Object-Relational Databases

Object-relational data model
- relational data model 확장 : Object orientation을 포함하는 풍부한 type system 제공
- 첨가된 data type을 다루기 위해 SQL과 같은 relational query를 확장함
- tuple attribute들이 complex type을 갖도록 함

내용
- ***Nested Relational Model :*** 1st NF(Normal Form)에 존재하지 않는 relation과 Hierarchical Structure의 direct representation을 허락함
- ***SQL DDL의 확장 :*** type system과 object orientation을 첨가함으로써 확장
- ***SQL query language의 확장 :*** nested relation과 object를 다루기 위한 SQL query language의 확장
- ***Persistent Programming Language와 Object-relation System의 차이점과 선택을 위한 criteria***

Nested Relations

- ***1NF :*** all attributes have ***atomic domains***
  cf. A set of integers vs. A set of all sets of integetrs
  차이점 : interger들은 subpart를 갖지 않는다.
  Set of integers는 subpart(integer)를 갖는다
  각 integer를 digit들의 ordered list로 고려한다면(즉, 245를 (2,4,5)로 고려) nonatomic 함
  모든 application이 1NF에 일치하는 것은 아님
- ***Nested Relation Model :*** Relational Model의 확장
  . Domain이 atomic이거나 relation일 수 있다. 즉, tuple에서 attribute 값이 relation이 될 수 있다.
  (Relation이 relation들 내에 저장 가능)
  . Complex object가 nested relation의 단일 tuple에 의해 표현 가능함
- OIS(Office Information System)에서의 nested relation의 예 : Document의 구조
  . Document title
  . Author list : nonatomic
  . Date : (day, month, year) 구조의 경우 nonatomic
  . Keyword list : nonatomic

title	author-list	date	keyword-list
		(day, month, year)	
salesplan	{Smith	(1, April, 89)	{profit, strategy}
status r	r{John, Fre	(17, June, 99}	{profit, personnel}

. Nonatomic으로 하기 위해서는 decompiosition이 요구되며(p277,p278의 Fig9.2, Fig9.3 참조)
사용 시에는 join 연산이 요구됨

Complex Type and Object Orientation

- Complex Type system과 Object orientation은 Entity ID, multivalued attributes, generalization, specialization 등의 모델 개념을 relational model로의 복잡한 변환 없이 직접 표현 가능하게 함
- nested relation 특성과 object-oriented 특성을 포함할 수 있도록 하는 **SQL의 확장**이 필요함
  . 여러 확장이 제공되었고 표준화 작업이 진행되고 있음
  . SQL:1999 표준에 근거하여 소개 함

Structured and Collection Types

- OIS 예의 Document(Doc)의 선언 예

```
create type MyString char varying
create type MyDate
     (day integer,
      month chat(10),
      year integer)
create type Document
    (name MyString,
     author-list setof(MyString),
     date MyDate,
     keyword-list setof(MyString))
create table doc of type Document
```

- 위의 특성은 ER diagram의 composite attributes와 multivalued attribute가 직접 표현 가능하게 함
- Collection types(array, multisets 등)을 지원하는 Complex Type system의 예

```
author-array MyString[10]
print-runs multiset(integer)
```

Inheritance

- Inheritancy 선언 예

```
create type Person
     (name MyString,
      social-security  integer)
create type Student
    (degree MyString,
     department MyString)
     under Person
create type Teacher
    (salary integer,
     department MyString)
     under Person
/* ********** conflict 발생 예********** */
create type TeachingAssistant
     under Student, Teacher   // conflict가 발생함(Student와 Teacher의 departmant)

/* ********** conflict 해결 예********** */
create type TeachingAssistant
     under Student with (department as student-dept),
           Teacher with (department as teacher-dept)
```

- Inheritancy 선언 예(table 선언)

```
create table person
     (name MyString,
      social-security  integer)
create table student
    (degree MyString,
     department MyString)
     under Person
create table teacher
    (salary integer,
     department MyString)
     under Person

/* ************************************ */
create table teaching-assistant
     under student with (department as student-dept),
           teacher with (department as teacher-dept)
```

Reference Types

- Type에 대한 attribute는 그 type의 object에 대한 reference가 될 수 있음
  .Ex)
  author-list **setof(ref**(Person)) : a set of references to Person objects
- table의 primary를 사용하여 reference를 구현
  (다른 방법 : table의 각 tuple이 implicit attribute로서 tuple ID를 갖도록 하고 reference는 이 tuple ID임
  subtable들은 묵시적으로 이 table ID를 inherit함

Querying with Complex Types

- Complex type을 다룰 수 있도록 SQL query language를 확장
  . Ex) **select** name, date.year
  from doc

Relation-Valued Attributes

- Example: **create table** pdoc
  (name MyString,
  author-list **setof**(**ref**(people)),
  date MyDate,
  keyword-list **setof**(MyString))
  select name
  from pdoc
  where “database” **in** keyword-list

  select B.name, Y.name
  from pdoc **as** B, B.author-list **as** Y

  select name, **count**(author-list)
  from pdoc

Path Expressions

- dot notation : composite attribute를 참조하기 위한 dot notation은 reference를 나타내기 위해 사용됨
- Example: **create table** phd-students
  (advisor **ref**(people)) // a foreign key of **people** table
  under people

select phd-student.advisor.name // *path expression*
from phd-students

// path expression이 collection이 될 수 있다.

select Y.name
from pdoc.author-list **as** Y

Nesting and Unnesting

- ***Unnesting :*** *a transformation of a nested relation into unnested relation*
  . Nested relation을 nested relation이 없고 attribute로서 structures type이 없는 single flat relation으로의 변환

```
select name, A as author, date.day, date.month, date.year, K as keyword
from pdoc as B, B.author-list as A, B.keyword-list as K
```

- ***Nesting :*** *1NF relation을 nested relation으로 변환하는 과정*

```
/**  keyword 로 relation을 nest함 **/
    select title, author, (day, month, year) as date, set(keyword) as keyword-list
    from flat-doc
    group by title, author, date
/**   **/
    select title, set(author) as author-list, (day, month, year) as date, set(keyword) as keyword-list
    from flat-doc
    group by title, date
```

Functions

- 사용자가 정의한 function을 허용함
- Function은 C, C++과 같은 programming language나 SQL과 같은 DML로 정의가능
- Example :

```
create function author-count(one-doc Document)
    returns integer as
    select count(author-list)
    from one-doc

select name
from doc
where author-count(doc) > 1
```

- Function의 return type이 collection이면 그 결과는 collection임

Creation of Complex Values and Objects

- Complex type을 갖는 relation에 tuple을 삽입하고 갱신하는 방법
- Example : *doc* relation에 tuple (“salesplan”, **set**(“Smith”,”Jones”), (1, ”April”, 89), **set**(“profit”, ”strategy”))
  을 생성

```
insert into doc
values (“salesplan”, set(“Smith”,”Jones”), (1, ”April”, 99), set(“profit”, ”strategy”))
```

- Query에 complex value를 사용 가능

```
select name, date
from doc
where name in set(“salesplan”, “opportunities”, “risks”)  // Complex values
```

- Object을 생성하기 위해 *constructor* function을 사용 가능
  . Object T의 constructor function은 **T()**
  . Constructor function이 호출될 때 초기화되지 않은 type 의 object을 생성하고 **oid** 필드를 채우고
  그 object을 return함
- Update 는 1NF relation의 update와 유사하게 SQL **update** 명령을 사용하여 complex relation의
  update를 수행

Comparison of Object-Oriented and Object-Relational Databases

Object-Relational Database

- ORDB의 SQL language의 선언적 성질과 제한성으로 인한 장점
  . Programming error로부터 데이터를 보호
  . I/O 감소와 같은 고수준의 optimization을 수행함
- Complex data types을 통하여 data modeling과 querying을 더 수월하게 한다.
- application은 멀티미디어 데이터를 포함하여 complex data에 대한 저장과 질의를 포함할 수 있다.

Object-Oriented Databases

- 높은 성능 요구를 갖는 형태의 application에 초점을 맞추고 있다.
- Persistent data에 대해 낮은 overhead를 제공하고, 데이터가 programming language를 사용하여 조작될 경우 data를 변환할 필요가 없다.
- 단점 : programming error로 인한 data corruption
  강력한 query 언어의 부재

각 database system의 일반적 특성

- **Relational systems :** simple data types, powerful query language, high protection
- **Persistent programming language based OODBs :** complex data type, integration with programming, high performance
- **Object-relational system :** complex data type, powerful query languages, high protection