 식별자를 생성할 수 있다.
- 자기 참조 속성은 참조되는 테이블의 타입 정의의 일부분으로 기술되어야 하며, 테이블 정의에서는 참조를 “user generated” 라고 기술하여야 한다.

```
create type Person
    (name varchar(20)
     address varchar(20))
  ref using varchar(20)

create table people of Person
  ref is person_id user generated
```

departments

name	head
CS	01284567

people

person_id	name	address
01284567	John	23 Coyote Run

- 튜플을 생성할 때, 식별자 값을 제공하여야 한다:

```
insert into people (person_id, name, address ) values
  (‘01284567’, ‘John’, `23 Coyote Run’)
```

- 이 경우, *departments* 테이블에 튜플을 삽입하는 과정에서 식별자 값을 직접 사용할 수 있다.
  - 식별자를 추출하기 위한 별도의 질의를 필요로 하지 않는다:

```
insert into departments
values(`CS’, `02184567’)
```

User Generated Identifiers (Cont.)

- 타입 정의에서 **ref from** 절을 포함함으로써 이미 존재하는 주키를 식별자로 사용할 수도 있다:

```
create type Person
      (name varchar (20) primary key,
       address varchar(20))
      ref from (name);
create table people of Person
    ref is person_id derived;
```

- *departments* 테이블에 튜플을 삽입하는 경우, 다음과 같이 수행할 수 있다.

```
insert into departments
    values(`CS',`John')
```

departments

name	head
CS	John

people

person_id	name	address
John	John	23 Coyote Run

경로식 (Path Expressions)

- 참조들은 SQL:1999에서 -> 기호를 통하여 역참조된다.
- 모든 departments 의 부서장 이름과 주소를 찾으시오:

  select *head –>name*, *head –>address*
  from *departments*

- “head–>name” 과 같은 수식을 경로식(**path expression)**이라 한다.
- 경로식은 조인 연산을 피할 수 있는 방법이 되기도 한다.
  - 만약 department의 부서장(head) 속성이 참조 타입이 아니라면 부서장의 주소(address)를 얻기 위하여 *departments* 테이블과 *people* 테이블 사이의 조인 연산이 필요할 것이다.
  - 질의가 매우 간단해진다.

departments

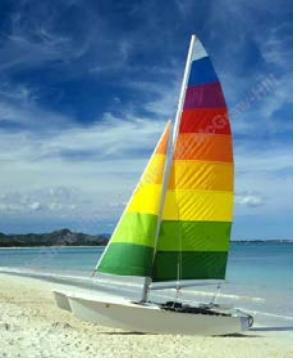
name	head
CS	sys_gen

people

person_id	name	address	dateOf Birth
sys_gen	John	NY	200101

O-O 와 O-R 데이터베이스 비교

- **Relational systems**
  - 단순한 데이터 타입, 강력한 질의어, 높은 보호력.
- **Persistent-programming-language-based OODBs**
  - 복합 데이터 타입, 프로그래밍 언어와의 결합, 높은 성능
- **Object-relational systems**
  - 복합 데이터 타입 , 강력한 질의어, 높은 보호력
- **Object-relational mapping systems**
  - 프로그래밍 언어와 결합된 복합 데이터 타입 지원. 단 기존 릴레이션 데이터베이스 시스템의 상위 레이어로 구현되어 있음.
- Note: 실제의 많은 시스템에서 이 들 경계가 명확하지 않다
  - E.g. 릴레이션 시스템 위에 래퍼로 구현된 영속 프로그래밍 언어는 처음 두 가지 시스템의 장점을 동시에 만족 시키나, 성능이 떨어질 수 있다.



End of Chapter 22