บทที่ 10

Object-Relational Databases Systems

Object-Relational Databases ก็คือฐานข้อมูลที่ถูกสร้างตามแบบของ Object-relational data models ซึ่ง Object-relational data models เป็นตัวแบบที่เพิ่มความสามารถของ relational data โดยการให้ type system ที่เพิ่มขึ้นรวมทั้ง object-oriented เช่น การกำหนด attribute ของ tuples ให้ complex type ได้ แต่มุมมองของข้อมูลยังคงมีรูปแบบในลักษณะของ tables จึงสะดวกสำหรับผู้ใช้ relational databases ที่ต้องการใช้ object-oriented features

10.1 Nested Relations

ใน relational database นิยาม first normal form (1NF) คือ attributes ทั้งหมดต้องมี atomic domain. Domain หนึ่งๆ จะเป็น atomic ถ้า element ของ domain นั้นๆ เมื่อพิจารณาแล้วไม่สามารถแบ่งแยกเป็นหน่วยย่อยได้ ตัวอย่างของการใช้ 1NF ที่อาจมีได้ เช่น ในระบบธนาคาร อย่างไรก็ตามมีงานบางอย่างที่ไม่สามารถเข้ากันได้กับ 1NF หากผู้ใช้มองว่า ฐานข้อมูล คือ set of objects แทนที่จะมองว่า ฐานข้อมูล คือ set of records ซึ่งแต่ละ object อาจต้องการ records มากกว่า 1 record ในการแสดงตัวเอง ดังนั้นจึงควรมีมุมมองแบบ one-to-one ที่เข้ากันได้ระหว่าง user's intuitive notion of an object และ database system's notion of a data item

Nested relational model คือการขยาย relational model ในส่วนของ domains ซึ่งเป็นได้ทั้ง atomic หรือ relation valued ดังนั้น value ของ tuple บน attribute หนึ่งๆ อาจเป็น relation และ relations ก็อาจถูกเก็บไว้ภายใน relations Complex object หนึ่งๆ จึงสามารถแสดงได้โดย single tuple ของ nested relation ดังนั้นหากเรามอง tuple ของ nested relation เป็น data item นั่นคือเรามีความเข้ากันได้ระหว่าง data items และ objects ในมุมมองของผู้ใช้ ฐานข้อมูล แล้ว

ตัวอย่างของ nested relations ต่อไปนี้มาจาก office information systems โดยพิจารณาจาก document-retrieval system ซึ่งเราเก็บข้อมูลของแต่ละ document ดังนี้

- Document title
- Author list
- Data acquired
- Keyword list

หากเรากำหนด relation สำหรับข้อมูลดังกล่าวจะมี domains จำนวนหนึ่งซึ่งเป็น nonatomic domains ดังนี้

title	*Author*

เช่น set หรือ structured เช่น *MyDate* ได้ ดังนั้น composite attributes และ multivalued attributes จาก E-R diagrams จะถูกแสดงโดยตรง

เราสามารถที่จะสร้าง table ได้โดยตรง โดยปราศจากการสร้าง intermediate type สำหรับ table เช่น table *doc* สามารถกำหนดได้ดังต่อไปนี้

create table *doc*

(name MyString,

author-list setof(MyString),

date MyDate,

keyword-list setof(MyString))

Complex type systems มักจะสนับสนุน collection types อื่นๆ เช่น arrays และ multisets (นั่นคือ unordered collections ซึ่ง element อาจมีขึ้นได้หลายครั้ง) ตัวอย่างการกำหนด attribute ต่อไปนี้แสดงถึงการใช้ collection types ดังกล่าว :

author-array MyString[10]

print-runs multiset(integer)

author-array คือแถวลำดับของ author names จาก array นี้สามารถที่จะบอกได้ว่าใครคือผู้แต่งคนแรก ใครคือผู้แต่งคนที่สองหรือสามและอื่นๆ ซึ่งถ้าเป็นชนิด set จะไม่สามารถทำให้เพราะเซตไม่มีการเรียงลำดับ attribute ที่สอง *print-runs* คือ multiset ของเลขจำนวนเต็มใช้สำหรับแสดงจำนวนสำเนาในการพิมพ์แต่ละครั้งของเอกสาร เนื่องจากการพิมพ์สองครั้งอาจมีจำนวนพิมพ์สำเนาเท่ากัน ดังนั้น multiset จึงควรถูกใช้มากกว่าที่จะเป็น set

10.2.2 Inheritance

Inheritance สามารถมีที่ระดับของ types หรือที่ระดับของ tables การพิจารณาแรกเป็นการพิจารณา inheritance ของ type สมมุติว่าเรามีการประกาศ type สำหรับ people :

create type *Person*

(name MyString,

social-security integer)

เราอาจต้องการเก็บข้อมูลพิเศษเกี่ยวกับ people ที่เป็น students และ people ที่เป็น teachers เนื่องจากทั้ง students และ teachers ต่างก็เป็น people เราสามารถใช้ inheritance เพื่อกำหนด student และ teacher types ได้ดังนี้ :

create type *Student*

(degree MyString,

department MyString

under *Person*

create type *Teacher*

(salary integer,

department MyString)

under *Person*

ทั้ง Student และ Teacher inherit attributes ของ Person กล่าวคือ name และ social-security Student และ Teacher ถูกเรียกว่าเป็น subtypes ของ Person และ Person คือ supertype ของ Student รวมทั้ง Teacher ด้วย

หากว่าเราต้องการเก็บข้อมูลของ teaching assistants ผู้ซึ่งเป็นทั้ง students และ teachers ซึ่งอาจอยู่ใน department ที่ต่างกันก็ได้ ถ้าหากระบบของเราสนับสนุน multiple inheritance เราสามารถที่จะกำหนด type สำหรับ teaching assistant ได้ดังนี้ :

create type *TeachingAssisteant*

under *Student, Teacher*

TeachingAssistant ควรที่จะสืบทอด attributes ทั้งหมดของ *Student* และ *Teacher* ปัญหาหนึ่งที่เกิดขึ้นคือ เนื่องจาก attribute *social-security, name* และ *department* ต่างแสดงไว้ใน *Student* และทั้ง *Teacher* ด้วย

Attributes social-security และ name ถูก inherit มาจาก common source, Person ดังนั้นจึงไม่เกิดความขัดแย้งกันอันเนื่องมากจากการ inherit ทั้ง Student และ Teacher อย่างไรก็ตาม attribute department ถูกกำหนดแยกกันทั้งใน Student และ Teacher ซึ่งในความเป็นจริงแล้ว teaching assistant อาจเป็น student ใน department หนึ่ง แต่อาจเป็น teacher ในอีก department หนึ่ง

เพื่อหลีกเลี่ยงความขัดแย้งกันระหว่าง occurrences ของ department เราสามารถเปลี่ยนชื่อโดยใช้ as clause ดังแสดงในตัวอย่างการกำหนด type TeachingAssistant ต่อไปนี้ :

create type *TeachingAssstean*

under *Student* with *(deparment as student-dept),*

Teacher with *(department as teacher-dept)*

CT 316 (S) 161

Teacher with *(department as teacher-dept)*

Inheritance ของ types ควรที่จะถูกใช้อย่างระมัดระวัง ฐานข้อมูลของมหาวิทยาลัยแห่งหนึ่งอาจมีหลาย subtype ของ *Person* เช่น *Student, Teacher, FootbalPlayer, ForeignCitizen* และอื่นๆ Student อาจมี subtypes ของตัวเองเช่น UndergraduateStudent, FraduateStudent และ PartTimeStudent และ person หนึ่งๆ อาจมีสถานะเหล่านี้ได้มากกว่าหนึ่งสถานะในเวลาเดียวกัน

สำหรับแต่ละ entity เราต้องมีการกำหนด type ที่เฉพาะเจาะจง เราอาจจะมีการสร้าง subtype สำหรับทุกๆ combination ที่เป็นไปได้ของ suprtype สำหรับตัวอย่างของเรา เราอาจมี subtypes เช่น ForeignUndergraduate, ForeignGraduateStudentFootballPlay และอื่นๆ โชคไม่ดีที่เราจะมีจำนวน subtypes ของ type Person อย่างมหาศาล

แนวทางพิจารณาที่ดีกว่าในสภาวะแวดล้อมของ database systems คือให้ object หนึ่งมี multiple types (นั่นคือไม่ต้องมี specific type จำนวนมาก) Object-relational systems สามารถสร้างสิ่งนี้โดยการใช้ inheritance ที่ระดับของ table แทนระดับของ types และให้ entity หนึ่งๆ มีใน table มากกว่าหนึ่ง table ในเวลาเดียวกัน เราสามารถแสดงตัวอย่างของวิธีนี้โดยใช้ตัวอย่างของ students และ teachers ขั้นตอนแรก เรากำหนด *people* table ดังนี้ :

```
create table people
        (name MyString,
        soial-security intger)
```

จากนั้นกำหนด *students* และ *teachers* tables :

```
create table student
    (degree MyString,
     department MyString)
     under people
create table teacher
     (salary integer,
     department MyString)
     under people
```

subtables *students* และ *teachers* inherit attributes ทั้งหมดของ table *people* จึงไม่จำเป็นที่จะต้องสร้าง subtables (เช่น *teaching-assistants*) ที่ inherit ทั้ง *students* และ *teachers* นอกจาก

- แต่ละ tuple ของ supertable people สามารถตรงกันกับ tuple (นั่นคือ values มีค่าเดียวกันสำหรับทุกๆ inherited attributes) อย่างมากที่สุดเพียงหนึ่ง tuple สำหรับแต่ละ tables students และ teachers
- แต่ละ tuple ใน students และ teachers จะต้องมีเพียงหนึ่งเท่านั้นที่ตรง tuple ใน people (นั่นคือมีมูลค่าเหมือนกันสำหรับ inherited attributes ทั้งหมด)

หากปราศจากเงื่อนไขแรก อาจจะมี tuples ใน students (หรือ teachers) ที่ตรงกับบุคคลคนๆ เดียวกัน ถ้าหากไม่มีเงื่อนไขที่สอง อาจจะมีหนึ่ง tuple หนึ่งใน student (หรือ teachers) ซึ่งไม่ตรงกับ tuple ใดเลยที่มีอยู่ใน people หรืออาจจะตรงกันหลาย tuples ใน people ซึ่งสถานะเหล่านี้ไม่ถูกต้องกับความเป็นจริงเป็นดังนั้นจึงเป็นข้อผิดพลาด

Multiple inheritance บางทีอาจเกิดขึ้นกับ tables ได้เช่นเดียวกันกับที่เกิดกับ types เช่น teachering-assistant entity เป็นทั้งของ table students และ table teachers อย่างไรก็ตามหากเราต้องการเราสามารถสร้าง table สำหรับ teaching-assistant entity เป็นทั้งของ table students และ table teachers อย่างไรก็ตามหากเราต้องการเราสามารถสร้าง table สำหรับ teaching-assistant entities ดังนี้ :

```
create table teaching-assistens
    under students with (department as student-dept),
        teachers with (department as teacher-dept)
```

ข้อกำหนดความสอดคล้องสำหรับ subtables ต้องรับรองว่า ถ้ามีหนึ่ง entity ถูกแสดงใน teaching-assistants table มันก็จะต้องถูกแสดงใน teachers และ students tables ด้วย

Inheritance ทำให้การกำหนด schema มีความเป็นธรรมชาติมากขึ้น หากไม่มี inheritance ของ tables ผู้ออกแบบ schema จะต้อง link tables ระหว่าง subtables โดยเด่นชัด โดย subtables จะติดต่อกับ supertables ผ่านทาง primary keys และจะต้องกำหนดเงื่อนไขระหว่าง tables เพื่อรับรองถึง referential และ cardinality constraints การใช้ inheritance ทำให้สามารถใช้ functions ที่ได้ถูกกำหนดไว้แล้วของ supertypes บน objects ที่เป็นของ subtypes ได้ และให้ส่วนขยายของ database systems รวมแบบชนิดใหม่ได้

CT 316 (S) 163

10.2.3 Reference Type

Object-oriented languages ให้ความสามารถในการอ้างอิงถึงวัตถุได้ attribute ของ type หนึ่งสามารถอ้างอิงไปยัง object ของชนิดที่ระบุไว้ ตัวอย่างเช่นการอ้างอิงไปยัง people ของ type ref(Person) ซึ่ง author-list field ของ type Document สามารถกำหนดใหม่เป็น

author-list setoff(ref(Person))

ซึ่งเป็นเซตของการอ้างอิงไปยัง Person objects

Tuples ของ table หนึ่งสามารถถูกอ้างอิงได้ด้วยเช่นกัน การอ้างอิงไปยัง tuples ของ table people มี type เป็น ref(people) เราสามารถ implement การอ้างอิงไปยัง tuples ของ table หนึ่งได้โดยการใช้ primary key ของ table ในอีกวิธีหนึ่งแต่ละ tuple ของ table หนึ่งๆ อาจมี tuple identifier เป็นแบบ implicit attribute (attribute ที่ถูกซ่อนไว้) และการอ้างอิงไปยัง tuple หนึ่งๆ สามารถกระทำได้โดยการใช้ tuple identifier ซึ่งเป็นเช่นเดียวกับ attributes อื่นๆ ที่มาจาก super tables

10.3 Querying with Complex Types

ในส่วนของหัวข้อนี้ เป็นการนำเสนอส่วนของ SQL query language เพื่อจัดการกับ complex types เริ่มต้นตัวอย่างคือ : ค้นหาชื่อและปีในการพิมพ์ของแต่ละเอกสาร ซึ่ง query ต่อไปนี้ทำหน้าที่ดังกล่าว

select *name, date.year*

from *doc*

ข้อสังเกต field year ของ composite attribute date ถูกอ้างอิงโดยการใช้ dot notation

10.3.1 Relation-Valued Attributes

extended SQL ให้นิพจน์หนึ่งๆ สามารถประมวลผลไปยัง relation ได้ทุกแห่ง เมื่อมีชื่อของ relation นั้นๆ ปรากฏขึ้น เช่นใน from clause ด้วยการใช้ subexpression อย่างอิสระทำให้สามารถใช้ข้อดีของโครงสร้าง nested relations ได้ดังตัวอย่างต่อไปนี้

สมมุติว่าเราได้ให้ relation *pdoc* ซึ่งมีการสร้าง schema ดังนี้

create table *pdoc*

(name MyString,

author-list setof*(ref(people)),*

dat MyDate,

keyword-list setof*(MyString))*

ถ้าเราต้อการค้นหาเอกสารทั้งหมดซึ่งมีคำว่า "database" เป็นส่วนหนึ่งของ keywords เราสามารถใช้ query ต่อไปนี้ :

select *name*

from *pdoc*

where "database" in *keyword-list*

ข้อสังเกต เราได้ใช้ relation-valued attribute *keyword-list* ในตำแหน่งหนึ่งได้โดย nested relations ไม่ต้องมีการใช้ select-from-where subexpresson อีก

ตอนนี้หากเราต้องการ relation หนึ่งซึ่งมีคู่อันดับในรูป "document-name, author-name" สำหรับแต่ละเอกสารและแต่ละผู้แต่งของเอกสารเราสามารถใช้ query ต่อไปนี้ :

select *B.name, Y.name*

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

เนื่องจาก *author-list* attribute *pdoc* เป็น set-valued field จึงสามารถใช้ใน **from** clause ซึ่งถูกคาดว่าเป็น relation หนึ่ง

Aggregate functions (ตัวอย่างเช่น **min,max** และ **count**) ใช้การรวบรวมของมูลค่าเป็น argument และส่งกลับคืนมูลค่าเดียวเป็นผลลัพธ์ ซึ่ง functions เหล่านี้สามารถประยุกต์ใช้กับ relation-valued expression ได้ เช่น query ค้นหาชื่อและจำนวนของผู้แต่งสำหรับแต่ละเอกสาร สามารถเขียนได้ดังนี้

select *name, count(author-list)*

from *pdoc*

เนื่องจาก *author-list* คือ set-valued attribute ซึ่งบรรจุหนึ่ง tuple สำหรับแต่ละ author ดังนั้น การนับเซตจึงให้จำนวนของผู้แต่ง

10.3.2 Path Expression

dot notation สำหรับการอ้างอิงไปยัง composite attributes สามารถถูกใช้ในการอ้างอิงได้ สมมุติว่าเรามี table people ซึ่งได้นิยามไว้ในหัวข้อก่อนและมี table phd-student ซึ่งกำหนดดังนี้

create table ***phd-students***

(advisor* ref*(people))

under ***people***

จากนั้นเราสามารถใช้ query ต่อไปนี้ค้นหาชื่อของ advisors สำหรับ doctoral students ทั้งหมด

select ***phd-students.advisor.name.***

form ***phd-students***

เนื่องจาก *phd-students.advisor* คือการอ้างอิงไปยัง tuple ใน *people* table, attribute *name* ใน query คือ *name* attribute ของ tuple จาก *people* table การอ้างอิงสามารถใช้เพื่อซ่อน join operations ในตัวอย่างดังกล่าวหากไม่มีการใช้การอ้างอิง *advisor* field ของ *phd-students* จะเป็น foreign key ของ table *people* ในการค้นหาชื่อของ advisor ของ doctoral student เราจะต้องมี explicit joint ของ *phd-students* กับ *people* relation บนฐานข้อมูลของ foreign key การใช้การอ้างอิงสามารถทำให้การเขียน query ง่ายขึ้นมาก

นิพจน์ที่อยู่ในรูป "*students.advisor.name*" ถูกเรียกว่า *path expression* จากตัวอย่างแต่ละ attribute ใน path expression จะมี single value (เป็น reference หนึ่งในกรณีของ advisor) โดยทั่วไป attributes ที่ถูกใช้ใน path expression สามารถเป็น collection ได้ เช่น set ของ multi set หากเราต้องการที่จะได้ชื่อของผู้แต่งทั้งหมดของ documents ใน *pdoc* relation สามารถเขียนเป็น query ได้ดังนี้

select *Y.name*

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

ตัวแปร *Y* มีพิสัยครอบคลุมถึงแต่ละผู้แต่งของแต่ละเอกสารใน *pdoc* relation

10.3.3 Nesting and Unnesting

การแปลง nested relation ไปสู่ 1NF เรียกว่า *unnesting* ,*doc* relation มีสอง attributes คือ *author-list* และ *keyword-list* เป็น nested relations และ สอง attributes, *name* และ *date* ซึ่งไม่เป็น nested relations สมมุติว่าเราต้องการแปลง relation ดังกล่าวไปเป็น flat relation ซึ่งไม่มี nested relations หรือ structured types เป็น attributes เราสามารถใช้ query ต่อไปนี้เพื่อทำงานดังกล่าว :

select name, *A* **as** *author, date.day, date.month, date.year, K* **as** *keyword*

from *doc* **as** *B, B.author-list* **as** *A, B.keyword-list* **as** *K*

ตัวแปร *B* ใน from clause ถูกประกาศมีพิสัยครอบคลุม *doc* ตัวแปร *A* ถูกนิยามมีพิสัยครอบคลุม authors ใน *author-list* สำหรับ document นั้น และ *K* ถูกนิยามมีพิสัยครอบคลุม keywords ใน *keyword-list* ของ document รูปที่ 10.1 ในหัวข้อที่ 1 แสดง instance ของ *doc* relation และรูปที่ 10.2 แสดง 1NF relation ซึ่งเป็นผลของ query ดังกล่าว

กระบวนการในการแปลง 1NF relation ไปสู่ nested relation เรียกว่า *nesting* nesting สามารถทำได้โดยการใช้ส่วนขยายของ grouping ใน SQL ในการใช้โดยปกติของ grouping ใน SQL จะมีหนึ่ง temporary multiset relation จะ (logically) ถูกสร้างสำหรับแต่ละ group และ aggregate function หนึ่งจะถูกประยุกต์ไปบน temporary relation โดยการส่งคืน multiset แทนที่ การประยุกต์ทำให้เราสามารถสร้าง nested relation สมมุติว่าเรามี 1NF relation *flat-doc* ซึ่งแสดงไว้ในรูปที่ 10.2 query ต่อไปนี้ซ้อน relation ไปบน attribute *keyword* :

select *title, author, (day, month, year)* **as** *date, set(keyword)* **as** *keyword-list*

from *flat-doc*

group by *title, author, date*

title	*Author*	*date*	*keyword-list*
		(day, month, year)	
salesplan	Smith	(1, April, 89)	{profit, strategy}
salesplan	Jones	(1, April, 89)	{profit, strategy}
status report	Jones	(17, June, 94)	{profit, personnel}
status report	Frick	(17, June, 94)	{profit, personnel}

Figure 10.4 : A partially nested version of the *flat-doc* relation

ผลลัพธ์ของ query บน *doc* relation จากรูปที่ 10.2 ถูกแสดงในรูปที่ 10.4 ถ้าเราต้องการซ้อน author attribute ด้วยเช่นกัน ด้วยวิธีนี้ เราจะแปลง 1NF table *flat-doc* ในรูปที่ 10.2 เป็น nested table *doc* ดังแสดงในรูปที่ 10.1 เราสามารถทำได้โดย query ดังต่อไปนี้

select *title, set(author)* **as** *author-list, (day, month, year)* **as** *date, set(keyword)* **as** *keyword-list*

from *flat-doc*

group by *title, date*

10.3.4 Function

Object-relational systems ให้ผู้ใช้สามารถกำหนด functions ได้โดย programming language เช่น C หรือ C++ หรือใน data manipulation language เช่น SQL เราจะดูการกำหนด function ใน extended SQL เป็นลำดับแรก

สมมุติว่าเราต้องการ function ซึ่งเมื่อให้ document หนึ่งจะส่งคืนค่าจำนวนของผู้แต่งเอกสาร เราสามารถกำหนด function ได้ดังนี้

create function *author-count(one-doc Document)*

returns integer as

select count(*author-list***)**

from *one-doc*

จากตัวอย่างก่อน *Document* เป็นชื่อของ type โดย function จะถูกเรียกด้วย single document object ซึ่งเป็น argument ของ function และ **select** statement จะประมวลผลกับ relation *one-doc* ซึ่งมี single tuple เท่านั้น ผลลัพธ์ของ **select** statement คือ single value หรือ พูดอย่างถูกต้องคือเป็น tuple ซึ่งประกอบด้วย single attribute ซึ่ง type ถูกแปลงเป็น value หนึ่ง

Function ดังกล่าวสามารถใช้ใน query ซึ่งส่งคืนชื่อของ documents ทั้งหมดซึ่งผู้แต่งมากกว่า 1 คน :

select *name*

from *doc*

where *author-count(doc)*>**1**

ข้อสังเกตใน SQL expression ข้างบนถึงแม้ว่า doc อ้างถึง relation หนึ่งใน **from** clause แต่มันจะถูกจัดโดยนัยเป็นให้ tuple variable ใน **where** clause ด้วยเหตุนี้มันจึงสามารถใช้เป็น argument ไปยัง *author-count* function ได้

โดยทั่วไปแล้ว select statement สามารถส่งคืน collection ของ values ได้ ถ้า return type ของ function คือ collection type ผลลัพธ์ของ function ก็คือ collection ทั้งหมด ถ้า return type ไม่ใช่ collection type ดังตัวอย่างก่อนหน้านี้ collection ซึ่งถูกสร้างด้วย **select** statement จะมีเพียงหนึ่ง tuple ซึ่งจะส่งคืนเป็นคำตอบ ถ้าหากมีมากกว่าหนึ่ง tuple ในผลลัพธ์ของ **select** statement ระบบจะมีทางเลือก 2 แนวทางคือ ระบุว่าสถานการณ์นี้เป็นเหตุผิดพลาดหรือเลือก tuple มาจากบางส่วนของ collection และส่งคืน tuple นั้นเป็นคำตอบ

ระบบฐานข้อมูลบางระบบอนุญาติให้เรากำหนด function ได้โดยใช้ภาษาโปรแกรม เช่น C หรือ C++ function ที่กำหนดในลักษณะนี้จะมีประสิทธิภาพมากกว่า function ที่กำหนดโดยใช้ SQL การคำนวณที่ไม่สามารถทำใน SQL ได้สามารถที่จะกระทำได้โดย function เหล่านี้ ตัวอย่าง เช่น การคำนวณทางคณิตศาสตร์ที่ซับซ้อนโดยใช้ข้อมูลใน tuple หนึ่งๆ

function ที่ถูกกำหนดโดยการใช้ภาษาโปรแกรมและ compile ภายนอกระบบฐานข้อมูลจะต้องถูก load และ execute ด้วย database system code ซึ่งกระบวนการนี้เสี่ยงต่อการเกิด bug ใน program และสามารถทำให้ database internal structure เกิดความเสียหายได้ และจะทำให้อยู่นอกระบบการควบคุมของ database system

10.4 Creation of Complex Values and Objects

ก่อนหน้านี้เราได้เห็นถึงการกำหนด complex type และ query ที่ใช้กับ complex types ขณะนี้เราจะเข้าถึงการสร้าง และปรับปรุง tuples ใน relations ที่มี complex types

เราสามารถสร้างหนึ่ง tuple สำหรับ type ที่ถูกกำหนดโดย *doc* relation ได้ดังนี้ :

("salesplan", **set**("Smith", "Jones"), (1, "April", 80), **set**("profit", "strategy"))

tuple ด้านบนแสดงถึงการเกิดขึ้นของ complex types จำนวนหนึ่ง เราสร้าง value สำหรับ composite attribute *date* โดยตามลำดับของ *day, month, year* ที่อยู่ในวงเล็บ เราสร้าง set-valued attributes *author-list* และ *keyword-list* โดยการระบุถึงสมาชิกภายในวงเล็บซึ่งตามด้วย keyword **set** เราสามารถใช้ complex values constructed ดังที่แสดงไว้ได้ในหลายลักษณะ ถ้าเราต้องการที่จะใส่ tuple ดังกล่าวลงใน relation *doc* เราสามารถปฏิบัติได้ดัง statement ต่อไปนี้

insert into *doc*

values

("salesplan", **set**("Smith", "Jones"), (1, "April", 89), **set**("profit", "strategy"))

เรายังสามารถใช้ complex values ใน quires ตัวอย่างเช่น เมื่อไรก็ตามที่ query มี set เกิดขึ้น เราสามารถระบุเซตได้ดังตัวอย่างต่อไปนี้

select *name, date*

from *doc*

where *name* **in set**("salesplan", "opportiunities", "risks")

query ข้างบนทำหน้าที่ค้นหาชื่อและวันที่พิมพ์ของเอกสารทั้งหมดซึ่งมีชื่อเป็น "salesplan", "opportunityes" หรือ "risks"

เราสามารถสร้าง multiset values ได้เช่นเดียวกับ set values โดยการเปลี่ยนคำว่า **set** เป็น **multiset** list หรือ array values สามารถถูกสร้างได้ในลักษณะเดียวกับ tuple values

ในการสร้างวัตถุขึ้นมาใหม่เราสามารถใช้ *constructor* functions constructor function สำหรับ object T คือ $T()$; เมื่อมันถูกเรียกขึ้นมามันจะสร้าง new initialized object สำหรับ type T แล้วส่งคืน object (ก่อนส่งคืนจะมีการเติม **oid** (object identifier) filed ก่อน) จากนั้น fields ต่างๆ ของ object จะถูกกำหนดค่าต่อไป

เราสามารถ update complex relations ได้โดยใช้ SQL **update** clause ตามปกติ เนื่องจากการ update ต่างๆ มีลักษณะเหมือนกันมากกับการ updates ของ 1NF relations

10.5 Comparison of Object-Oriented and Object-Relational Databases

เราได้ศึกษาถึง object-oriented database ประกอบขึ้นกับสภาวะแวดล้อม persistent programming languages เช่นเดียวกันกับ object-relational database ซึ่ง object-oriented database ถูกสร้างขึ้นบนยอดสุดของ relational model ปัจจุบันระบบฐานข้อมูลทั้งสองประเภทมีอยู่ในท้องตลาด ซึ่ง database designer จำเป็นที่จะต้องเลือกชนิดของระบบที่เหมาะสมกับความต้องการของ application

Persistent extension ของ programming languages และ object-relational systems มีตลาดเป้าหมายที่ต่างกัน ธรรมชาติของการกำหนดและมีพลังที่จำกัด (เมื่อเทียบกับภาษาโปรแกรม) ของ SQL language ให้การป้องกันข้อมูลจากความผิดพลาดในการเขียนโปรแกรมได้ดีและทำให้เกิด optimization ได้ง่ายในระดับสูง เช่น การลด I/O เป็นต้น Object-relational databases ให้ตัวแบบของข้อมูลและการเขียน query ทำได้ง่ายขึ้นโดยการใช้ complex data types application ทั่วไปที่ใช้ฐานข้อมูลระบบนี้ได้แก่พวกที่ต้องการ storage และต้องการ query ของ complex data รวมถึง multimedia data ด้วย

declarative language เช่น SQL มีข้อเสียเปรียบในเรื่องของผลกระทบจำนวนหนึ่งต่อการทำงานของ application ที่ run บน main memory ซึ่งทำให้ต้องมีการ accesses จำนวนมากไปยัง database persistent programming language กำหนดให้ application ในรูปแบบดังกล่าวมีประสิทธิภาพในการทำงานสูง ซึ่งมี overhead ต่ำในการเข้าถึง persistent data และลดความต้องการ

ในการเคลื่อนย้ายข้อมูลหากข้อมูลเหล่านั้นถูกจัดการโดย programming language อย่างไรก็ตาม ความไวในการเสียหายของข้อมูลสูงเนื่องจากการเขียนโปรแกรมที่ผิดพลาดและมักจะไม่มีความสามารถทางด้าน query มากนัก application ที่ใช้กับฐานข้อมูลประเภทนี้ได้แก่พวก CAD databases

ข้อได้เปรียบของระบบฐานข้อมูลประเภทต่างๆ สามารถสรุปได้ดังนี้

- **Relational systems :** แบบชนิดข้อมูลอย่างง่าย query languages ที่มีความสามารถสูง, การปกป้องข้อมูลที่ดี
- **Persistent programming language based OODBs :** แบบชนิดข้อมูลเชิงซ้อน, การรวมเข้ากันกับ programming language, ประสิทธิภาพสูง
- **Object-relation systems :** แบบข้อมูลเชิงซ้อน, query language ที่มีความสามารถสูง, การป้องกันข้อมูลที่ดี

คำพูดเหล่านี้เป็นถูกเพียงในกรณีทั่วไปแต่ต้องคำนึงถึงเสมอว่า database systems บางระบบอาจจะข้ามขอบเขตเหล่านี้ ตัวอย่างเช่น object-oriented database systems ซึ่งรวมเข้ากับสภาวะแวดล้อมของ persistent programming language ถูกทำให้เกิดผลขึ้นที่ระดับบนสุดของ relational database system ระบบบางระบบที่สร้างบน storage system อาจให้ประสิทธิภาพที่ต่ำกว่า object-oriented database systems แต่อาจให้การป้องกันข้อมูลที่แข็งแรงกว่า relational systems อีก

10.6 Summary

Complex types เช่น nested relations มีประโยชน์สำหรับการสร้าง complex data ใน applications จำนวนมาก ซึ่งเป็น relational model ที่ถูกขยายหลังจาก 1NF เพื่อสนับสนุน nonatomic types Object-relational systems ได้รวมเอา complex data ซึ่งมีพื้นฐานอยู่บน extended relational model กับ object-oriented ocncepts เช่น object identity และ inheritance SQL-data definition และ query languages ได้ถูกขยายเพื่อจัดการกับ complex type และการปรับให้เข้ากับ object เราได้เห็นถึง features ต่างๆ ของ extended data definition language เช่นเดียวกัน query language และส่วนที่เพิ่มขึ้นโดยเฉพาะ set-valued attributes, inheritance, object และ tuple references.

ตัวอย่างของ object-Relational Database Systems ที่ได้รับความนิยมในขณะนี้คือ ORACLE 8.05 for MICROSOFT WINDOWS NT ซึ่งผลิตภัณฑ์ของ ORACLE จะเป็น ORDBSs ตั้งแต่ version 8 เป็นต้นไป สำหรับ commercial DBMSs อื่นๆ ซึ่งมี object-relational support ได้แก่ Informix Universal Server, IBM DB/2 C2 V2 และ UniSQL

1. ตัวอย่างของ object-relational systems (Motivating example)

ในบทนี้เป็นตัวอย่างของระบบฐานข้อมูลที่มีความต้องการใช้ object-relation systems ซึ่งจะเป็นปัญหาของ new business data processing โดยจะเน้นในส่วนของ intangible products นั่นก็คือ video และ video และ audio

เราจะพิจารณาถึงบริษัทการ์ตูน Dinky Entertainment Company ซึ่งเป็นส่วนหนึ่งของผู้ร่วมก่อตั้ง Hollywood โดยทรัพย์สินส่วนใหญ่ได้มาจากการจำหน่ายภาพยนตร์และลิขสิทธิ์ของตัวการ์ตูนและโดยเฉพาะละครที่เป็นที่ชื่นชอบของคนส่วนใหญ่ คือ Herbert the Word ซึ่ง Dinky มี film ของละครเรื่อง Herbert the Worm ฉายอยู่เป็นจำนวนมากทั่วโลก Dinky ทำเงินได้มาก จากการจำหน่ายลิขสิทธิ์ของ Herbert's image, voice และส่วนหนึ่งของ video เพื่อใช้ในจุดประสงค์ต่างๆ เช่น action figures, video games, การจดทะเบียนผลิตภัณฑ์และอื่นๆ ฐานข้อมูล ของ Dinky ถูกใช้เพื่อในเรื่องของการขายและสัญญาต่างๆ ที่ของผลิตภัณฑ์ที่เกี่ยวข้องกับ Herbert รวมทั้ง video และ audio data ที่สร้างในฟิล์มเรื่อง Herbert

1.1 แบบข้อมูลชนิดใหม่ (New Data Types)

ปัญหาที่ Dinky's database designers พบคือความจำเป็นในการสนับสนุนตัวแบบข้อมูลซึ่งมากกว่าที่อยู่ใน relation DBMS นั่นคือ :

- **User-defined abstract data types (ADTs) :** เรามีความต้องการ special functions เพื่อจัดการกับข้อมูลจำพวก image, voice, video footage ตัวอย่างเช่น เราอาจเขียน functions ซึ่งทำการสร้าง compressed version ของ image หรือ lower-resolution image
- **Constructed types :** ในตัวอย่าง เราต้องการ new types ที่สร้างจาก atomic types โดยใช้ constructors สำหรับการสร้าง sets, tuples, arrays, sequences และอื่นๆ
- **Inheritance :** เมื่อขนาดของ data types เพิ่มขึ้น การจดจำถึงลักษณะของ types ต่างๆ จึงเป็นสิ่งจำเป็นสำหรับการนำ types เหล่านี้มาใช้ ตัวอย่างเช่น compressed images และ lower-resolution images (ในบางระดับ) ทั้งคู่ต่างก็คือรูปภาพ ดังนั้น

จึงเกิดการ *inherit* features ของ image objects ในขณะที่มีการกำหนด (หรือ จัดการในภายหลัง) compressed image objects และ lower-resolution image objects รูปต่อไปนี้แสดง SQL3 DDL statements สำหรับส่วนต่างๆ ของ Dinky's ORDBMS schema

1. **CREATE TABLE** Frames
   (*frameno* **integer,** *image* **jpeg_image,** *category* **integer**);
2. **CREATE TABLE** Categories
   (*cid* **integer**, *name* **text,** *lease_price* **float**, *comments* **text**);
3. **CREATE ROW TYPE** theater_t
   (*tno* **integer**, *name* **text,** *address* **text,** *phone* **text**);
4. **CREATE TABLE** Theaters **OF TYPE theater_t WITH IDENTITY;**
5. **CREATE TABLE** Nowshowing
   (*film* **integer,** *theater* **ref**(theater_t), *start* **date**, *end* **date**);
6. **CREATE TABLE** Films
   (*filmno* **integer**, *title* **text**, *stars* **setof**(text), *director* **text,** *budget* **float**);
7. **CREATE TABLE** Countries
   (*name* **text,** *boundary* **polygon**, *population* **integeter,** *language* **text**);

Figure 1 : DDL Statement for Dinky Schema

1.2 การจัดการกับแบบข้อมูลชนิดใหม่ (Manipulating the New Kinds of Data)

ตัวอย่างแรกของเรามาจาก Clog breakfast cereal company (พวกเขาทำอาหารธัญพืชที่เรียกว่า Delirios) มีความต้องการเช่ารูปภาพของ Herbert the Worm ในขณะที่พระอาทิตย์ขึ้นหนึ่งรูปเพื่อประกอบเข้ากับการออกแบบกล่องของ Delirios Query สำหรับแสดงหนึ่ง collection ของ images และ lease prices ที่เป็นไปได้ สามารถเขียนในลักษณะของ SQL ได้ดังนี้

SELECT F.frameno, thumbnaail(F.imag), C.lease price
FROM Framxs F, Categorics C
WHERE F.category = C.cid AND is_sunrise(F.image) AND is_herberrt(F.image)

Dinky ยังมี methods จำนวนหนึ่งซึ่งถูกเขียนใน imperative languages (เช่น C) เพื่อใช้กับ

database system โดย methods เหล่านี้สามารถนำไปใช้ใน queries ได้ลักษณะเดียวกันกับที่เป็น built-in methods โดยการใช้งานก็เหมือนกับ SQL ใน relational language จากตัวอย่าง thumbnail method ใน Select clause ใช้สำหรับสร้าง small version ของ full-size input image สำหรับ is_sunrise method คือ boolean function ซึ่งใช้วิเคราะห์ image ถ้าหากภาพนั้นเป็นรูปพระอาทิตย์ขึ้นจะส่งค่า true กลับมา ส่วน is herbert ก็เช่นเดียวกันจะส่วนค่า true กลับมาถ้ารูปนั้นเป็นรูปของ Herbert ซึ่ง query นี้จะให้ frame code number, image thumbnail และ price สำหรับทุก frames ที่มีรูปของ Herbert และพระอาทิตย์ขึ้น

ตัวอย่างที่สองมากจาก Dinky's executives พวกเราทราบว่า Delirios เป็นที่นิยมในรัฐเล็กๆ ใกล้กับ Andorra ดังนั้นพวกเขาจึงต้องการแน่ใจว่าจะมี Herbert films ถูกฉายอยู่ที่โรงภาพยนตร์ใกล้ Andorra เมื่ออาหารเช้าวางตลาด เพื่อตรวจสอบสถานการณ์ในปัจจุบันผู้บริหารจึงต้องการชื่อของโรงภาพยนตร์ทั้งหมดที่กำลังฉาย Herbert films ภายใน 100 กิโลเมตรของ Andorra สามารถเขียนในลักษณะของ SQL ได้ดังนี้

```
SELECT N theater -> name, N.theater -> address, F.title
FROM  Nowshowing N, Films F,Countries C
WHERE  N.film – F.filmno AND
        Overlas(C.boundary, radius(N.theater -> address, 100)) AND
        C.name = 'Andorra' AND 'Herbert the Worm' ∈ F.staris
```

Extended SQL เพื่อค้นหา Herbert Films ที่ฉายใกล้กับ Andorra

theater attribute ของ Nowshowing table คือ reference ไปยัง object ใน table อื่น ซึ่งมี attrbute *name, address* และ *location* การอ้างอิงนี้สามารถทำได้ในอีกลักษณะโดยการใช้*สัญลักษณ์ N.thearer -> name* และ *N.theater -> address* แต่ละส่วนอ้างอิงถึง attributes ของ theater_t object ซึ่งถูกอ้างอิงใน Nowshowing rowN สำหรับ stars attribute ของ film table คือเซตของชื่อต่างๆ สำหรับแต่ละ film's stars สำหรับ radius methods จะส่งคืนวงกลม(ที่มีรัศมีอยู่ในช่วงของ second argument) โดยผ่านทาง first argument และ overlaps method จะทดสอบ spatial overlap (การซ้อนทับของพื้นที่กับวงกลมดังกล่าว) จากนั้น Nowshowing และ Films จะถูก join กันโดย equijoin clause ในขณะที่ Nowshowing และ Countries จะถูก join กันโดย spatial overlap clause

CT 316 (S) 175

จากตัวอย่างสอง object-relational queries คล้ายกันกับ SQL-92 queries แต่มีลักษณะเฉพาะบางอย่างที่ต่างกันคือ

- **Operators for constructed types :** สำหรับ constructed types ที่มีใน data model, ORDBMSs ให้ natural methods สำหรับ types เหล่านี้ ตัวอย่างเช่น setof type มี standard set methods คือ $\in, \ni, \subset, \subseteq, =, \supseteq, \supset, \cup, \cap$ และ – (รายละเอียดอยู่ในหัวข้อที่ 3.1)
- **Operators for reference types :** Refernc types สามารถถูก *deference* ได้โดยการใช้ arrow (->) notation (รายละเอียดอยู่ในหัวข้อที่ 4.2)
- **User-defined methods :** User-defined abstract types ถูกดูแลโดยการใช้ methods ของตัวเอง ตัวอย่างเช่น *is_herbert* (รายละเอียดอยู่ในหัวข้อที่ 2)

2. แบบชนิดข้อมูลเชิงนามธรรมที่ผู้ใช้กำหนด (User-defined abstract data types)

พิจารณารูปที่ 1 จากหัวข้อที่ 1 มี column image ของ type jpeg_image ซึ่งเก็บ compressed image ซึ่งใช้แสดงหนึ่ง frame ของหนึ่ง film โดย jpeg_image type ไม่ได้เป็นส่วนหนึ่งของ DBMS built-in types และได้ถูกกำหนดโดย user สำหรับ Dinky application เพื่อเก็บ image data ที่ถูก compress โดยการใช้มาตรฐาน JPEG อีกตัวอย่างหนึ่งคือ type polygon ซึ่งถูกกำหนดในบรรทัดที่ 7 ซึ่งประกอบด้วยการแสดงรูปร่างและขนาดของประเทศที่มีอยู่บนแผนที่โลก

การให้ผู้ใช้สามารถกำหนด data types ชนิดใหม่ๆ ได้คือลักษณะสำคัญของ ORDBMSs จากตัวอย่าง DBMS ให้ผู้ใช้จัดเก็บและค้นคืน objects ของ type jpeg_image ได้เช่นเดียวกับ object ของ type อื่นๆ (ตัวอย่างเช่น integer) โดย atomic data types ชนิดใหม่ต้องมี type-specific operations ซึ่งกำหนดโดย user ผู้ซึ่งสร้างมัน ตัวอย่างเช่น มีผู้ต้องการกำหนด operation บน image data type ซึ่งได้แก่ compress, rotate, shrink และ crop การรวมกันของ atomic data type และ method ของมันจะถูกเรียกว่า abstract data (ADT) SQL แบบเก่าจะมีเฉพาะ built-in ADTs ตัวอย่างเช่น integer (ซึ่งรวมมากับ arithmetic methods) หรือ string (ซึ่งมี equality, comparison และ LIKE methods) ซึ่ง Object-relational systems มี ADTs เหล่านี้และยังให้ผู้ใช้สามารถกำหนด ADTs ที่เป็นของตนเองได้

คำว่า "abstract" ถูกใช้กับแบบชนิดข้อมูลเหล่านี้เพราะว่า database system ไม่จำเป็นที่จะต้องทราบว่า ADT's data ถูกเก็บอย่างไรหรือ ADT's methods ทำงานอย่างไร DBMSs ต้องการ

ทราบแต่เพียงว่ามี methods ไหนที่ใช้ได้บ้างและ input หรือ output types ของ methods การซ่อนส่วนในของ ADT ถูกเรียกว่า encapsulation (บาง ORDBMSs อ้างถึง ADTs ว่า opaque types เพราะถูก encapsulate ดังนั้นจึงไม่มีใครมองเห็นรายละเอียด) ข้อสังเกตุคือแม้แต่ relation system, atomic types อย่างเช่น integers ก็มี methods ที่ถูก encapsulate รวมอยู่ใน ADTs ด้วยในกรณีของ integers มี standard methods คือ arithmetic operators และ comparators ในการประมวลของ adding operator บน integers, database system ไม่จำเป็นต้องเข้าใจถึงกฎของการบวก มันเพียงแต่จำเป็นต้องรู้ว่าจะทำอย่างไรจึงจะเรียก code ของ addition operator ได้และ type ของ data ที่ส่งคืนเป็นอย่างไร

ใน object-relational systems ความชัดเจนของ encapsulation คือสิ่งสำคัญเพราะมันซ่อนความแตกต่างที่แท้จริงระหว่าง data types และยังให้ ORDBMS สามารถถูกสร้างได้โดยไม่ต้องแยก type และ methods ที่ user ต้องการเพิ่มเติมออกจากกัน ตัวอย่างเช่นการบวกเลขจำนวนเต็มและการ overlay images สามารถถูกปฏิบัติได้ในรูปแบบเดียวกันของระบบ ซึ่งมีข้อแตกต่างก็เพียง code ที่ถูกเรียกขึ้นมาสำหรับสอง operations นี้ และค่าของวัตถุที่ส่งคืนจะต่างกัน

2.1 การกำหนดการทำงานสำหรับข้อมูลนามธรรม (Defining methods of an ADT)

อย่างน้อยที่สุด สำหรับแต่ละ new atomic type ผู้ใช้จะต้องกำหนด methods ที่ให้ DBMS สามารถทำการอ่านและ output objects ของ type นี้ได้และการคำนวณพื้นที่ในการจัดเก็บเพื่อเก็บ object ผู้ใช้ซึ่งสร้าง new atomic type จะต้องบันทึก (register) methods เหล่านี้ให้กับ DBMS :

- **size :** จะส่งค่าจำนวนของ bytes ที่ใช้ในการจัดเก็บ item ของ type หรือ special value *variable* ถ้า items มีขนาดไม่เท่ากัน
- **import :** สร้าง new items สำหรับ type นี้ได้จาก textual inputs (ตัวอย่างเช่น **INSERT** statements)
- **export :** Maps items สำหรับ type นี้ให้อยู่ในรูปแบบที่เหมาะสำหรับแสดงผลหรือสำหรับใช้ใน application program (ตัวอย่างเช่น ASCCII string หรือ file handle) ในการบันทึก new method สำหรับ atomic type, users จะต้องเขียน code สำหรับ method แล้วจึงบอก database system เกี่ยวกับ method สำหรับ code ที่เขียนขึ้นอยู่กับ DBMS และ Operating system ที่ให้การสนับสนุน ตัวอย่างเช่น ORDBMS อาจมีส่วนจัดการ C code สำหรับ Linux operating system ในกรณีนี้ method code จะต้องเขียนอยู่ในรูปของ C และต้อง compiled ไปสู่ object file ที่เก็บบน Linux file system จากนั้น SQL-

style method registration command จะถูกส่งให้แก่ ORDBMS เพื่อให้มันยอมรับ new method ดังตัวอย่างต่อไปนี้ :

CREATE FUNCTION is_sunrise(jpeg_image) **RETURNS** boolean

AS EXTERNAL NAME '/a/b/c/dinky.o' ;

Statement นี้จะกำหนดรูปแบบเฉพาะของ method ได้แก่ : type ที่รวมมากับ ADT, return type, location ของ code และเมื่อ method ถูกบันทึกแล้ว DBMS จะใช้ dynamic linking ของ operating system เพื่อ link method code เข้าสู่ระบบฐานข้อมูลจึงจะสามารถถูกเรียกขึ้นมาทำงานได้ ตัวอย่างต่อไปนี้เป็นคำสั่งสำหรับบันทึก methods จำนวนหนึ่งเข้าสู่ Dinky database.

1. **CREATE FUNCTION thumbnail(jpeg_image) RETURNS jpeg image**

   AS EXTERNAL NAME '/a/b/c/dinky.o';

2. **CREATE FUNCTION** is_sunrise(jpeg_image) **RETURNS** boolean

   AS EXTERNAL NAME '/a/b/c/dinky.o';

3. **CREATE FUNCTION** is_herbert(jpeg_image) **RETURNS** boolean

   AS EXTERNAL NAME '/a/b/c/dinky.o';

4. **CREATE FUNCTION** radius(polygon, flat) **RETURNS** polygon

   AS EXTERNAL NAME '/a/b/c/dinky.o';

5. **CREATE FUNCTION** overlaps(polygon, polygon) **RETURNS** boolean

   AS EXTERNAL NAME '/a/b/c/dinky.o';

Method Registration Commands for the Dinky Database

Type definition statements สำหรับ user-defined atomic data types ใน Dinky schema แสดงดังตัวอย่างต่อไปนี้

1. **CREATE ABSTRACT DATA TYPE** jpeg_image

   (*internallength* = **VARIABLE,** *input* = jpeg_in, *output* = jpeg_out);

2. **CREATE ABSTRACT DATA TYPE** polygon

   (*internallength* = **VARIABLE,** *input* = poly_in, *output* = poly_out);

Atomic Type Declaration Commands for Dinky Database

3. ชนิดที่ถูกสร้าง (Constructed types)

Atomic types และ user-defined types สามารถรวมกันเพื่ออธิบายโครงสร้างที่ซับซ้อนขึ้น โดยการใช้ตัวสร้างชนิด **(type constructors)** ตัวอย่างเช่น บรรทัดที่ 6 ของการกำหนด schema ในหัวข้อที่ 1 มีการกำหนด column *stars* ของ type setof(text); แต่ละการบันทึกใน column นี้คือ set ของ text string บอกถึงผู้แสดงใน film สำหรับ setof syntax คือตัวอย่างหนึ่งของ type constructor สำหรับ common type constructors อื่นๆ ประกอบด้วย :

- row($n_1t_1,n_2t_2,...,n_nt_n$) : ชนิดนี้แสดงถึงแถวหนึ่งแถว (ซึ่งก็คือ tuple นั่นเอง) ของ n fields ซึ่ง fields $n_1,n_2,...,n_n$ มี type เป็น $t_1,t_2,...,t_n$ ตามลำดับ
- listof(base) : เป็น type ที่แสดงถึงลำดับของ base-type items
- array0f(base) : เป็น type ที่แสดงถึง array ของ base-type items
- setof(base) : เป็น type ที่แสดงถึงเซตของ base-type items ซึ่ง set ไม่สามารถเก็บ elements ซ้ำกันได้
- bagof(base) : เป็น type ที่แสดงถึง *bag* หรือ *multiset* ของ base-type items ซึ่ง bag เป็น unordered collection เช่นเดียวกับ set แต่สามารถเก็บข้อมูลซ้ำกันได้ จำนวนของข้อมูลที่ซ้ำกันเป็นสิ่งสำคัญ

ตัวอย่างเช่น {a, b, b} และ {b, a, b} แสดงถึง bag ที่เหมือนกันแต่ต่างจาก bag {a, a, b} type constructors สามารถใช้ประกอบกันได้ ตัวอย่างเช่น setof(arrayof(integer)) Type ที่ถูกกำหนดโดยการใช้ type constructors จะถูกเรียกว่า **constructed types** โดยจะใช้ listof, arrayof, bagof, setof เป็น type constructor นอกสุดซึ่งในบางครั้งถูกอ้างถึงเป็น collection types หรือ bulk data types

3.1.1.1 การจัดการกับข้อมูลของชนิดที่ถูกสร้าง (Manipulating data of constructed types)

DBMS ได้ให้ built-in methods สำหรับ types ที่สนับสนุนโดย type constructors โดย methods เหล่านี้ต่างเป็น built-in operations เช่นเดียวกับการบวกหรือการคูณของ atomic type เช่น เลขจำนวนเต็ม ในหัวข้อนี้เราจะกล่าวถึง methods สำหรับ type constructors และจะอธิบายถึง SQL queries ว่าสามารถสร้างและจัดการกับ values ของ constructed types ได้อย่างไร

Built-in operators for constructed types

ขณะนี้เราจะพิจารณาถึง built-in operators สำหรับแต่ละ constructed types ซึ่งได้แสดงในหัวข้อที่ 3 ซึ่งมีดังนี้ :

- **Rows :** ถ้าให้หนึ่ง item *i* ของ type row($n_1t_1,n_2t_2,...,n_nt_n$), field extraction method ที่ให้เราสามารถเข้าถึงในแต่ละ field n_k ได้โดยคือ dot notation *i* n_k ถ้ามีสอง constructors ถูกซ้อนอยู่ใน type definition เราสามารถใช้ dot เพื่อเข้าถึง field ใน nested row ได้ ตัวอย่างเช่น $i.n_k.m_l$ ถ้าหากเรามีหนึ่ง collection ของ rows, dot notation จะให้หนึ่ง collection เป็น
- ผลลัพธ์แก่เรา ตัวอย่างเช่น ถ้า *i* เป็นหนึ่ง list ของ rows, $i.n_k$ จะให้หนึ่ง list ของ items จาก type t_n; ถ้า *I* คือหนึ่งเซตของ rows, $i.n_k$ จะให้หนึ่งเซตของ items จาก type t_n

Dot ที่ซ้อนกันมักถูกเรียกว่า **path expression** เพราะมันทำให้ทางผ่านไปยัง nested structure

- **Sets และ multisets :** Set objects สามารถเปรียบเทียบกันได้โดยใช้ set methods แบบเดิม คือ $\subset,\subseteq,=,\supseteq$ หนึ่ง item ของ type **setof**(foo) สามารถถูกเปรียบเทียบกับหนึ่ง item ของ type foo โดยใช้ method ดังตัวอย่างของ query ที่สองในหัวข้อ 1.2 ซึ่งมีการเปรียบเทียบ 'Herbert the Worm' F.stars สองเซต objects (ที่มี elements type เดียวกัน) สามารถถูกรวบรวมกันเพื่อสร้าง object ใหม่ได้โดยใช้ $\cup$ และ – operators

แต่ละ methods สำหรับ sets สามารถถูกกำหนดให้กับ multisets ได้, โดยการนับจำนวนชุด elements ที่นับแล้วไว้เป็นบัญชี ซึ่งสำหรับ $\cup$ operation จะเพิ่มจำนวนที่เป็นชุดของหนึ่ง element ส่วน $\cap$ จะนับจำนวนครั้งที่พบหนึ่ง element อยู่ในสองเซตของ input multisets และ – จะลดจำนวนครั้งของการพบหนึ่ง element ใน multiset ที่สองจากจำนวนครั้งของการพบหนึ่ง element ใน multiset ที่หนึ่ง ตัวอย่างเช่น $\cup$ ({1,2,2,2}, {2,2,3} = {1,2,2,2,2,2,3}; $\cap$ ({1,2,2,2}, {2,2,2}) = {2,2}; และ –({1,2,2,2}, {2,2,3}) = {1,2}

- **List :** Traditional list operations ซึ่งประกอบด้วย head ซึ่งส่งคืน element ตัวแรก ; tail จะส่งคืน list โดยการย้าย first element; cons, ซึ่งนำหนึ่ง element มาแล้วใส่โดยจัดให้เป็น first element ในหนึ่ง list; และ append ซึ่งจะเพิ่มหนึ่ง list ไปยัง list อื่นๆ
- **Others :** operators ที่แสดงไว้ด้านเป็นตัวอย่างหนึ่ง เรายังมี aggregate operators count, sum, avg, max และ min ซึ่งสามารถ (โดยหลักการ) ถูกใช้ไปยังวัตถุใดๆ ของ collection type หนึ่งๆ ได้ Operators สำหรับ type conversion ก็จัดเป็นแบบสามัญด้วย ตัวอย่าง

เช่น เราสามารถให้ operators ในการเปลี่ยน multiset object เป็น set object โดยการลดความ ซ้ำซ้อนลง

หมายเหตุ : collection types ถูกรวมไว้ใน drafts ของ SQL3 standard ส่วนสนับสนุนเต็มอาจจะพบในส่วนมาตรฐานย่อยของ SQL4

4. วัตถุ, สิ่งระบุวัตถุและแบบชนิดอ้างอิง (Object, object identity, and reference types)

ใน object-database systems, data objects สามารถให้ object identifier (oid) ได้ ซึ่งมีลักษณะคือเป็นค่าที่ไม่ซ้ำกัน โดย DBMS ทำหน้าที่ในการสร้าง oid และต้องรับผิดชอบว่า oid ของหนึ่ง object ต้องไม่มีค่าซ้ำกันกับของ object อื่นๆ ตลอดช่วง life time ของมัน ตัวอย่างเช่น ใน SQL3 tuple ใน relation สามารถกำหนด oid ได้โดยการเพิ่มวลี WITH IDENTITY ไปในการกำหนด table เช่น ในบรรทัดที่ 4 ของการกำหนด table ในหัวข้อที่ 1

oid ของหนึ่ง object สามารถถูกอ้างอิง (หรือชี้) ถึงมันได้จากที่อื่นในข้อมูล หนึ่งการอ้างอิงที่แน่นอนจะมีหนึ่ง type (คล้ายกันกับ type ของ pointer ใน programming language) ซึ่งสมนัยกับ type constructor :

ref(base) : a type representing a reference to an object of type base.

ตัวอย่างเช่นบรรทัดที่ 5 ของการกำหนด relation ของหัวข้อที่ 1 การกำหนด column *theater* of type ref(theater_t) โดย items ใน column นี้จะอ้างถึง objects ของ type theater_t, ดังเช่น rows ใน Theaters table, ซึ่งถูกกำหนดในบรรทัดที่ 4

ref(type constructor สามารถถูกซ้อนไว้ใน type constructors สำหรับ constructed types ตัวอย่างเช่น setof(ref(arrayof(integer))

4.1 แนวคิดของความเสมอภาค (Notions of equality)

ความแตกต่างระหว่าง reference types และ reference-free constructed types ทำให้เกิดแนวคิดอื่นๆ นั่นคือคำนิยามของความเสมอภาค สอง objects มี type เหมือนกันและจะถูกกำหนดให้เป็น deep equal ถ้าหากว่า :

- Objects เป็น atomic type (คือไม่มี subobject) และมี value เท่ากัน หรือ
- Objects เป็นของ reference type และ *deep equals* operator เป็น true สำหรับสอง referenced objects หรือ

- Objects เป็น constructed type และ *deep equals* operator เป็น true สำหรับส่วนประกอบย่อยที่ตรงกันทั้งหมดของสอง objects

สอง objects ซึ่งมี reference type เหมือนกันจะถูกกำหนดเป็น shallow equal ถ้าทั้งสองต่างอ้างถึง object เดียวกัน (นั่นคือการอ้างอิงใช้ oid เดียวกัน)

ตัวอย่างเช่น พิจารณาจาก complex objects row(538, **t89**, 6-3097, 8-7-97) และ row(538, **t33**, 6-3-97, 8-7-97) ซึ่ง type คือ type ของ rows ใน table Nowshowing สอง object นี้ไม่ได้เป็น shallow equal เพราะต่างกันตรง attribute value ที่สอง อย่างไรก็ตาม พวกนี้อาจเป็น deep equal ได้ ถ้าทั้งสอง objects อ้างถึง instance ของ objects (ซึ่งมี id **t89** และ **t33**) ของ type theater_t ซึ่งมี values เท่ากัน ตัวอย่างเช่น tuple(54, 'Majestic', '115 King', 2556698')

4.2 Differencing reference types

หนึ่ง item ของ reference type ref(foo) ไม่เหมือนกับหนึ่ง foo item ในตำแหน่งของมัน ในการเข้าถึง foo item ที่ถูกอ้างอิง หนึ่ง built-in deref() method จะถูกกำหนดให้แก่ ref type constructor ตัวอย่างเช่น ถ้าให้หนึ่ง tuple จาก Nowshowing table เราสามารถเข้าถึง *name* field ของ referenced theater_t object ด้วย syntax Nowshowing.deref(*theater*).name ซึ่งเป็นการเข้าถึง tuple ที่เป็น common type ซึ่งรวมกับ postfix versio ของ dereference operator และ tuple-type dot operator (ในบางระบบใช้การอ้างอิงด้วย C-style arrow operator) ถ้าใช้ arrow notation, *name* ของ reference theater สามารถเข้าถึงได้โดยใช้ syntax Nowshowing.*theater -> name*

5. การสืบทอด (Inheritance)

เป็นแนวคิดที่มีประโยชน์อย่างยิ่งในการออกแบบ classes ของ objects ที่มีลักษณะคล้ายกันแต่มีรายละเอียดบางส่วนต่างกัน ซึ่งใน object-database systems, inheritance สามารถถูกใช้ได้ใน 2 ลักษณะคือ สำหรับการนำกลับมาใช้ใหม่และการกลั่นกรองแบบชนิด และสำหรับการสร้างลำดับชั้นของกลุ่ม (collections) ของ objects ที่คล้ายกัน (แต่ไม่ใช่ object เดียวกัน)

5.1 การกำหนดชนิดด้วยการสืบทอด (Defining types with inheritance)

ใน Dinky database เราสร้าง movie theaters ด้วย type theater_t และ Dinky ต้องการ ฐานข้อมูล ของเขาแสดงผลของเทคนิคการตลาดแบบใหม่ในธุรกิจภาพยนตร์ : นั่นคือ *theater-cafe'* ซึ่งเสริฟ pizza และอาหารในขณะที่ชมภาพยนตร์ ซึ่ง Theater-cafes ต้องการข้อมูลเพิ่มเติมในการแสดงผลใน ฐานข้อมูล ในกรณีนี้ theater-cafe' ก็คล้ายกับ theater เพียงแต่มี attribute ที่แสดงถึง ราย

การอาหารเพิ่มขึ้นมา ซึ่ง Inheritance สามารถให้เรายึด "specialization" ที่ชัดเจนใน การออกแบบ ฐานข้อมูล ได้ด้วย DDL statement ต่อไปนี้

CREATE TYPE theatercafe_t **UNDER** theater_t(*menu* text);

statement นี้จะสร้าง new type คือ theatercafe_t ซึ่งมี attributes และ methods เช่นเดียวกับ theater_t เพียงแต่มี attribute *menu* ซึ่งมี type เป็น text ซึ่ง methods ที่ใช้บน theater_t สามารถใช้กับ type theater_t ได้ทั้งหมดแต่ไม่ใช่ในทางกลับกัน เราสามารถพูดได้ว่า theatercafe_t สืบทอด attribute และ methods ของ theater_t

ข้อสังเกตคือ กลไกของ inheritance ไม่ใช่เป็นหนึ่ง MACRO เช่น CREATE statements แต่มันจะสร้าง relationship ใน ฐานข้อมูล ระหว่าง subtype (theatercafe_t) และ supetype(theater_t) : *โดยหนึ่ง object ของ subtype ก็จะถูกพิจารณาเป็นหนึ่ง object ของ supertype ด้วย* การกระทำนี้หมายถึงว่า operations ใดก็ตามที่ใช้กับ supertype (method รวมทั้ง query operations เช่น projection หรือ join) ก็สามารถใช้กับ subtype ได้ ซึ่งเป็นหลักทั่วไปดังนี้ :

The Substitution Principle : Given a supertype A, and a subtype B, it is always possible to substitute an object of type B into a legal expression written for objects of type A, without producing type errors.

หลักการนี้ทำให้การนำ code กลับมาใช้สามารถทำได้ง่ายเพราะ queries และ methods ที่ถูกเขียนสำหรับ supertype สามารถถูกนำไปใช้กับ subtype ได้โดยไม่ต้องทำการแก้ไข

หมายเหตุ : inheritance สามารถนำไปใช้กับ atomic types รวมทั้ง row types ได้ในการให้ supertype image_t และ method *title(), number_of_color()* และ *display()* เราสามารถกำหนด subtype thumbanil_image_t สำหรับรูปเล็กๆ ที่ inherits methods จาก image_t

5.2 Binding of methods

ในการกำหนดหนึ่ง subtype บางครั้งเราอาจต้องการเปลี่ยน method ที่ได้จาก supertype ด้วย new version ที่ทำงานบน subtype ต่างออกไป พิจารณา image_t type และ subtype jpeg_image_t จาก Dinky database โชคไม่ค่อยดีที่ *display()* method สำหรับ standard images ไม่สามารถทำงานได้กับ JPEG images (เพราะถูก compressed เป็นแบบพิเศษ) ดังนั้นในการสร้าง type jpeg_image_t เราเขียน special *display()* method สำหรับ JPEG images และบันทึกมันลงใน database system โดยใช้ **CREATE FUNCTION** command :

CREATE FUNCTION *display*(jpeg_image) **RETURN** jpeg_image

AS EXTERNAL NAME '/a/b/c/jpeg.o';

การบันทึก new method ด้วยชื่อเดียวกับ old method เรียกว่า **overloading** ชื่อของ method เนื่องจาก overloading ระบบจะต้องเข้าใจว่า method ไหนที่ถูกพิจารณาใน expression ตัวอย่างเช่น เมื่อระบบต้องการเรียก *display()* จากหนึ่ง object ของ type jpeg_image_t มันจะใช้ specialized *display()* จากหนึ่ง object ของ type jpeg_image_t มันจะใช้ specializec *display* method และเมื่อมันต้องการเรียก *display* บน object ของ type image_t (ซึ่งไม่ใช่ subtype) มันจะเรียก standard *display* method ซึ่งกระบวนการในการตัดสินใจในการเรียก method เรียกว่า **binding** method ของ object

ในสภาวะที่แน่นอน binding สามารถกระทำได้ในขณะที่ expression ถูก parse **(early binding)** แต่ในกรณีของ most specific type ของ หนึ่ง object จะไม่สามารถจดจำได้จนกว่าจะถึง runtime ดังนั้น method ไม่สามารถทราบขอบเขตได้จนกว่าจะถึงเวลานี้ **(late binding)**

5.3 ลำดับชั้นของกลุ่ม, ขอบเขตของชนิดและคิวรี (Collection hierarchies, type extents and queries)

Database system มี query languages สำหรับจัดการกับ tabular datasets ซึ่งกลไกลของ programming languages จะถูกนำมาเพิ่มในส่วนของ object database เพื่อจัดการกับ table และ queries ได้ดีขึ้น ซึ่งในกรณีของ object-relational systems เราสามารถกำหนดหนึ่ง table ที่มี objects ชนิดต่างๆ ได้ ตัวอย่างเช่น Theaters table ใน Dinky schema ในการให้ new subtype เช่น theater_cafe' เราต้องการสร้างอีก table หนึ่งคือ Theater_cafes เพื่อเก็บข้อมูลที่เกี่ยวกับ theater cafes แต่เมื่อเราเขียน query บน Theater table (ซึ่งบางครั้งเป็น query ที่เหมือนกับของ Theater cafes table) เราอาจจะ project column ของ Theater_cafes tables ส่วนที่เหลือเพื่อแสดงผล query ของ Theater_cafes table ได้(เนื่องจากเราสามารถมอง instance ของ Theater_cafes table ว่าเป็น instance ของ Theaters table ได้)

แทนที่จะให้ผู้ใช้ระบุ query สำหรับแต่ละ table เราสามารถบอกระบบว่า new table ของ subtype ให้ถูกจัดเป็นส่วนหนึ่งของ table ที่เป็น supertype ได้ ความเกี่ยวเนื่องระหว่าง queries ของ table หลังจากตัวอย่างของเราเขียนได้เป็น :

CREATE TABLE theater_cafes **OF TYPE** theater_cafe'_t **UNDER** Theaters;

Statement นี้จะบอกให้ระบบว่า queries ที่กระทำกับ theaters table ควรที่จะถูก run over เป็น union

ของ theaters และ Theater_cafes tables ในกรณีนี้ ถ้าการกำหนด subtype มี method overloading, late-bined จะถูกใช้เพื่อประกันถึงการเรียก methods ที่เหมาะสมสำหรับแต่ละ tuple

โดยทั่วไป UNDER clause สามารถใช้เพื่อสร้างต้นไม้ตัดสินใจ (arbitrary tree) ของ tables ต่างๆ ได้ ซึ่งต้นไม้ที่จะได้ เรียกว่า collection hierarchy ซึ่ง Queries ของ table T ใน hierarchy คือ เป็น run over ของการ union T และตัวตาม (descendants) ของมันทั้งหมด syntax เพิ่มเติมสำหรับ ตัวอย่างคือ keyword ONLY สามารถใช้ใน FROM clause ของ query ได้

บางระบบมีการกำหนด special tables สำหรับแต่ละ type โดยอัตโนมัติ ซึ่งจะใช้เก็บ references ไปยัง instance ทั้งหมดของ type ที่มีอยู่ใน ฐานข้อมูล ซึ่ง tables เหล่านี้ถูกเรียกว่า type extents และยังให้ queries บน objects ทั้งหมดสำหรับหนึ่ง type ที่ได้รับ ไม่ว่า objects จะอยู่ส่วนไหนใน ฐานข้อมูล ก็ตาม โดย type extents จะสร้างหนึ่ง collection hierarchy สำหรับ type hierarchy ที่เทียบเท่ากัน

6. การออกแบบฐานข้อมูลสำหรับ ORDBMS (Database design for an ORDBMS)

Data types ต่างๆ ที่มีอยู่ให้ ORDBMS ให้โอกาสแก่ผู้ออกแบบฐานข้อมูลสามารถออกแบบฐานข้อมูลได้อย่างเป็นธรรมชาติและมีประสิทธิภาพมากขึ้น ในหัวข้อนี้เราจะกล่าวถึงความแตกต่างระหว่าง RDBMS และ ORDBMS database design โดยตัวอย่างต่างๆ

6.1 ตัวอย่างชนิดและข้อมูลเชิงนามธรรม (Constructed types and ADTs)

ตัวอย่างแรกของเราประกอบด้วยหลาย space probes ซึ่งแต่ละอันบันทึกหนึ่ง video อย่างต่อเนื่องโดยหนึ่ง single video stream จะถูกรวมเข้ากับแต่ละ probe(และในขณะที่ stream นี้ถูกรวมผ่านไปช่วงเวลาหนึ่ง) เราสมมุติว่าตอนนี้ video คือหนึ่ง object ที่สมบูรณ์ซึ่งถูกรวมกับ probe ไป ซึ่ง video ถูกบันทึกในขณะที่ช่วงเวลาผ่าน (เพื่อข้อมูลต่างๆ สามารถไปเป็นส่วนประกอบตอนต้นของหนึ่ง video stream ซึ่งเป็นมาตรฐาน MPEG ได้ง่าย) ดังนั้น information ที่มีอยู่ในแต่ละ probe จะมีสามส่วนคือ (1) a probe id ซึ่งทำหน้าที่ระบุหนึ่ง probe โดยไม่ซ้ำกัน (2) video steam (3) หนึ่ง location sequence ของคู่อันดับของ (time, location) เราควรจะเก็บข้อมูลใช้ database schema ที่มีลักษณะอย่างไร?

An RDBMS database design

ใน **RDBMS** เราจะต้องเก็บแต่ละ video steam เป็น blob และแต่ละลำดับตำแหน่งจะเป็น tuple ในหนึ่ง table สำหรับ **RDBMS** database design ถูกแสดงไว้ดังข้างล่างนี้ :

Probes(*pid*: **integer**, *time* : **timestamp**, *lat* : **real**, *long* : **real**, *camera* : **string**, *viedeo* **:blob)**

มี single table ซึ่งเรียกว่า Probes, และมันมี rows ต่างๆ สำหรับแต่ละ probe ซึ่งแต่ละหนึ่งของ rows เหล่านี้มี *pid.camera* และ *video* values เท่ากันแต่ต่างกันที่ *time, lat* และ *long* values (เราใช้ latitude และ longitude เพื่อแสดงถึง location) key สำหรับ table นี้เราสามารถแสดงได้โดย functional dependency : **PTLN → CV** (N เป็นตัวย่อสำหรับ longitude) และอีกหนึ่ง dependency คือ **P → CV** เนื่องจาก relation นี้ไม่ใช่ BCNF และ 3NF เราจึง decompose Probes เพื่อให้ได้ BCNF schema

Probes_Loc(*pid* **: integer**, *time* **: timestamp,** *lat* **: real,** *long* **: real)**

Probes_Video(*pid* **: integer,** *camera* **: integer,** *camera* **: string,** *video* **: blob)**

นี่คือการออกแบบที่เกือบดีที่สุดซึ่งสามารถทำได้ใน **RDBMS** อย่างไรก็ตามยังมีข้อจำหลายอย่างคือ

ข้อแรกในการแสดง videos เป็น blobs หมายถึงเราจะต้องเขียน application code ใน external language เพื่อจัดการกับ video object ใน ฐานข้อมูล พิจารณาได้จาก quest นี้ "For probe la, display the video recorded between 1:10 p.m. and 1:15 p.m. on May 10 1996" เราจะต้องค้นคืน video object ซึ่งมีใน probe 10 ทั้งหมด (ซึ่งใช้เวลาในการบันทึกหลายชั่วโมง) เพื่อจะแสดงเพียงส่วนบันทึกหนึ่งเล็กๆ ที่มีเวลาไม่เกิน 5 นาที

ข้อถัดไปคือแต่ละ probe จะมีลำดับของการอ่านที่ไม่ชัดเจน และลำดับของ information ที่มีใน 1 probe จะอยู่กระจัดกระจายไปตาม tuples ต่างๆ ข้อจำกัดที่สามคือเราต้องใช้ความพยายามมากในการแบ่ง video information จากลำดับของ information เพื่อหนึ่ง probe ข้อจำกัดเหล่านี้ถูกแสดงโดย queries ที่ต้องการให้เราพิจารณาถึง information ทั้งหมดที่มีอยู่ในแต่ละ probe ยกตัวอย่างเช่น "For each probe, print the earliest time at which it recorded, and the camera type" ซึ่ง query นี้ต้องมีการ join กันระหว่าง Probes_Loc และ Proes_Video บน *pid* field

An ORDBMS database design

ORDBMS ให้การสนับสนุนคำตอบที่ดีกว่า สิ่งแรก เราสามารถเก็บ video เป็นหนึ่ง **ADT** object ได้ และเขียน methods ซึ่งควบคุมการปรับปรุงพิเศษต่างๆ ที่เราต้องการกระทำอย่างที่สองเราเก็บ contructed types เช่น lists ได้ เราจึงสามารถจัดเก็บลำดับตำแหน่งสำหรับหนึ่ง probe ในหนึ่ง sigle tuple ได้ รวมทั้ง video information ด้วย การออกแบบนี้ช่วยลดความต้องการของการ join ใน queries ที่ต้องการทั้งลำดับและข้อมูล video ซึ่ง **ORDBMS** design สำหรับตัวอย่างของเรามีเพียงหนึ่ง single relation คือ Probes_AllInfo :

Probes_AllInfo(*pid* : **integer**, *locseq* : **location_seq**, *camera* : **string**, *video* : **mpeg_stream**)

การกำหนดนี้มีสอง new types คือ location_seq และ mpeg_stream ซึ่ง mpeg_stream type ถูกกำหนดเป็น **ADT** กับ method *display()* ซึ่งใช้สำหรับฉายส่วนของ video ที่ถูกบันทึกในช่วงของ parameters ที่รับ (start time และ end time) method นี้สามารถสร้างได้อย่างมีประสิทธิภาพโดยการมองหาเวลาที่ใช้บันทึกทั้งหมดและความยาวของ video และแทรกเข้าไปเพื่อนำส่วนที่ถูกบันทึกของช่วงเวลาที่ระบุใน query

ดิวรีตัวอย่างแรกที่เราจะแสดงใน extended SQL syntax คือการใช้ *display* method ในตอนนี้เราจะค้นหาเพียงส่วนหนึ่งที่ต้องการใน video มากกว่าที่จะเป็นทั้งหมดของ video

SELECT display(P.video, 1:10 p.m. May 10 1996, 1:15 p.m. May 10 1996)

FROM Probes_AllInfo P

WHERE P.pid = 10

พิจารณา location _seq type ในขณะนี้ เราสามารถกำหนดมันเป็น list type ซึ่งประกอบด้วยหนึ่ง list ของ row type objects ได้ :

CREATE TYPE location seq listof

(row (*time* : timestamp, *lat* :real, *long* : real))

พิจารณา locseq field ในหนึ่ง พนไ สำหรับ probe ที่กำหนดให้ซึ่ง field นี้ประกอบด้วยหนึ่ง list ของ rows แต่ละส่วนมีสาม fields ถ้า ORDBMS สร้าง collection types ในรูปแบบเต็ม เราสามารถดึง time column จาก list นี้ได้เพื่อสร้างหนึ่งเซตของ timestamp values และใช้ MIN aggregate operator ไปยัง list นี้เพื่อค้นหาเวลาเริ่มต้นของ probe นี้ว่าบันทึกเมื่อไร การสนับสนุนของ collection types ทำให้เราสามารถแสดง query ที่สองได้ดังนี้

SELECT P.pid, MIN(P.locseq.time)

FROM Probes_AllInfo P

ตัวอย่างต่อไปนี้คือเราอาจต้องการทำ specialized operations บนลำดับตำแหน่งของเรามากกว่าที่ทำโดย standard aggregate operators ตัวอย่างเช่น เราอาจต้องการกำหนดหนึ่ง method ที่นำหนึ่งช่วงเวลาแล้วมาคำนวณระยะทางที่ผ่านโดยในช่วงเวลาดังกล่าว ซึ่ง code สำหรับ method นี้จะต้องเข้าใจถึงรายละเอียดการเดินทางของ probe และระบบกระจายพิกัด ด้วยเหตุผลเหล่านี้เราจึงเลือกที่จะกำหนด location_seq มี type เป็น ADT

6.2 สิ่งระบุวัตถุ (Object identity)

ในหัวข้อนี้จะกล่าวถึงความสำคัญในการใช้ reference types หรือ oids ซึ่งจะมีประโยชน์เมื่อขนาดของวัตถุมีขนาดใหญ่ซึ่งเนื่องมาจากวัตถุเป็น constructed data type หรือ big object เช่น image

ถึงแม้ว่า reference types และ constructed types ต่างดูคล้ายกัน แต่จริงๆ แล้วมันต่างกัน ตัวอย่างเช่น พิจารณา constructed type my_theater tuple(*tno* integer, *name* text, *address* text, *phone* text) และ reference type theater(ref(theater_t)) ของการประกาศ schema ในหัวข้อที่ 1.1 มีความแตกต่างที่สำคัญซึ่งส่งผลต่อในการทำ database updates ต่อสอง types นี้คือ

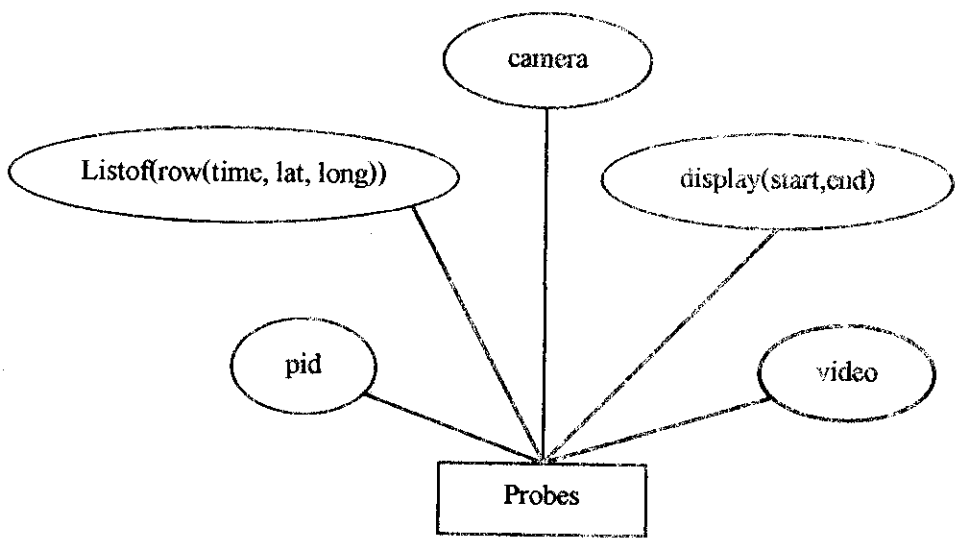
- **Deletion :** Objects ซึ่งมีการอ้างอิง อาจได้รับผลกระทบจากการลบ objects ที่พวกมันอ้างอิงอยู่ในขณะที่ reference-free constructed objects ไม่ถูกผลกระทบจากการลบ objects อื่นๆ ตัวอย่างเช่น ถ้า Theaters table ถูก drop จาก ฐานข้อมูล หนึ่ง object ของ type theater อาจถูกเปลี่ยน value เป็น *null* เพราะว่า theater_t object ที่มันอ้างถึงได้ถูกลบไปแล้ว ในขณะที่ similar object ของ type my theater จะไม่ถูกเปลี่ยนมูลค่า
- **Update :** Objects ของ reference types จะเปลี่ยนมูลค่าถ้า referenced object ถูก update ในขณะที่ objects ของ reference-type constructed types จะเปลี่ยน value ต่อเมื่อมันถูก update โดยตรง
- **Sharing versus Copying :** หนึ่ง identified object สามารถถูก referenced โดย multiple reference-type items ดังนั้นในการ update แต่ละครั้งของ objects จะส่งผลในหลายตำแหน่ง ถ้าต้องการให้ได้ผลเช่นนี้ใน reference-free types จะต้อง update ชุดสำเนาของ object นั้นๆ ทั้งหมด

Object identity versus foreign keys

การใช้หนึ่ง oid เพื่ออ้างไปยังหนึ่ง object คล้ายกันกับการใช้หนึ่ง foreign key เพื่ออ้างไปยัง tuple ใน repeation อื่นๆ แต่ไม่เหมือนกันทีเดียวนักเพราะการใช้ oid สามารถชี้ไปยังหนึ่ง object ของ theater_t ซึ่งถูกเก็บไว้ที่ใดสักแห่งใน ฐานข้อมูล ในขณะที่การอ้างอิงของหนึ่ง foreign มีเงื่อนไขคือต้องชี้ไปยังหนึ่ง object ใน relation ที่ถูกระบุ

6.3 ส่วนขยายของตัวแบบ ER (Extending the ER model)

ส่วนเพิ่มเติม ER Model เพื่อใช้สำหรับการออกแบบฐานข้อมูลใน ORDBMS จะต้องมีส่วนสนับสนุน constructed attributes (เช่น set, list, arrays เป็น attribute values) และต้องให้เราสามารถออกแบบ entities ซึ่ง attribute มี methods รวมอยู่ได้ ตัวอย่างเช่น การใช้ Extend ER diagram เพื่ออธิบาย space probe data (สัญลักษณ์ที่ใช้ในที่นี้เพื่อแสดงตัวอย่างเท่านั้น)



Space Probe Entity Set

ตัวอย่างการกำหนดของ Probes ในรูปบนแสดงถึงสิ่งใหม่ 2 ลักษณะ อย่างแรกคือมี constructed-type attribute listof(row(time, lat, long)) แต่ละมูลค่าที่ถูกกำหนดยัง attribute นี้คือหนึ่ง list ของ tuples ซึ่งมี 3 fields อย่างที่สอง Probes มีหนึ่ง attribute ซึ่งเรียกว่า video ซึ่งก็คือ abstract data type object ซึ่งเป็นเส้นทึบต่อไปยัง Probes อีกทั้ง attribute นี้มีหนึ่ง attribute เป็นของมันเองซึ่งก็คือ method ของ ADT

ส่วนเพิ่มเติมที่สำคัญของ E-R model คือมันต้องสนับสนุนการออกแบบของ nested collections ตัวอย่างเช่น ถ้าหนึ่งลำดับตำแหน่งถูกออกแบบเป็น entity และเราต้องการกำหนดหนึ่ง attribute สำหรับ Probes ซึ่งเก็บหนึ่งเซตของ entities ต่างๆ ซึ่งเนื้อหาจะอยู่ในหัวข้อถึงไป

6.4 การใช้ nested collections (Using nested collections)

Nested collections ให้การออแบบที่ดีขึ้น แต่ก็ทำให้เกิดการตัดสินใจที่ยากในการออกแบบ ตัวอย่างเช่น พิจารณาแนวทางในการออกแบบ location sequences ต่อไปนี้ (ข้อมูลอื่นๆ ที่เกี่ยวกับ probes ถูกละไว้เพื่อความชัดเจนในการอธิบาย)

Probes1(*pid* : integer, *locseq* : location_seq)

นี้คือทางเลือกที่ดีถ้า queries ที่สำคัญในงานส่วนใหญ่ต้องการให้เรามองหาลำดับตำแหน่งสำหรับหนึ่ง probe ตัวอย่างเช่น "For each probe, print the earliest time at which it recorded, and the camera type." ในอีกทางหนึ่งพิจารณา query ที่ต้องการให้เราค้นหาลำดับตำแหน่งทั้งหมด "Find the earliest time at which a recording exists for lat = 5, long = 90" ซึ่ง query นี้สามารถตอบได้อย่างมีประสิทธิภาพถ้าใช้ schema ต่อไปนี้

Probes2(*pid* : integer, *time* : timestamp, *lat* : real, *long* : real)

ดังนั้น การเลือก schema จะต้องพิจารณางาน expected workload เสมอ ซึ่งตัวอย่างนี้แสดงถึงความเหมาะสมของ nested collections ที่ใช้กับกรณีต่างๆ แต่บางครั้งสมบัตินี้อาจนำไปใช้อย่างผิดพลาดได้ง่าย ดังนั้นการใช้ nested collections ควรทำอย่างระวัง

7. แนวทางในการสร้าง ORDBMS (New challenges in implementingt an ORDBMS)

ในหัวข้อนี้เราจะศึกษาถึงกุญแจหลักซึ่งจะทำให้การสร้าง ORDBMS เป็นไปอย่างมีประสิทธิภาพ นั่นคือ ORDBMS เป็นแบบ fully functional

7.1 หน่วยจัดเก็บและวิธีการเข้าถึง (Storage and access methods)

เนื่องจาก object-relational databases มีการเก็บ new types ของ data ผู้สร้าง ORDBMS จะต้องดูกลับไปยังบางส่วนของหน่วยจัดเก็บและการจัดดรรชนีเพื่อให้ระบบสามารถเก็บ ADT objects และ constructed objects รวมทั้งการเข้าถึงได้อย่างมีประสิทธิภาพ

Storing large ADT and constructed type objects

ADT objects ขนาดใหญ่และ constructed objects ทำให้เกิดความสับสนของรูปแบบข้อมูลบน disk ซึ่งปัญหานี้เป็นที่เข้าใจดีและได้ถูกแก้ไขแล้วใน ORDBMSs และ OODBMSs เราจะแสดงถึงปัญหาหลักที่นี่

User-defined ADTs สามารถอาจขนาดค่อนข้างมากซึ่งมักจะเกินกว่าหนึ่ง single disk page และ ADTs ขนาดใหญ่เช่น blobs มีความต้องการหน่วยจัดเก็บพิเศษ ซึ่งโดยปกติคือตำแหน่งต่างๆ บน disk จาก tuples ซึ่งบรรจุพวกมัน ในกรณีนี้ Disk-based pointers จะถูกรักษาจาก tuples ที่ objects นั้นๆ บรรจุอยู่

Constructed objects อาจมีขนาดมากได้เช่นเดียวกัน แต่ต่างจาก ADT objects ตรงที่ขนาดของ objects ชนิดนี้มักจะมีการเปลี่ยนแปลงได้ในช่วง lifetime ของ ฐานข้อมูล ตัวอย่างเช่น *stars* attribute ของ *films* table เมื่อเวลาผ่านไป ผู้แสดงประกอบบางคนในภาพยนตร์เก่าๆ อาจมีชื่อเสียงขึ้นมาได้ Dinky อาจต้องการโฆษณาการแสดงของพวกเขาในฟิล์มก่อนๆ สิ่งนี้รวมถึงการใส่ *starts* attribute ของแต่ละ tuple ใน *films* เพราะ bulk attributes เหล่านี้สามารถเพิ่มได้ตามอิสระ จึงต้องมีกลไกลของ flexible disk layout

อีกปัญหาหนึ่งคือปัญหาที่เกิดกับ array type ซึ่งมีการจัดเก็บแบบ row-by-row (คือมี arrays หลาย arrays) การเข้าถึง subarrays ที่ไม่ได้ถูกเก็บอย่างต่อเนื่องบน disk จะทำให้เกิด I/O cost สูง การแก้ปัญหาทำได้โดยการแตก arrays เป็น *chunks* ที่ต่อเนื่องกันแล้วเก็บอย่างเป็นลำดับบน disk ซึ่งไม่จำเป็นต้องเก็บแบบ row-by-row หรือ column-by-column

การทำดรรชนีแบบชนิดใหม่ (Indexing new types)

ใน RDBMS โครงสร้างของดรรชนีจะสนับสนุนเพียง equality conditions (B+trees และ hash indexes) และ range conditions (B+ tree) ซึ่งปัญหาที่สำคัญของ ORDBMSs คือการให้ดรรชนีที่มีประสิทธิภาพสำหรับ ADT methods และ operators บน constructed objects

มีเทคนิคการทำดรรชนีมากมายที่ผู้ผลิต ORDBMSs ต่างคิดค้นขึ้น แต่ถึงอย่างไรก็ตามคงไม่มีบริษัทใดที่นำเทคนิคทั้งหมดที่มีอยู่มาใส่ลงในผลิตภัณฑ์ของตนเองได้ ดังนั้นแนวคิดหนึ่งสำหรับการสร้างดรรชนีคือให้ผู้ใช้สามารถกำหนดได้เอง ตัวอย่างเช่น ผู้เชี่ยวชาญในการทำแผนที่ไม่เพียงแค่นำ ADT (ซึ่งก็คือคู่อันดับของ (latitude, longitude) ไปใช้บนแผนที่เท่านั้น แต่ยังสามารถกำหนด index structure ที่สนับสนุน map queries ได้

แนวทางหนึ่งในการให้ผู้ใช้สามารถกำหนด index structure เองได้ คือจัดพิมพ์ *access method interface* ซึ่งให้ผู้ใช้สร้าง index structure ภายนอก DBMS ซึ่ง index และ data ถูกเก็บไว้ในระบบ file system และ DBMS จะส่ง *open, next* และ *close* iterator request ไปยัง external index code ของ user เพื่อทำการติดต่อ อย่างไรก็ตามวิธีนี้ยังคงมีข้อเสียคือ data ใน external index จะไม่ถูกปกป้องด้วย concurrency และ recover functions ของ **DBMS** สำหรับทางเลือกอีกแนวหนึ่งที่ถูกคิดขึ้นเพื่อแก้ปัญหานี้คือ สร้าง "template" index structures ซึ่งมี index structures เพียงพอกับที่ผู้ใช้อาจสร้างขึ้นได้ ดังนั้นโครงสร้างของตรรชนีจึงถูกสร้างขึ้นภายใน DBMS มันจึงสนับสนุน concurrency และ recovery ในระดับที่สูง ตัวอย่างเช่น *GeneralizedSearchTree(GiST)* คือตัวอย่างของ template index structure ซึ่งมีพื้นฐานบน B+ trees ซึ่งมี tree index structures ซึ่งถูกคิดค้นเพื่อสร้างมากมากเพียงแต่ให้ user-defined ADT code เพียงไม่กี่บรรทัดเท่านั้น

7.2 การประมวลผลคำถาม (Query processing)

ADTs และ constructured types เรียกการทำงานใหม่ๆ ในการประมวลผล queries ใน ORDBMSs สิ่งเหล่านี้ยังเปลี่ยนแปลงข้อมูลสันนิษฐานบางประการที่มีผลกระทบต่อประสิทธิภาพของ queries ด้วย โดยในหัวข้อนี้เราจะพิจารณาถึงแนวทางของสอง functionality issues (user-defined aggregates และ security) และสอง efficiency issues(method caching and pointer swizzling)

User-defined aggregation functions

ORDBMSs ส่วนมากจะให้ผู้ใช้สามารถบันทึก aggregation functions กับระบบได้ซึ่งผู้ใช้จะต้องสร้าง 3 methods ซึ่งเราเรียนกว่า *initialize, iteratie* และ *terminate* สำหรับ initialize method จะตั้งต้น internal state สำหรับ aggregation ส่วน *iterate* method จะปรับปรุงทุกๆ tuple ที่พบและ *terminate* method จะคำนวณ aggregation ซึ่งผลลัพธ์ขึ้นอยู่กับ final state จากนั้นจะ clean up ยกตัวอย่างเช่น aggregation function ที่ใช้ในการคำนวณ second-highest value ในหนึ่ง field *initialize* call จะจองพื้นที่ของ storage สำหรับมูลค่าสูงสุดสองมูลค่า iterate call จะเปรียบเทียบ current tuples's value กับ top two แล้ว update top two ตามความจำเป็น และ *terminate* call จะลบ storage สำหรับ top two values แล้วส่งคืนค่า copy ของ second-highest value

Method security

ADTs มีพลังให้ผู้ใช้เพิ่ม code ไปยัง DBMS ได้ ซึ่งอาจเป็นผลร้ายได้ถ้า ADT นั้นมี bug หรือ code ที่เป็นอันตรายซึ่งอาจทำให้ database server เกิดความเสียหายได้ ซึ่ง DBMS จะต้องมีกลไกในการทำหน้าที่ป้องกัน ADTs ที่ทำให้เกิดปัญหาได้

แนวทางหนึ่งในการป้องกันปัญหานี้คือให้ user method ถูก interpret มากกว่าที่จะถูก compile เพื่อที่จะตรวจสอบในแต่ละขั้นว่า method นั้นปลอดภัยก่อนที่จะ execute มัน ส่วนอีกแนวทางหนึ่งคือให้ user method ถูก compile จาก general purpose programming language อย่างเช่น C++ ได้แต่ต้อง run ภายนอกที่อยู่ของ DBMS ในกรณีนี้ DBMS จะส่ง explicit interprocess communications (IPCs) ไปยัง user method ซึ่งจะส่ง IPCs กลับมา

Method Caching

User-defined ADT methods สิ้นเปลืองมากในการประมวลผล และในความเป็นจริงแล้วเวลาส่วนใหญ่ที่เสียไปในการประมวลผล query ก็คือ bulk จึงเป็นไปได้ว่าในระหว่างที่ query ถูกประมวลผลอาจมีการ cache ผลลัพธ์ของ method ในกรณีที่มีการเรียกหลายครั้งและมี arguments เหมือนกัน สำหรับภายใน scope ของหนึ่ง single query เราสามารถเลี่ยงการเรียก method ซ้ำบน values ของหนึ่ง column ได้โดยการเรียงลำดับของ column นั้นหรืออีกแนวทางหนึ่งคือการสร้าง cache ของ method inputs และ matching outputs กับหนึ่ง table ใน ฐานข้อมูล จากนั้นหามูลค่าของหนึ่ง method ได้จาก inputs ที่ให้ แล้วจะสามารถ join input tuples กับ cache table ได้ โดยแนวทางทั้งสองนี้สามารถใช้ร่วมกันได้

Pointer Swizzing

ในบาง applications, objects จะถูกเรียกเข้าสู่ memory และเข้าถึงบ่อยๆ โดยผ่านทาง oids การ dereference จะต้องถูกสร้างอย่างมีประสิทธิภาพ บางระบบดูแลหนึ่ง table ของ oids จาก objects ที่(ปัจจุบัน)อยู่ใน memory เมื่อหนึ่ง object *O* ถูกนำเข้าสู่ memory applications จะ check แต่ละ oid ที่มีอยู่ใน *O* และแทนที่ oids ของ in-memory objects ด้วย in-memory pointers ไปยัง objects เหล่านั้น เทคนิคนี้ถูกเรียกว่า pointer swizziing และทำการอ้างอิงไปยัง in-,memory objects อย่างรวดเร็ว เมื่อหนึ่ง object ถูก page out, in-memory references ไปยังมันจะต้องไม่มีผลบังคับใช้ และจะต้องถูกแทนที่ด้วย oid ของมัน

7.3 Query optimization

ในการที่จะจัดการกับ new query processing functionality ได้นั้น optimizier จะต้องรู้เกี่ยวกับ new functionality และใช้มันได้อย่างถูกต้อง ในหัวข้อนี้เราจะกล่าวถึงการแสดงข้อมูลของ optimizer ใน 2 ลักษณะคือ (new indexes และ ADT method estimation) และอีกส่วนหนึ่งคือส่วนที่ถูกละเลยใน relational systems (expensive selection optimization)

Registering indexes with the optimizer

เมื่อ new index structures ถูกเพิ่มไปยัง system – ไม่ว่าจะโดยทาง external interfaces หรือ built-in template structure เช่น GiSTs -- optimizer ต้องได้รับแจ้งจากสิ่งที่มีอยู่จริงและมูลค่าของการ access ซึ่งถ้าให้หนึ่ง index structure ตัว optimizer จะต้องทราบคือ (a) WHERE-clause conditions ซึ่งตรงกับ index นั้นๆ และ (b) มูลค่าของการ fetch หนึ่ง tuple สำหรับ index นั้นๆ ด้วยข้อมูลเหล่านี้ optimizer จะสามารถใช้ index structure ใดๆ ก็ได้เพื่อสร้างแบบแผนของ query ซึ่ง ORDBMSs ต่างๆ จะมี syntax ที่ต่างกันสำหรับการบันทึก new index structures โดยระบบส่วนใหญ่ต้องการให้ผู้ใช้แสดงจำนวนของ cost of access (เพื่อเป็นแนวทางสำหรับ DBMS ในการวัดโครงสร้างที่จะถูกใช้) และจะบันทึกสถิตินั้นๆ ไว้

Reduction factor and cost estimation for ADT methods

การประเมิน reduction factors สำหรับ user-defined conditions เป็นปัญหาหนึ่งที่ยากและกำลังมีการศึกษากันอยู่ แนวทางที่ใช้กันในขณะนี้คือให้ user เป็นผู้จัดการโดย user ผู้ที่บันทึก method สามารถบันทึก auxiliary function ในการประเมินถึง reduction factor ของ method ถ้าหากไม่มีการบันทึก function นี้ optimizer จะสามารถใช้มูลค่าที่อยู่อย่างกระจัดกระจายได้เพียง 1/10 เท่านั้น

ADT methods อาจค่อนข้างสิ้นเปลืองและเป็นสิ่งสำคัญที่ optimizer จะต้องรู้ถึง cost ในการ execute ของ methods เหล่านี้ ซึ่งเป็นปัญหาเปิดอยู่ในขณะนี้ ในระบบที่มีอยู่ในปัจจุบัน users ผู้บันทึก methods สามารถระบุถึง cost ของ method เป็นจำนวนที่อยู่ในหน่วย units cost ของหนึ่ง I/O ในระบบ ซึ่งการประเมินนี้เป็นส่วนที่ยากสำหรับ users ในการจะทำให้ถูกต้องแนวทางที่น่าสนใจหนึ่งสำหรับ ORDBMS คือการ run method บน objects ที่มีขนาดต่างๆ กันและพยายามให้การประเมิน cost ของ methods เป็นไปอย่างอัตโนมัติ แต่ว่าแนวทางนี้ยังไม่มีการสำรวจในรายละเอียดและในปัจจุบันยังไม่มีการนำไปสร้างใน commercial ORDBMSs ใดๆ

Expensive selection optimization

ใน relational systems การเลือกจะถูกประเมินเป็น zero-time operation ตัวอย่างเช่น จะไม่มีความต้องการ I/O และ CPU cycles จำนวนมากในการทดสอบ *emp.salary < 10* อย่างไรก็ตาม เงื่อนไขเช่น *is_herbert(Frames.image)* ก่อนข้างที่จะสิ้นเปลืองเพราะมีการ fetch objects จาก disk และ process objects เหล่านี้ใน main memory ในลักษณะที่ค่อนข้างซับซ้อน

ORDBMS optimizers ต้องพิจารณาให้รอบคอบว่าจะทำอย่างไรในการเลือกเงื่อนไข ตัวอย่างเช่น selection query ที่ทำการทดสอบ tuples ใน Frames table ด้วยสองเงื่อนไข คือ *Frames.frameo < 100 ^ is_herbert(Frame.image)* มันเป็นแนวทางดีกว่าถ้าทำการทดสอบ *frameno* condition ก่อนทำการทดสอบ *is_herbert* เพราะว่าเงื่อนไขแรกทำได้เร็วกว่าและบ่อยครั้งที่อาจส่งคืนค่าเป็น false ซึ่งจะช่วยลดปัญหาของการทดสอบในเงื่อนไขที่สองได้ โดยทั่วไปการจัดลำดับที่ดีที่สุดในการเลือกคือใช้ function ของ costs และ reduction factors ของมัน ซึ่งการเลือกควรจะถูกจัดลำดับโดยการเพิ่มขึ้นของ *rank* ซึ่ง rank = (reduction factor – 1)/cost ถ้า selection มี rank สูงมากและเกิดขึ้นใน multitable query มันอาจทำการเลื่อนเวลาในการเลือกออกไปจนกระทั่งหลังการ joins เกิดขึ้น ซึ่งรายละเอียดการวาง expensive selection อย่างเหมาะสมระหว่างการ join บางครั้งค่อนข้างยุ่งยากและเพิ่มความซับซ้อนให้กับกระบวนการ optimization ใน ORDBMS

Summary

Comparing RDBMS with OODBMS and ORDBMS

ขณะนี้เราได้ศึกษาถึงหลักของ object-oriented DBMS extensions และถึงเวลาที่ต้องพิจารณาส่วนแตกต่างของที่สำคัญระหว่าง object-databases (ซึ่งก็คือ OODBMSs และ ORDBMSs) เมื่อเทียบกับ RDBMSs แม้ว่าเราได้แสดงถึงหลักของ object-databases หลายอย่าง แต่ก็ยังต้องกำหนดความหมายของคำว่า OODBMS และ ORDBMS อีก

ORDBMS คือ relational DBMS ซึ่งมีส่วนขยายต่างๆ ดังที่กล่าวมาแล้ว (ORDBMS systems ทั้งหมดที่มีอยู่ในปัจจุบันอาจจะยังไม่สนับสนุนส่วนขยายในรูปแบบทั้งหมดดังที่กล่าวมา แต่ว่าข้อพิจารณาในหัวข้อนี้ของเราคือหลักของมันมากกว่าที่จะเป็นระบบใดระบบหนึ่ง) ส่วน OODBMS คือ programming language ซึ่งมี type system ที่สนับสนุนลักษณะสำคัญต่างๆ ที่กล่าวถึงและให้ data object เป็น persistent นั่นคือมันสามารถคงอยู่พ้นช่วงนอกเหนือจากที่โปรแกรมต่างๆ ทำงาน โดยระบบต่างๆ ในปัจจุบันโดยมากยังไม่ปรับเข้ากับหลักการที่มีอยู่ได้อย่างครบถ้วน แต่จะพยายามทำให้ใกล้เคียงกับผู้อื่นมากที่สุดและสามารถแยกประเภทได้โดยตรง

- **OODBMS : ODL and OQL**

ถึงแม้ว่าเราได้กำหนดให้ OODBMS คือ programming language ที่มีส่วนสนับสนุนสำหรับ persistent objects และในความจริงที่ว่า OODBMSs ให้การสนับสนุน collection types จึงทำให้มันสามารถกำหนด query language สำหรับ collections ต่างๆ ได้ซึ่งมาตรฐานนี้ได้ถูกพัฒนาโดย Object Database Management Group (ODMG) และถูกเรียกว่า Object Query Language หรือ OQL ซึ่ง OQL คล้ายกับ SQL ในส่วนของ SELECT-FROM-WHERE-style syntax (รวมทั้งสนับสนุน GROUP BY HAVING และ ORDER BY ด้วย) และอื่นๆ ซึ่งถูกเสนอโดยส่วนขยายของมาตรฐาน SQL3 แต่ต่างกันตรงที่ OQL สนับสนุน constructed types รวมทั้ง sets, bages, arrays และ lists การจัดการกับ collections ของ OQL มีรูปแบบที่เป็น มาตรฐานเดียวมากเพราะมันไม่ได้แบ่งแยกการจัดการต่างหากกับ collection ของ rows ตัวอย่างเช่น OQL สามารถใช้ aggregate operation COUNT กับ list ได้เพื่อคำนวณความยาวของ list, OQL ยังให้การสนับสนุน reference type, path expressions, ADTs และ inheritance, type extents และ SQL-style nested queries รวมทั้งยังมีมาตรฐานของ data definition language สำหรับ OODBMSs (Object Data Language or ODL)

ซึ่งคล้ายกับ DDL subset ของ SQL แต่ให้การสนับสนุนลักษณะเพิ่มเติมที่พบใน OODBMSs เช่น การกำหนด ADT

- **RDBMS versus ORDBMS**

ข้อแตกต่างระหว่าง RDBMS กับ ORDBMS เห็นได้โดยตรงคือ RDBMS ไม่ได้ให้การสนับสนุนส่วนขยาย ความเรียบง่ายของ Relational Data Model ทำให้การ optimize queries เพื่อประสิทธิภาพในการ execute ทำได้ง่ายกว่า ตัวอย่างเช่น relation system จะใช้งานได้ง่ายกว่าเพราะมี features ต่างๆ น้อยกว่าในขณะที่มันจะมีความเป็นเอนกประสงค์น้อยกว่า ORDBMS

- **OODBMS versus ORDBMS : Similarities**

OODBMSs และ ORDBMSs ทั้งคู่ต่างให้การสนับสนุน user-defined ADTs, constructed types, object identity, reference types และ inheritance และทั้งสองระบบสนับสนุน query language สำหรับจัดการกับ collection types ORDBMSs สนับสนุนส่วนขยายในรูปแบบของ SQL และ ORDBMSs สนับสนุน ODL/OQL ส่วนที่คล้ายกันนี้ไม่ใช่สิ่งบังเอิญ เพราะว่า ORDBMS มีเจตนาต้องการในการเพิ่ม OODBMS features ไปยัง RDBMS และในทางกลับกัน OODBMSs มีความต้องการพัฒนา query languages ซึ่งมีพื้นฐานมาจาก relational query languages ทั้ง OODBMSs และ ORDBMSs ให้ DBMS มีการทำงานเช่น concurrency control และ recovery

- **OODBMS versus ORDBMS : Differences**

ข้อแตกต่างคือหลักการที่ใช้ในการปฏิบัติ OODBMSs พยายามที่จะเพิ่ม DBMS functionality ไปยัง programming language ในขณะที่ ORDBMSs พยายามที่จะเพิ่ม data types ที่มากขึ้นกับ relational DBMS ถึงแม้ว่าทั้งสองแนวนี้ให้ functionality ที่เหมือนกันแต่ก็ต่างกันในหลักปฏิบัติที่แต่ละแนวทางยึดถือ ซึ่งมีความสำคัญต่อลำดับในการออกแบบของแต่ละ DBMSs และประสิทธิภาพที่แต่ละ features ให้การสนับสนุน ซึ่งแสดงโดยการเปรียบเทียบต่อไปนี้

- OODBMSs มีจุดประสงค์ในความสำเร็จของรวมตัวกับ programming language เช่น C++ หรือ Smalltalk แต่การรวมตัวเข้าด้วยกันไม่ได้เป็นจุดประสงค์หลักของ ORDBMS SQL3(เช่นเดียวกันกับ SQL92) ให้เราสามารถซ้อน SQL command เข้าไว้ใน host language ได้ แต่การ interface จะเห็นชัดเจนมากจาก SQL programmer

- OODBMS มีจุดมุ่งหมายไปยัง applications เมื่อหนึ่ง object-centric viewpoint เหมาะสม นั่นคือผู้ใช้จะค้นคืน objects จำนวนหนึ่งและทำงานกับมันเป็นช่วงเวลานาน ซึ่ง related objects (คือ objects ที่ถูกอ้างอิงโดย original objects) จะถูก fetch เป็นครั้งคราว ซึ่ง objects อาจมีขนาดใหญ่และอาจต้องถูก fetch เป็นส่วนๆ ดังนั้นต้องให้ความสนใจแก่ buffering parts ของ objects ซึ่งคาดว่า applications ส่วนใหญ่จะสามารถ cache objects ที่ต้องการใน memory ได้เมื่อ object ถูกอ่านหนึ่งครั้งจาก disk ดังนั้นแนวทางนี้คือการสร้าง references ไปยัง in-memory objects อย่างมีประสิทธิภาพ สำหรับ transactions ที่มีการทำงานนานและจะถูก lock ไว้อาจทำให้ประสิทธิภาพของระบบลดลงได้จึงควรมีกระบวนการของ Two Phase locking เพื่อใช้กับ transactions

  ส่วน ORDBMS จะถูกทำให้เหมาะสมกับ applications ซึ่ง large data collection จะเป็นจุดสนใจของระบบ ถึงแม้ว่า objects นั้นๆ จะมีโครงสร้างมากมายและขนาดใหญ่ ก็ตามแนวคิดนี้ระบบจะคาดว่า applications มีการค้นคืน data จาก disk บ่อยครั้ง ดังนั้น optimizing disk accesses จึงเป็นส่วนสำคัญสำหรับประสิทธิภาพการทำงานของระบบ โดยสมมุติว่า transaction มีขนาดสั้น สำหรับเทคนิคที่ใช้สำหรับ concurrency control และ recovery จะเป็นเทคนิคเดียวกับที่ใช้ใน RDBMS
- ความสะดวกของ query ใน OQL ไม่ได้รับการสนับสนุนอย่างมีประสิทธิภาพจาก OODBMSs ส่วนใหญ่ในขณะที่ความสะดวกของ query เป็น centerpiece ของ ORDBMS ในบางขอบเขต สภาวะนี้คือผลที่เกิดจากมุมมองที่ต่างกันในการพัฒนาระบบเพื่อให้การประเมินค่าที่นัยสำคัญ เราจะต้องผลการทำงานของระบบที่ถูก optimized เข้าชนิดต่างๆ ของ applications

• บทส่งท้าย

Object-database systems มีนัยสำคัญในการขยายแบบชนิดต่างๆ จากที่พบใน relational database systems ซึ่ง Object-Relational DBMSs ทำสิ่งนี้โดยการเริ่มต้นกับ SQL และ relations และเพิ่ม new features เช่น ADTs, type constructors, object identity และ inheritance ส่วน Object-Oriented DBMSs ทำสิ่งนี้โดยเริ่มกับ object-oriented languages เช่น C++ และเพิ่ม DBMS facilities เช่น persistent data, indexes, concurrency และ recovery

Object-relational database system ให้ functionality ที่มีอยู่ทั้งหมดใน relational system

และ new object modeling features ซึ่ง features ใหม่ๆ เหล่านี้ได้ย้ายเงื่อนไขบางอย่างของ relational database design ออกไป ตัวอย่างเช่น table ใน ORDBMS ไม่จำเป็นต้องอยู่ในรูปของ 1NF และ users สามารถกำหนด data types แบบใหม่ๆ ได้ ซึ่ง features ใหม่ๆ เหล่านี้เปลี่ยนกฎของ database design และทางเลือกสำหรับการออกแบบใหม่ๆ จะต้องถูกพิจารณา

ข้อดีที่สำคัญของ object-database systems คือมันสามารถจัดเก็บ code ไว้กับ data ได้ ซึ่ง ADT methods จะถูกรวมไว้ใน ฐานข้อมูล และเซตสำหรับ methods เหล่านั้นสามารถค้นหาได้โดยการส่ง query ไปยัง database catalogs สำหรับมุมมองของ ORDBMS อาจดูคล้ายกับ software repository ซึ่งมี built-in query สำหรับสนับสนุนการระบุ software modules และ methods ที่มีอยู่เพื่อสร้าง new applications ได้

ข้อดีสำหรับการจัดเก็บ code ไว้ใน ฐานข้อมูล คือ ADT นำ code ไปยังข้อมูลมากกว่าที่จะนำข้อมูลไปยัง code ตัวอย่างเช่นถ้ามี object ขนาดใหญ่ปรากฏใน ฐานข้อมูล, ADT method ที่ใช้สำหรับ compress มันสามารถใช้ได้ที่ database server แทนที่จะเก็บที่ client (เพื่อเป็นการป้องกัน expensive network overhead ในการส่งข้อมูลไปยัง client) เช่นเดียวกัน customized methods สามารถถูกใช้ในการเลือกป้องกันการส่ง tuples จาก server ไปยัง client โดยไม่จำเป็น ข้อดีนี้เป็นส่วนที่ไม่มีอยู่ใน RDBMS

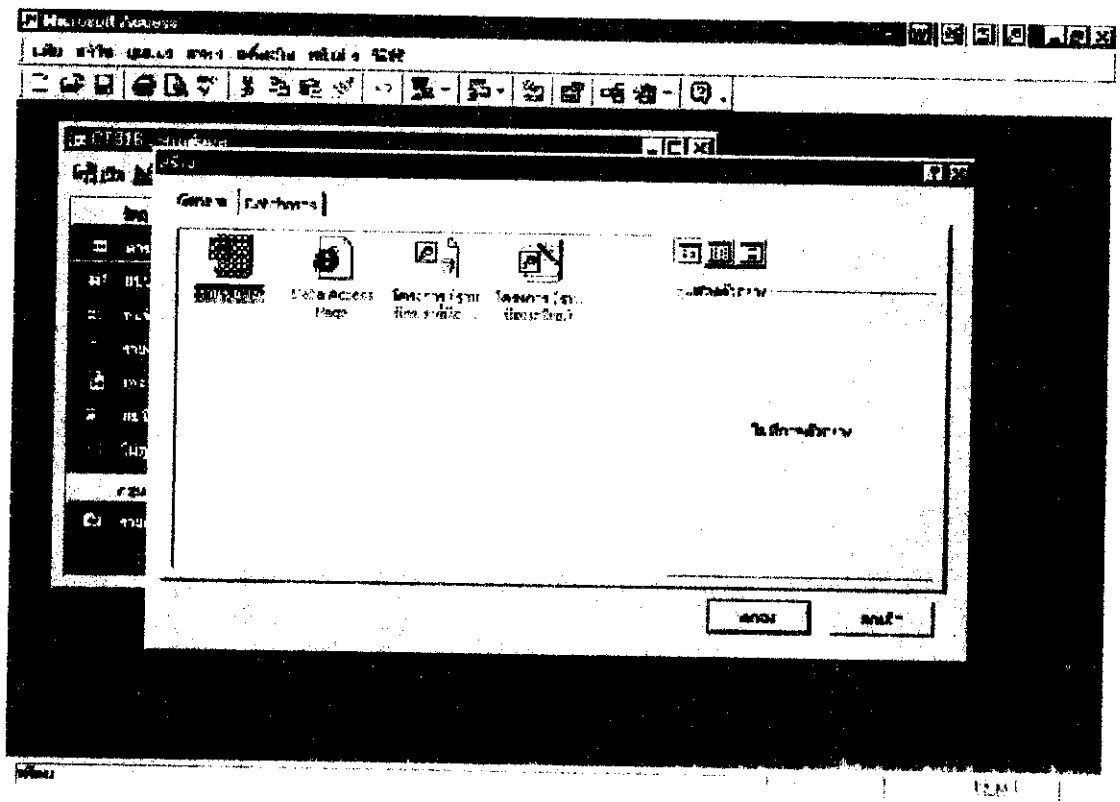
สุดท้ายคือข้อดีทาง logic สำหรับการเก็บ code ใน DBMS หรือในการรักษาความสอดคล้องกันของข้อมูล ซึ่ง DBMS สามารถจัดการกับเงื่อนไขเหล่านี้ได้อย่างอัตโนมัติ สำหรับ ORDBMS เงื่อนไขเหล่านี้รวมถึงเงื่อนไขที่มาจาก user-defined methods ด้วย ในอีกด้านหนึ่ง RDBMS อาจจะมี complex logic เหลืออยู่ใน client applications ซึ่งขัดขวางการทำงานในส่วน data semantics ของ DBMS

จึงมีเหตุผลพอที่จะคาดได้ว่า RDBMSs ทั้งหมดที่อยู่ในตลาดปัจจุบันจะมี ORDBMS functionality ภายในไม่กี่ปีข้างหน้า แต่โชคไม่ดีที่มีปัญหาต่างๆ ในการสร้างยังไม่มีการแก้ไขสมบูรณ์และทำได้รวดเร็วนัก เราจึงคาดได้ว่า features ที่ใหม่กว่านี้ของ ORDBMS ที่จะมีในอนาคตอาจไม่มีประสิทธิภาพเท่ากับการทำงานของ relational processing ที่มีอยู่ในปัจจุบัน อย่างน้อยที่สุดก็จะเป็นช่วงขณะหนึ่ง เช่นเดียวกันกับการศึกษาในเรื่องของ ORDBMSs ที่เพิ่งเริ่มมีขึ้นเมื่อไม่นานนี้ เราจึงคาดได้ว่าความซับซ้อนในการออกแบบฐานข้อมูลในอนาคตจะเพิ่มขึ้นตามพัฒนาการของ ORDBMS ซึ่งเช่นเดียวกันกับ technology อื่นๆ object-databases จะให้ส่วนผสมของ features ที่เพิ่มขึ้นและความคิดเห็นในแบบต่างๆ

หมายเหตุ : สำหรับรายละเอียดของมาตรฐาน SQL3 และ SQL4 สามารถ download ได้ที่ URL ftp://jcry.ecc.umassd.edu/isowg3/.

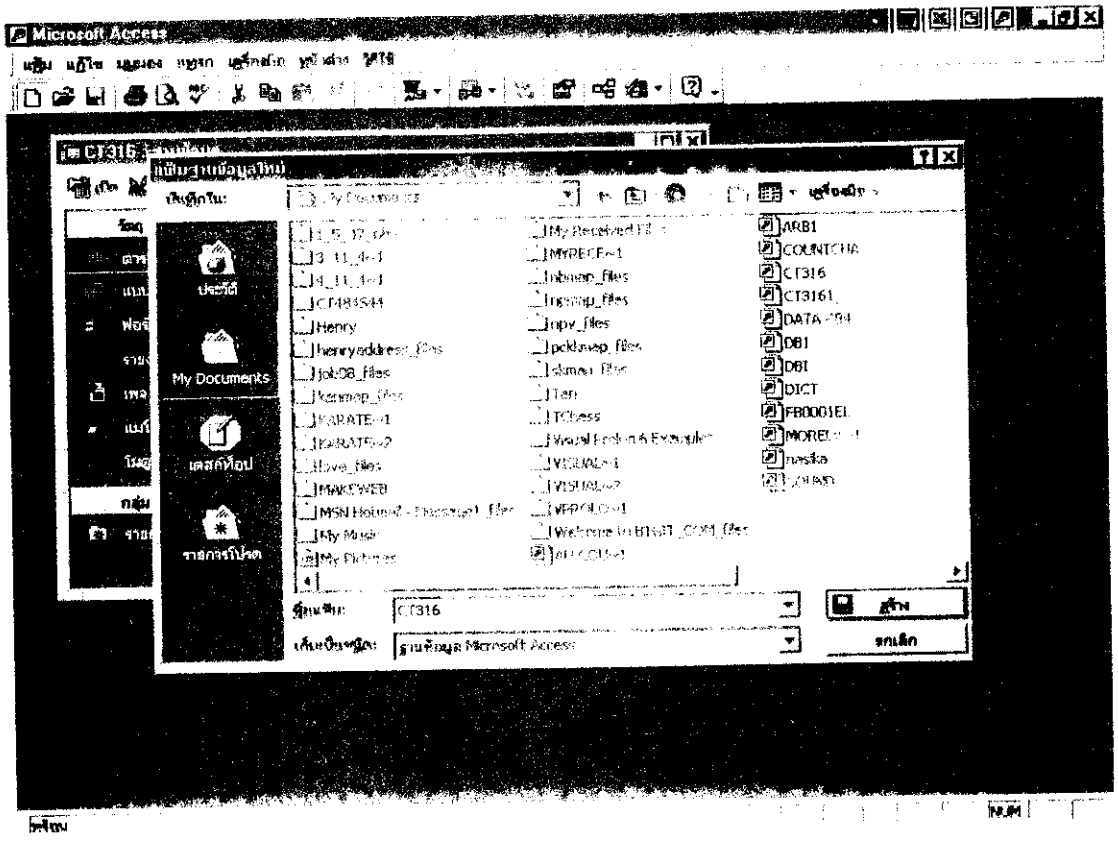
การทำงานเกี่ยวกับฐานข้อมูลโดย Microsoft Access

การสร้างฐานข้อมูลจะสร้างโดยใช้เมนู เริ่มต้นโดย เรียกใช้ Microsoft Access สร้าง ฐาน ข้อมูลใหม่ เลือกจาก ฐานข้อมูลในหน้าต่าง สร้าง



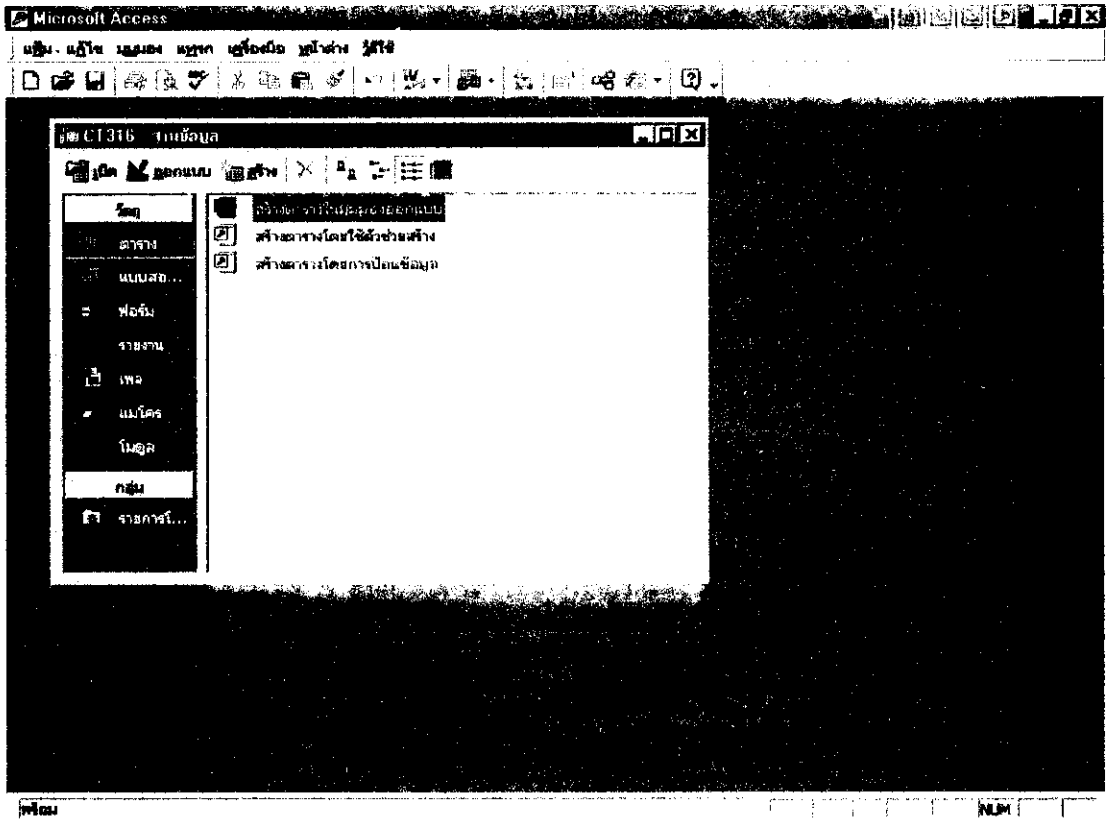
CT 316 (S) 201

ตั้งชื่อแฟ้มฐานข้อมูลจากแบบฝึกหัดก่อน แฟ้มฐานข้อมูลใหม่

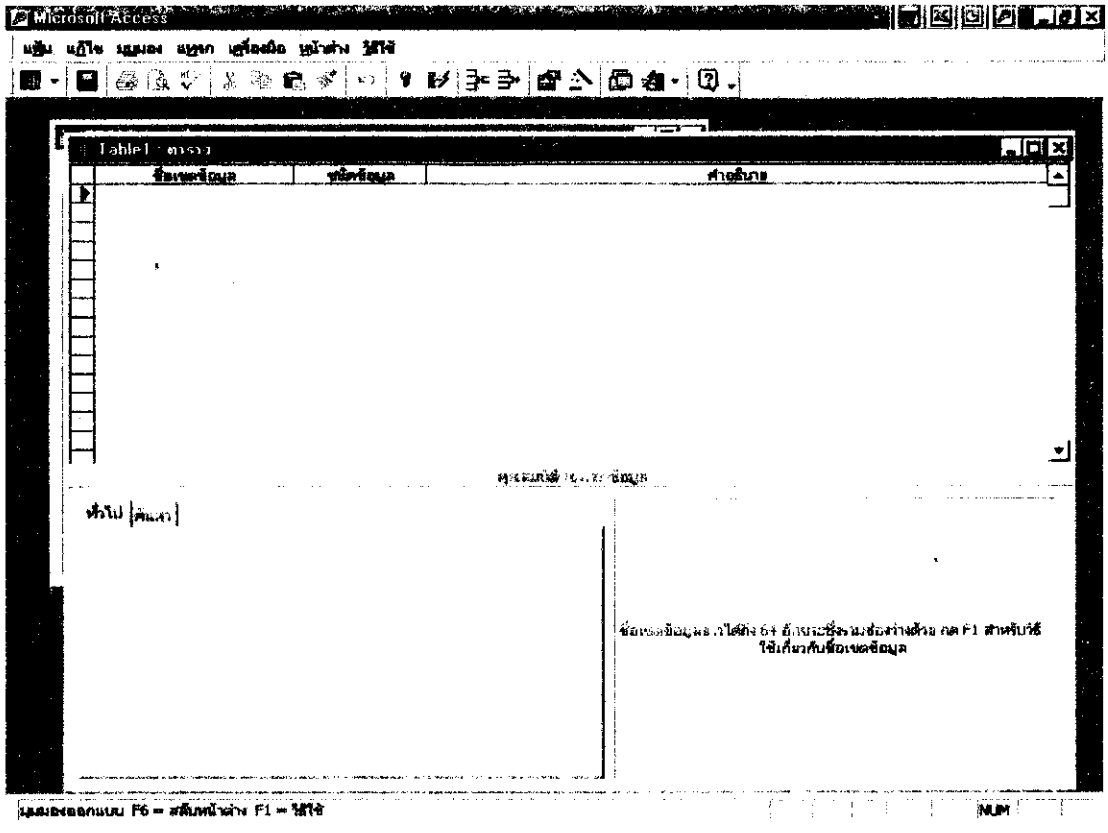


ในที่นี้ตั้งชื่อเป็น CT316

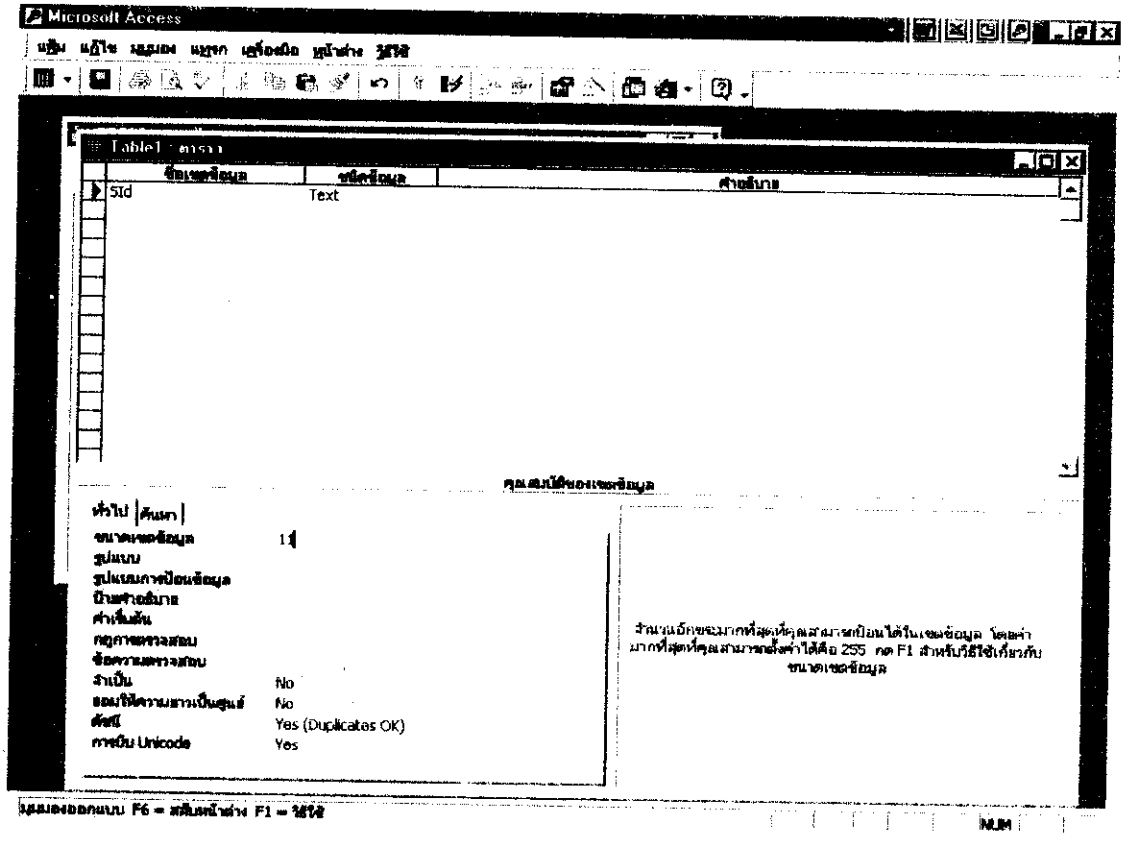
แล้วจะเข้าสู่หน้าต่าง



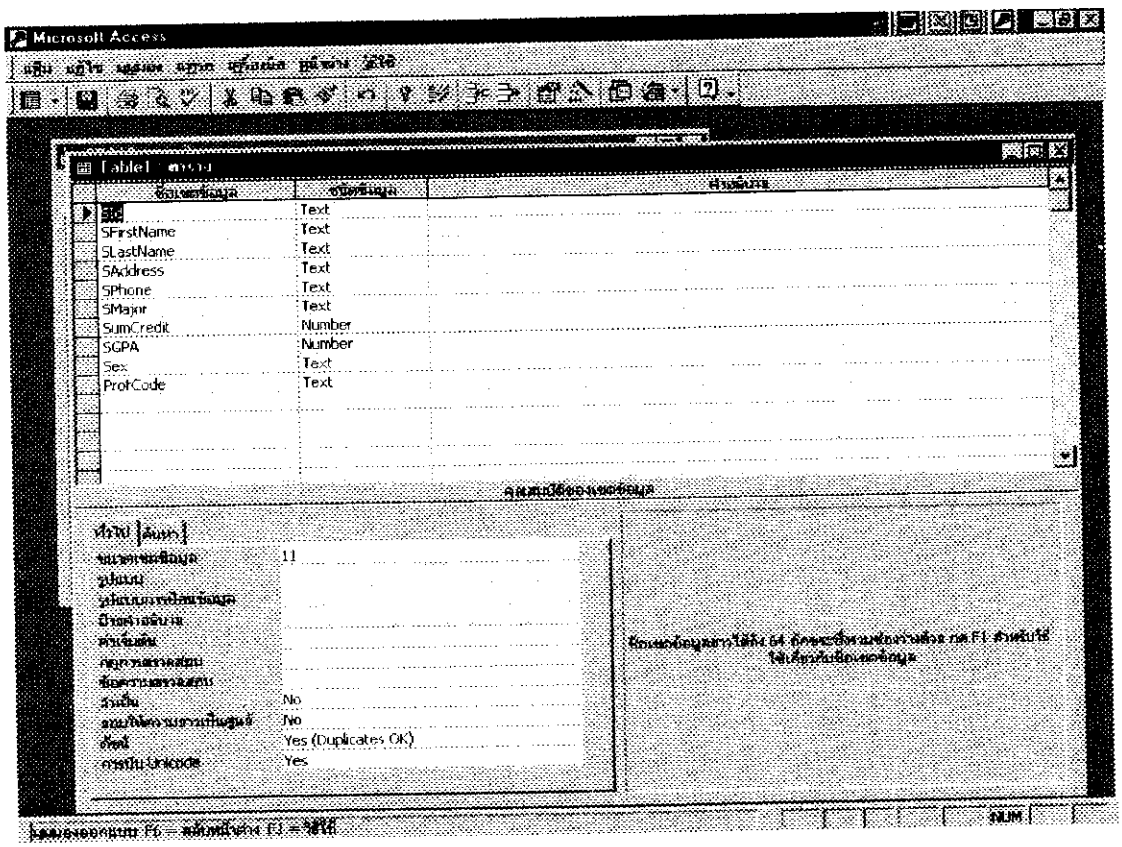
แล้วทำการสร้างตารางโดยเลือก “สร้างตารางในมุมมองออกแบบ” เพื่อกำหนดค่าของประเภทและขนาดของฟิลด์



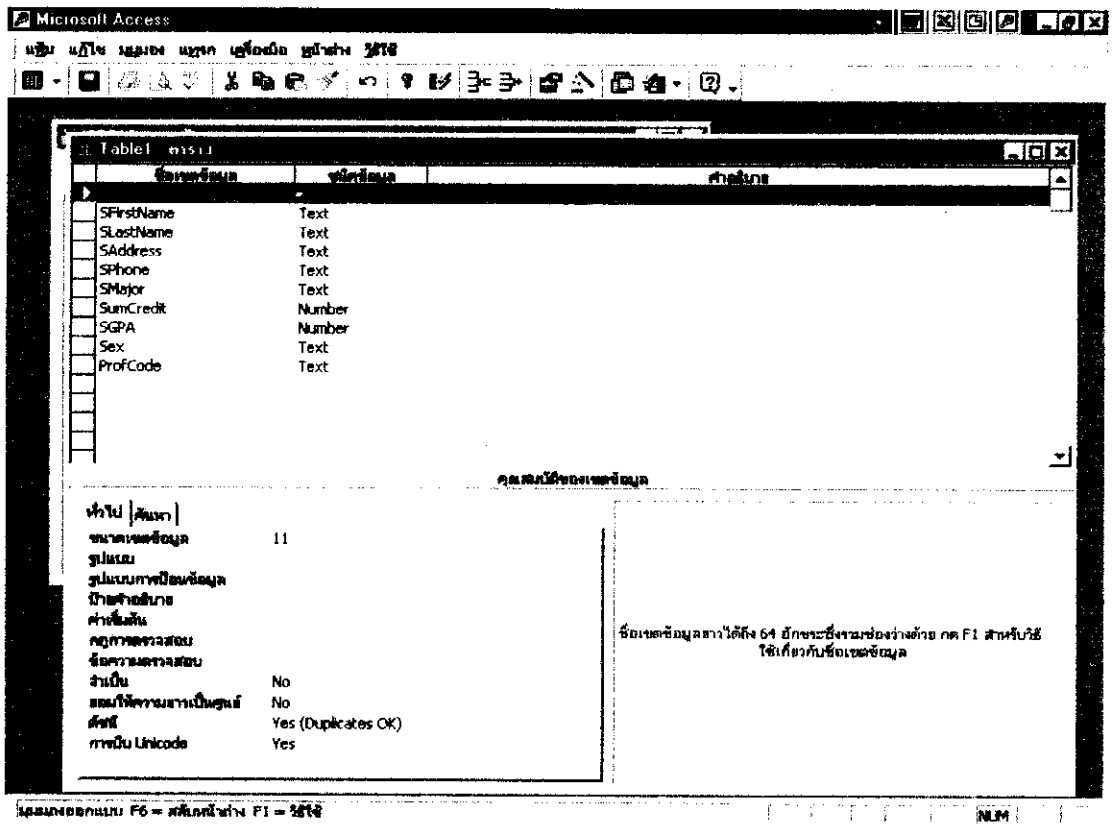
เมื่อใส่ข้อมูลของฟิลด์



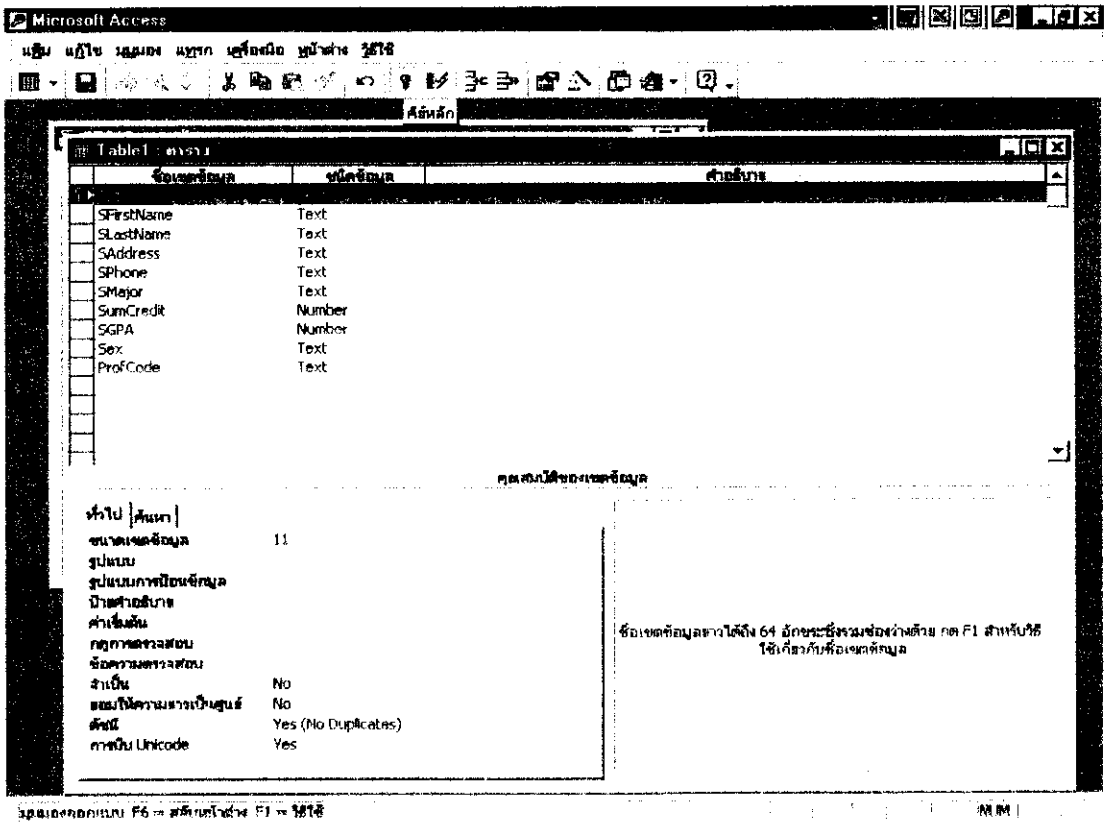
หลังจากใส่ข้อมูลของฟิลด์ทั้งหมด



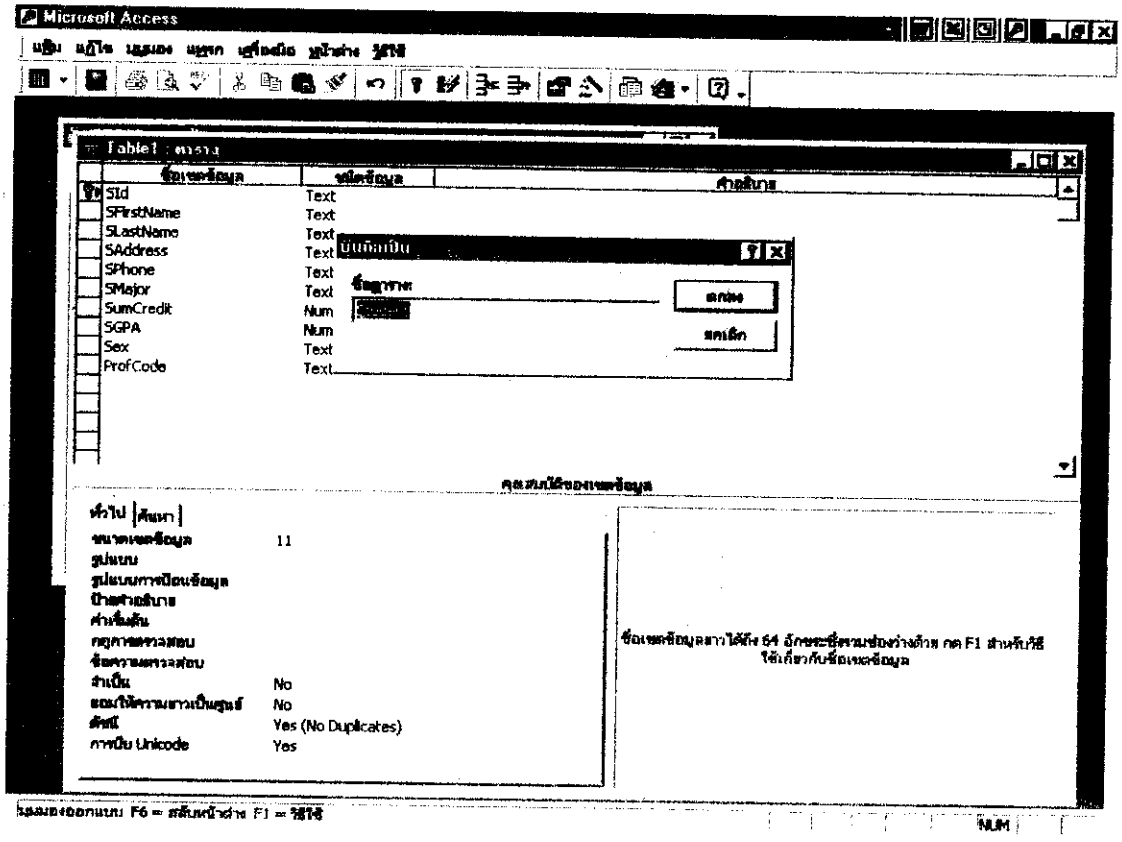
ก็กำหนดฟิลด์ที่เป็น key โดยเลือกฟิลด์นั้นจะมากกว่า 1 ฟิลด์ก็ได้แล้วกดปุ่ม 



จะมีรูปลูกกุญแจอยู่หน้าชื่อฟิลด์นั้นๆ



แล้วทำการตั้งชื่อตาราง ให้ชื่อ Student



ในการสร้างตารางอื่นๆ ก็ใช้วิธีเดียวกัน

โครงสร้างของตารางที่เหลือ

Table Grade

SId	Text	ขนาด 11
SubjectCode	Text	ขนาด 5
Grade	Text	ขนาด 2
Semester	Text	ขนาด 10

Table Professor

ProfCode	Text	ขนาด 10
ProFirstName	Text	ขนาด 50
ProLastName	Text	ขนาด 50
DeptCode	Text	ขนาด 50
ProfOffice	Text	ขนาด 50

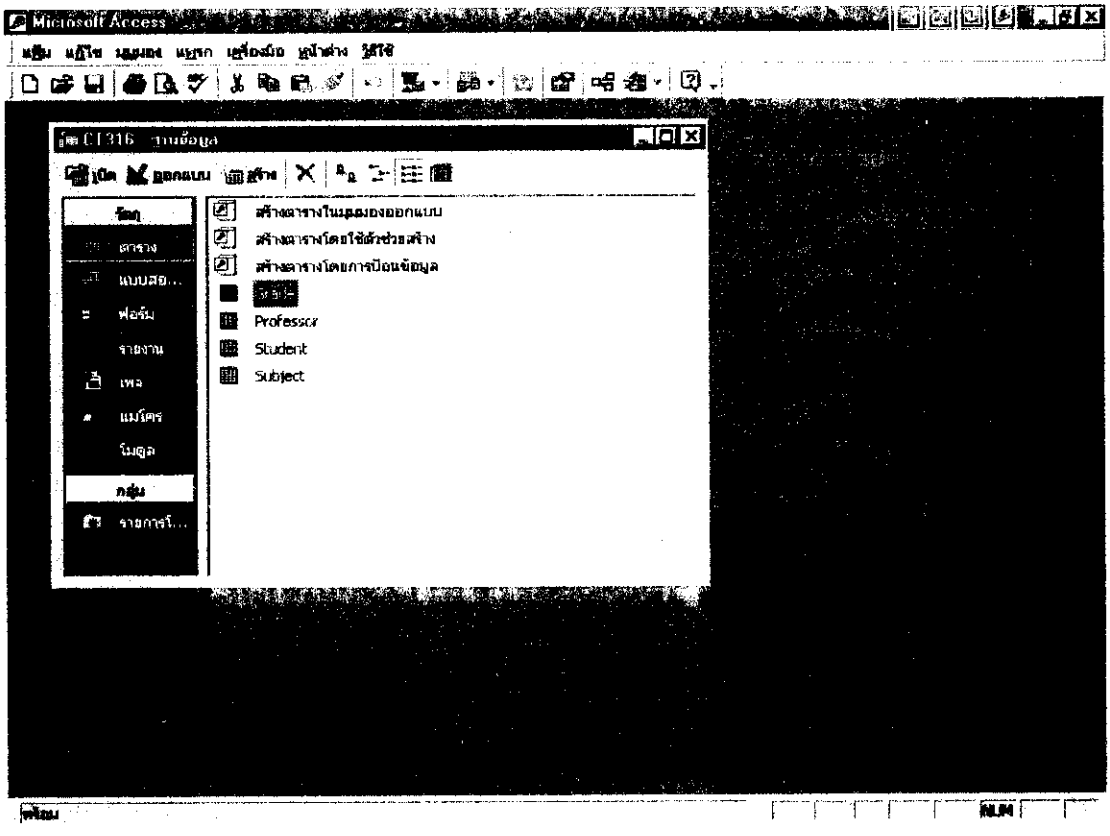
Table Student

SId	Text	ขนาด 11
SFirstName	Text	ขนาด 50
SLastName	Text	ขนาด 50
SAddress	Text	ขนาด 127
SPhone	Text	ขนาด 11
SMajor	Text	ขนาด 20
SumCredit	Number	Integer
SGPA	Number	Single
Sex	Text	ขนาด 1
ProfCode	Text	ขนาด 10

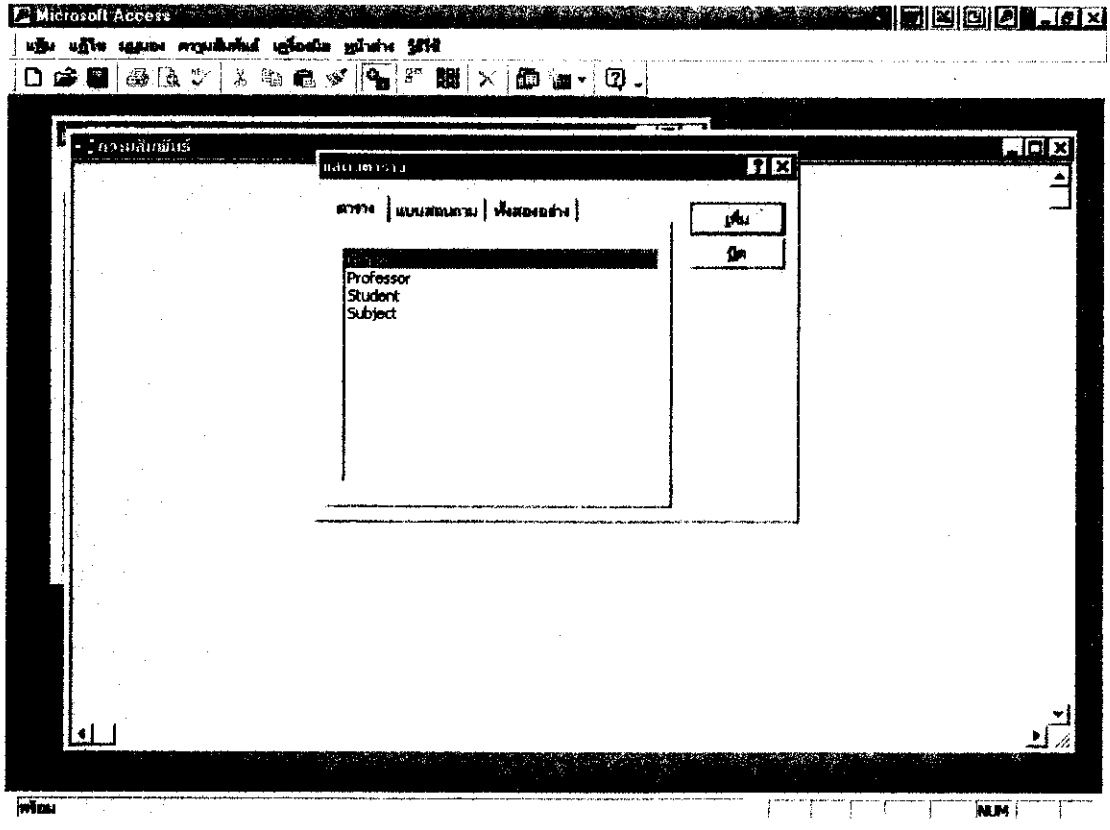
Table Subject

SubjectCode	Text	ขนาด 5
SubjectName	Text	ขนาด 127
SubjectCredit	Number	Byte
ProfCode	Text	ขนาด 10

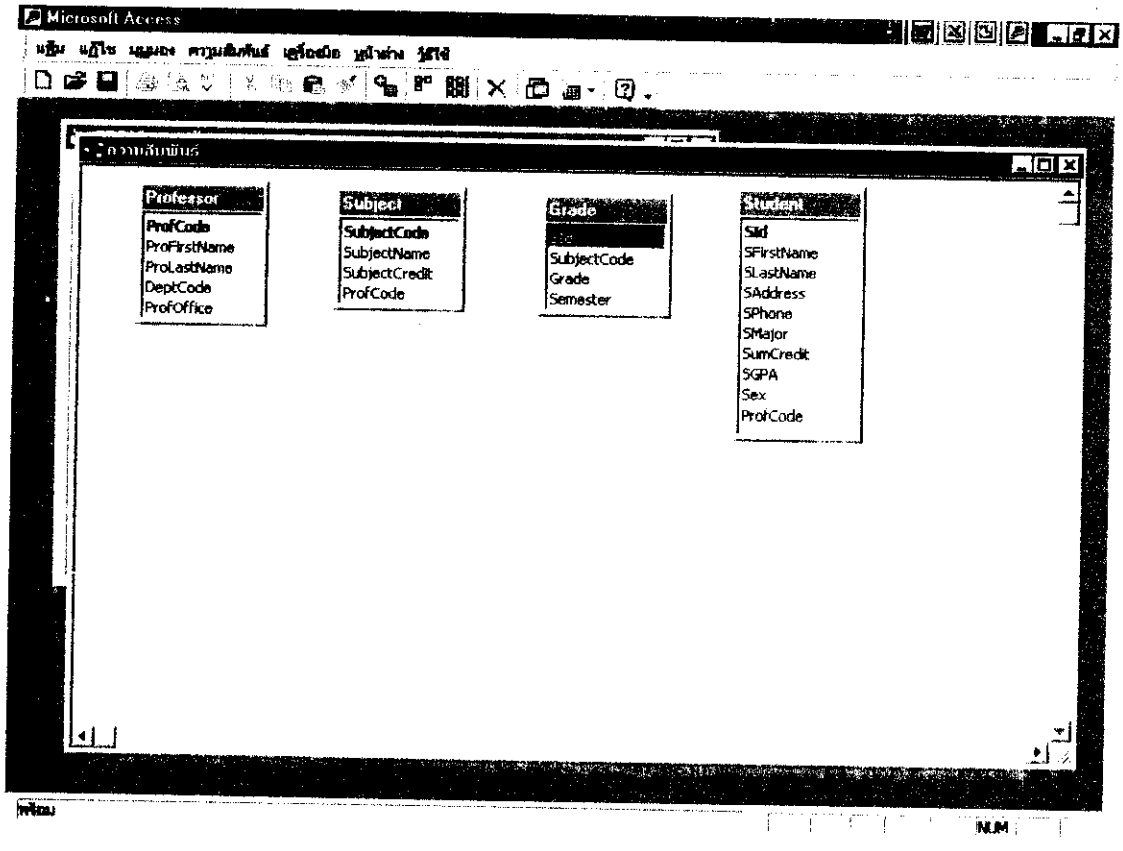
หลังจากสร้างตารางที่เหลือแล้วจะได้หน้าต่างดังนี้



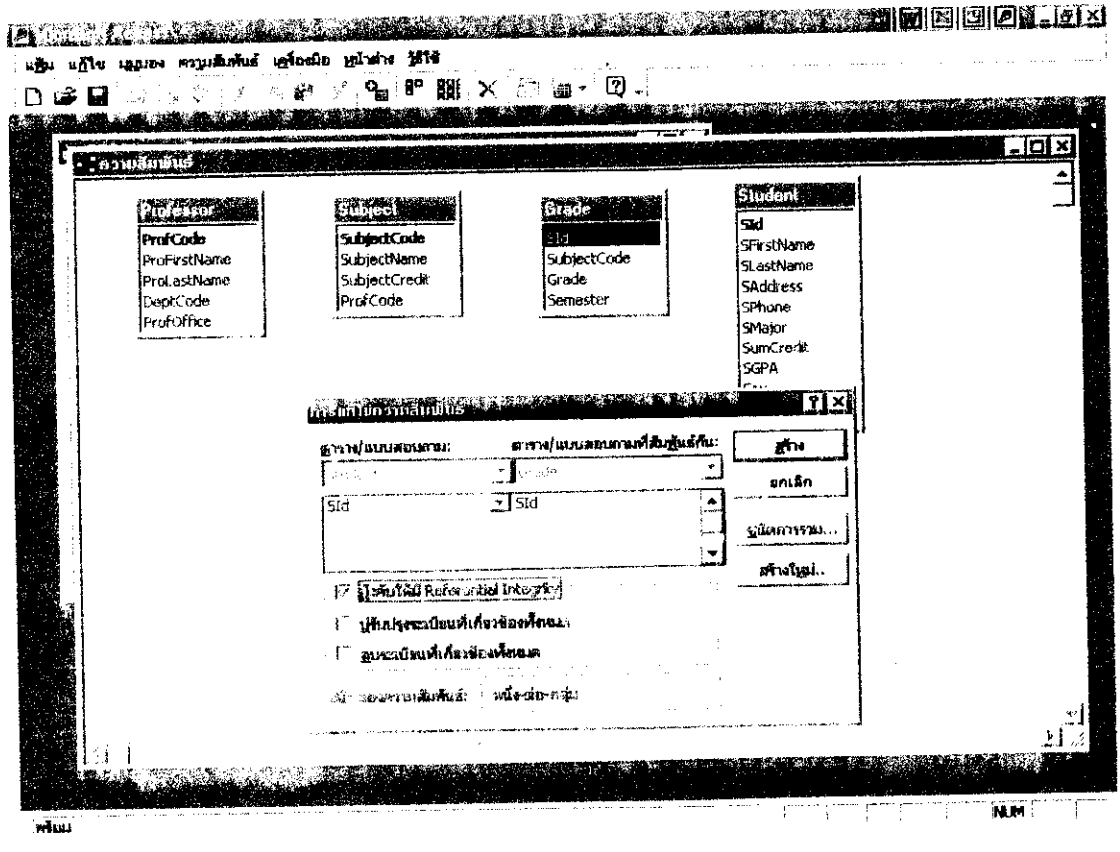
เมื่อได้ทุกตารางแล้วก็ต้องสร้างความสัมพันธ์ของตารางต่างๆ โดยกดปุ่มความสัมพันธ์  จะได้หน้าต่างนี้

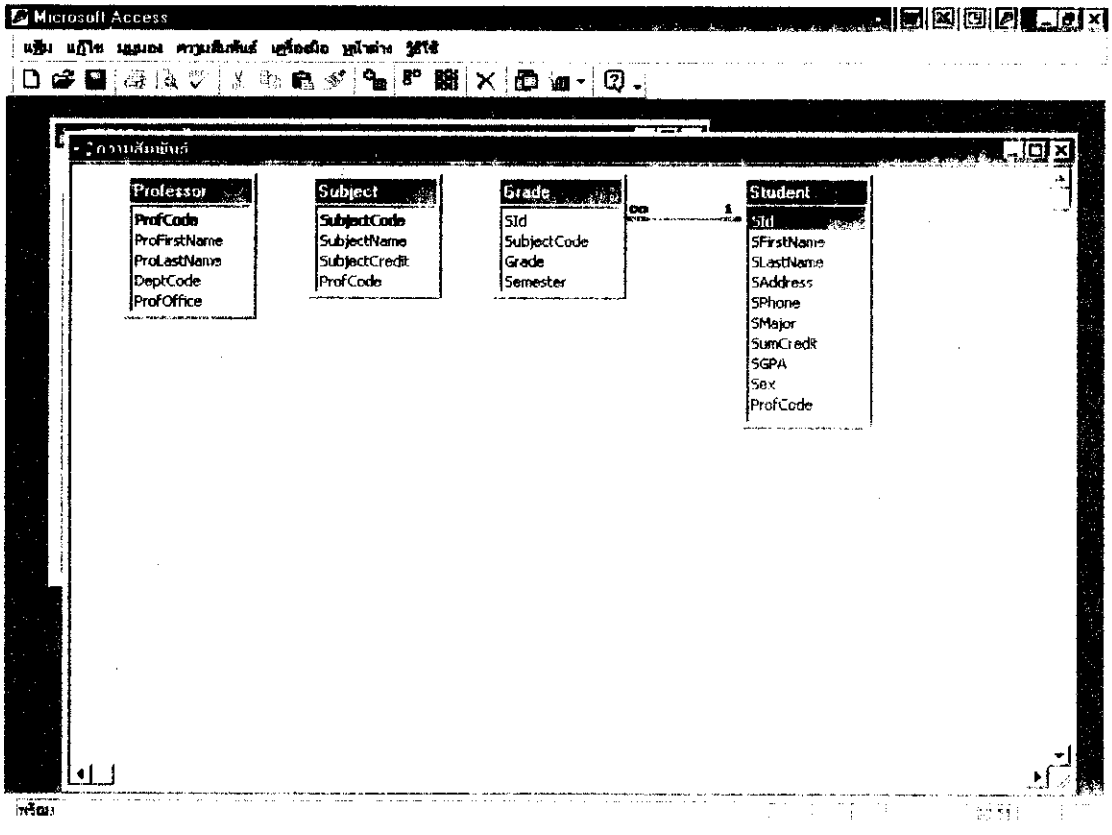


แล้วให้ทำการเพิ่มตารางที่จะสร้างความสัมพันธ์กัน

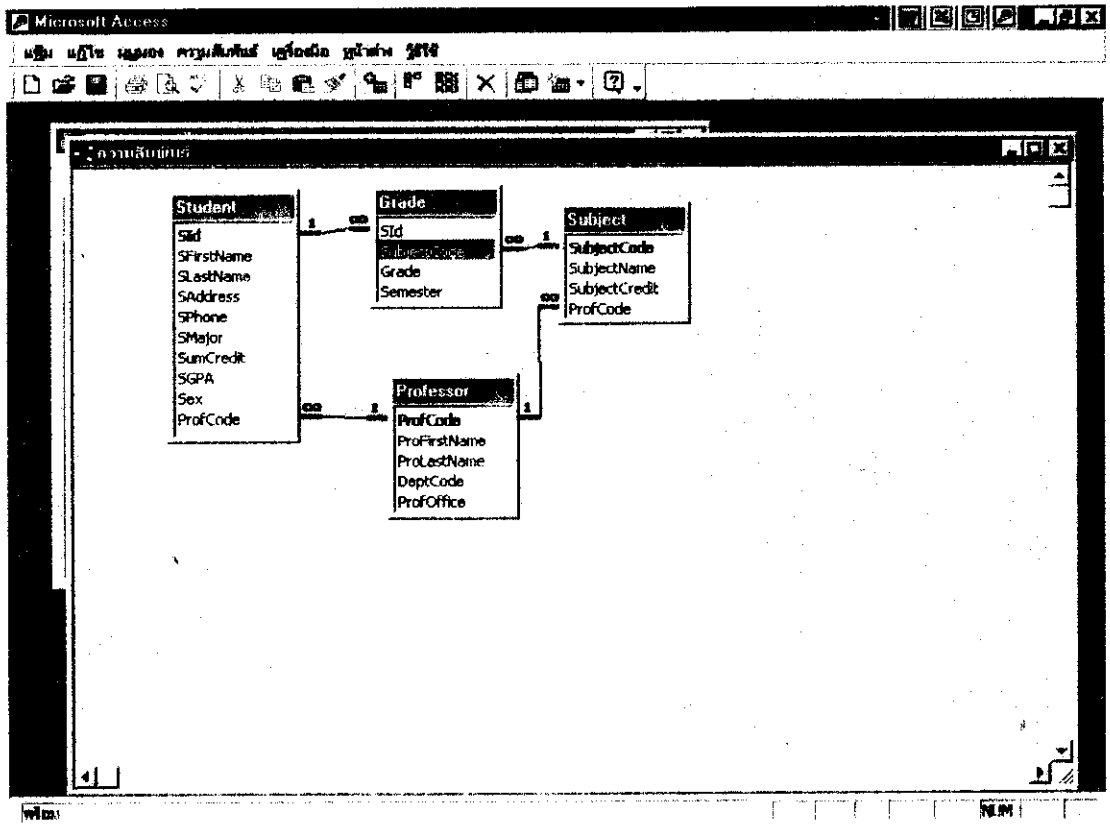


ถ้าต้องการสร้างความสัมพันธ์ของแต่ละตารางก็เลือกฟิลด์ชื่อเดียวกันของแต่ละตารางแล้วเลือกใน ในที่นี้เลือกตาราง Student กับตาราง Grade ฟิลด์ Sid ถ้าต้องการให้แสดงความสัมพันธ์ระหว่างตาราง ก็ให้เลือกช่อง Referential Integrity

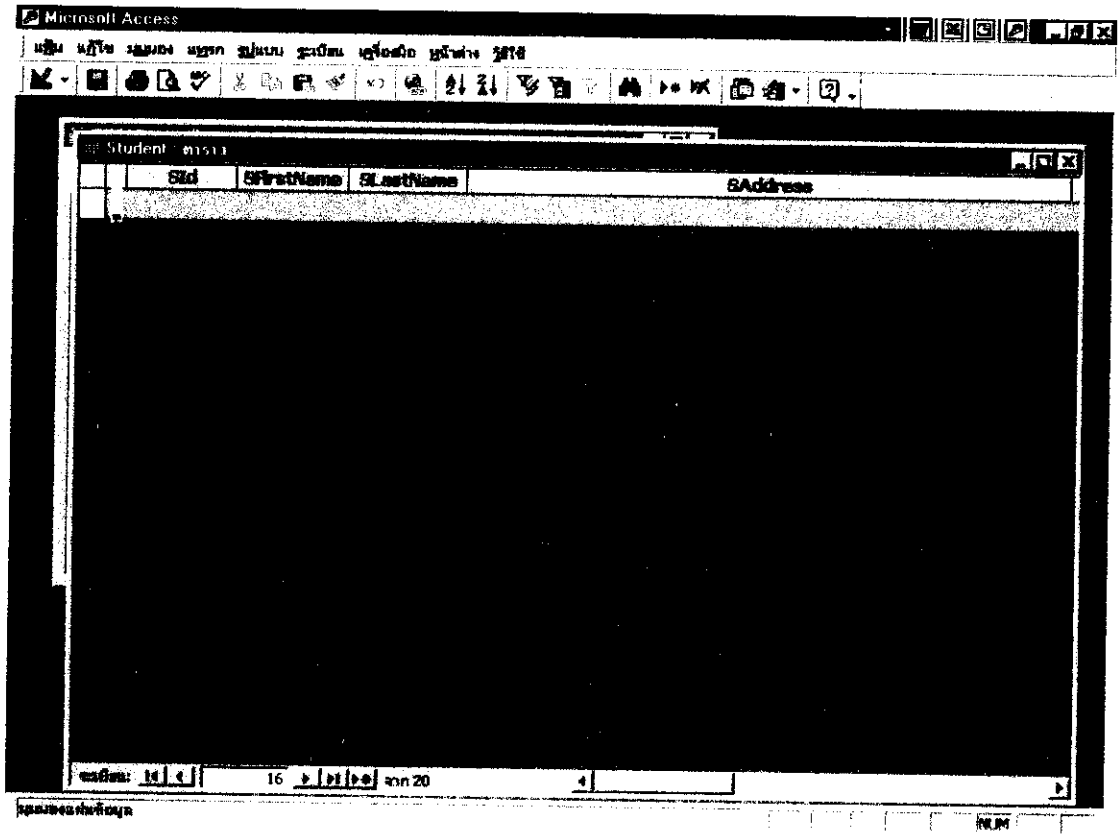



Microsoft Access
ความสัมพันธ์
Professor
ProfCode
ProFirstName
ProLastName
DeptCode
ProfOffice
Subject
SubjectCode
SubjectName
SubjectCredit
ProfCode
Grade
SId
SubjectCode
Grade
Semester
Student
SId
SFirstName
SLastName
SAddress
SPhone
SMajor
SumCredit
SGPA
Sex
ProfCode

ส่วนการสร้างความสัมพันธ์ระหว่างตารางอื่นๆ ก็เหมือนๆ กัน



เสร็จแล้วถ้าต้องการจะป้อนข้อมูลก็เลือกตารางที่จะป้อน ในภาพเลือก Student จะได้หน้าต่างดังนี้



ข้อมูล Table Student

SId	SFirstName	SLastName	SAddress	SPhone	SMajor	Sum Credit	SGPA	Sex	ProfCode
4005100275	จุฑารส	ฤกษ์สุริย	25 ม.1 ถ.ซักพระ ตลิ่งชัน กรุงเทพฯ 10170	0-24348897	คอมพิวเตอร์	138	2.81	ญ	3005401
4005171513	เดชาวุท	คงสุขสรรค์	โครงการพนาสิน ถ.รามคำแหง หัวหมาก บางกะปิ กรุงเทพฯ 10240	0-27184437	คอมพิวเตอร์	143	2.66	ช	2305035
4005217951	อาภรณ์	เรืองชัย	894 ถ.เทอดไทย บางยี่เรือ ธนบุรี กรุงเทพฯ 10600	0-24721119	คอมพิวเตอร์	135	2.55	ญ	2505054
4005768539	พรชัย	ตันติปัญญา	55/3 ม.1 ซ.เพชรเกษม 91 ถ.เพชรเกษม บางแค เหนือ ภาษีเจริญ กรุงเทพฯ 10160	0-25876599	คอมพิวเตอร์	136	2.77	ช	2505054
4102546937	ชิดชนก	ทองสี	3/2 ม.4 หมู่บ้านวงษ์ทอง ถ.เอกชัย บางบอน บางขุนเทียน กรุงเทพฯ 10150	0-24587511	บัญชี	135	2.99	ญ	
4105891374	รัชฎา	เจริญเมืองศรี	127/10 ซ.บางสะแกนอก ถ.เทอดไท บางยี่เรือ ธนบุรี กรุงเทพฯ 10600	0-24728854	คอมพิวเตอร์	139	2.75	ญ	
4106774985	บันเทิง	ยืนสุข	315 ม.9 บางแค เหนือ ภาษีเจริญ กรุงเทพฯ 10160	0-28087751	บริหารรัฐกิจ	140	3.11	ช	
4301587621	กมล	จัตรพิมลกุล	351/22 ถ.พญาไท พญาไท ราชเทวี กรุงเทพฯ 10400	0-22784612	นิติศาสตร์	101	2.6	ช	

SId	SFirstName	SLastName	SAddress	SPhone	SMajor	Sum Credit	SGPA	Sex	ProfCode
4305231190	วีระ	ศรีสุข	9/17 ม.วงศ์ชัยนคร ถ.รัตนาธิเบศน์ บางรักใหญ่ บางบัวทอง นนทบุรี 11110	0-29851119	คอมพิวเตอร์	111	2.5	ช	3505498
4305718913	สามารถ	สระน้อย	201 ม.3 ซ.79 ถ.เพชรเกษม หนองค้างพลู หนองแขม กรุงเทพฯ 10160	0-24441159	คอมพิวเตอร์	135	2.71	ช	3505498
4305721511	นิพนธ์	คงสุวรรณ	448/17 ถ.เทอดไท บางยี่เรือ ธนบุรี กรุงเทพฯ 10600	0-28034431	วิจัยดำเนินงาน	94	2.79	ช	4305507
4305813528	วีระ	จันทรตรา	366 ซ.เพชรเกษม19 ถ.เพชรเกษม วัดท่าพระ บางกอกใหญ่ กรุงเทพฯ 10600	0-28015572	สถิติ	95	2.6	ช	4305507
4307128471	กิตติพงษ์	คงสุข	78/11 ถ.เทอดไท บางยี่เรือ ธนบุรี กรุงเทพฯ 10600	0-28914571	เศรษฐศาสตร์การเงิน	103	2.9	ช	4305507
4402764559	สุวรรณา	ราษฎร์รักษา	351 ถ.ศรีเวียง สีลม บางรัก กรุงเทพฯ	0-28073766	การตลาด	95	2.7	ญ	
4404132155	วงศกร	ชื่นชม	44 ถ.ตากสิน จอมทอง กรุงเทพฯ 10150	0-24774715	จิตวิทยา	135	3.25	ช	
4405087659	พวงเพ็ญ	ประจำศิลธรรม	124 ถ.ประชาอุทิศ ต.ทุ่งครุ ราษฎร์บูรณะ กรุงเทพฯ 10140	0-22684531	คณิตศาสตร์	94	2.79	ญ	
4405113716	ณรงค์	สุตาภรณ์	205 ต.บางม่วง อ.บางใหญ่ จ.นนทบุรี 11140	0-29858843	ฟิสิกส์	105	2.71	ช	2305035
4405115732	มุกดา	สมศรี	55 ม.3 ม.อัมรินทร์นิเวศน์ 2	0-26375555	คอมพิวเตอร์	94	2.8	ญ	3005401

SId	SFirstName	SLastName	SAddress	SPhone	SMajor	Sum Credit	SGPA	Sex	ProfCode
			ถ.สุขาภิบาล 1 คลองกุ่ม บึงกุ่ม กรุงเทพฯ 10230						
4505002462	ธนจักร	ชำนาญ	40/165 ม.เอส.เค ซ.76 ถ.เอกชัย บางบอน บางขุนเทียน กรุงเทพฯ 10150	0-2809743	คอมพิวเตอร์	81	2.8	ช	2305035
4505871132	จรรยา	ตันติกุล	53/11 ม.3 บางมด จอมทอง กรุงเทพฯ 10150	0-24378541	คอมพิวเตอร์	80	2.67	ญ	3005401

ข้อมูล Table Professor

ProfCode	ProFirstName	ProLastName	DeptCode	ProfOffice
2305035	ระพีพรรณ	พิริยะกุล	คอมพิวเตอร์	วิทยาศาสตร์
2505054	พรชัย	จิตต์พานิชย์	คอมพิวเตอร์	วิทยาศาสตร์
3005401	ชนินทร์	วิเชียรสรรค์	คอมพิวเตอร์	วิทยาศาสตร์
3205095	ศรีจิตร	รัตนแก้วกาญจน์	คอมพิวเตอร์	วิทยาศาสตร์
3505498	อุไร	หัวไผทอง	คอมพิวเตอร์	วิทยาศาสตร์
4305507	มณีรัตน์	จรุงเดชากุล	สถิติ	วิทยาศาสตร์

ข้อมูล Table Subject

SubjectCode	SubjectName	SubjectCredit	ProfCode
BI115	Principles of Biology	3	
BI116	Biology Laboratory	1	
CH111	General Chemistry1	3	
CH112	General Chemistry 2	3	
CH113	Chemistry Laboratory 1	1	
CH114	Chemistry Laboratory 2	1	
CT105	Introduction to Computer Sciences	3	
CT203	Discrete Structure	3	
CT211	Program Design	3	2305035

SubjectCode	SubjectName	SubjectCredit	ProfCode
CT212	Programming Structure	3	3505498
CT214	Data Structure and Algorithms	3	
CT215	Computer Organization and Assembly Language	3	2505054
CT216	File Processing	3	
CT313	Theory of Computation	3	2305035
CT314	Programming Language	3	3005401
CT315	Computer Architechture	3	2505054
CT316	Database Systems	3	3005401
CT317	Numerical Methods	3	2305035
CT414	Compiler Comstruction	3	3205095
CT415	Operating Systems	3	3205095
CT417	Data Communications and Networks	3	
CT455	Digital Design	3	2505054
CT478	Management Information System	3	
CT479	System Analysis and Design	3	
CT484	Software Engineering	3	3505498
CT488	Artificial Instelligence	3	3005401
CT489	Computer Center Management	3	
CT490	Special Projects	3	
EN101	Basic Sentences and Essential Vocabulary in daily Life	3	
EN102	Sentences and Vocabulary in General Use	3	
EN201	Reading for Comprehension	3	
GM103		3	
GM203	Business Information and Communication 1	3	
GM204		3	
GM315	Office Management	3	
GM403	Small Business Management	3	
GM406	International Business	3	
IS103	Using the Library	1	
IT105	Introduction to Computer Systems	3	
IT203	Programming for Application	3	
LW104	Introduction to Law	3	
MA111	Analytic Geometry and Calculus1	3	

SubjectCode	SubjectName	SubjectCredit	ProfCode
MA112	Analytic Geometry and Calculus 2	3	
MA213	Analytic Geometry and Calculus3	3	
MA226	Matrix Theory and Linear Algebra 1	3	
OR203	Introduction to Operation Research	3	4305507
PH111	General Physics 1	3	
PH112	General Physics 2	3	
PH113	Physics Laboratory 1	1	
PH114	Physics Laboratory 2	1	
PS110	Thai Politics and Government	3	
PY101	Culture and Religions	3	
PY103	Introduction to Philosophy	3	
ST203	Principles of Statistics	3	
ST204	Introduction to Statistical Analysis	3	
TH101	Structure of Thai and Its Usage	3	

ข้อมูลการลงทะเบียนวิชา CT316 ภาค summer/45

SId	SFirstName	SLastName	Grade
4005171513	เดชาวุท	คงสุขสรรค์	P
4005217951	อาภรณ์	เรืองชัย	-
4005768539	พรชัย	ตันติปัญญา	P
4105891374	รัชฎา	เจริญเมืองศรี	P
4305231190	วีระ	ศรีสุข	G
4405115732	มุกดา	สมศรี	P
4505002462	ธนจักร	ชำนาญ	-
4505871132	จรรยา	ตันติกุล	-

ข้อมูล Table Grade

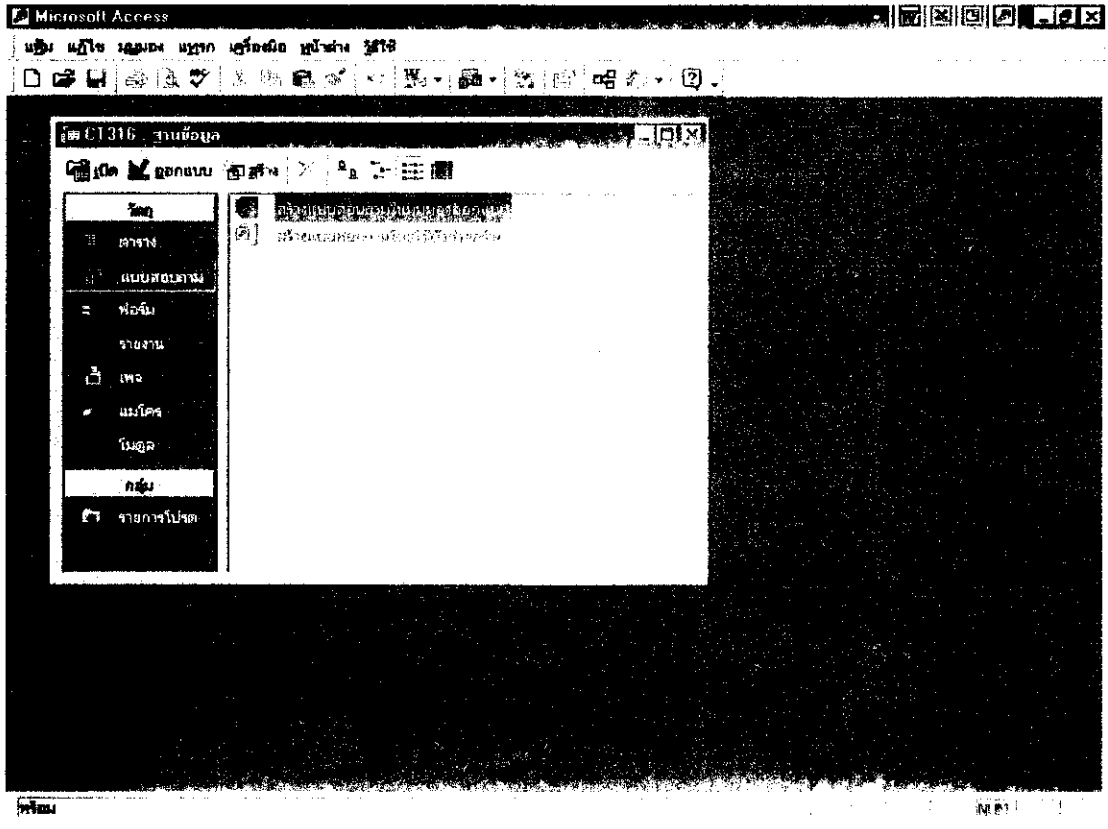
SId	SubjectCode	Grade	Semester
4005100275	BI115	-	1/40
4005100275	CH111	P	1/40
4005100275	CT105	P	1/40
4005100275	MA111	G	1/40
4005100275	PH111	-	1/40
4005100275	EN101	G	1/40
4005100275	IS103	G	1/40

SId	SubjectCode	Grade	Semester
4005100275	BI115	P	2/40
4005100275	CH112	-	2/40
4005100275	CH113	G	2/40
4005100275	CT203	P	2/40
4005100275	EN102	G	2/40
4005100275	PH111	P	2/40
4005100275	ST203	P	2/40
4005100275	MA112	G	S/40
4005100275	PY103	G	S/40
4005100275	TH101	G	S/40
4005100275	BI116	-	1/41
4005100275	CH112	P	1/41
4005100275	CT211	-	1/41
4005100275	CT212	-	1/41
4005100275	PH112	P	1/41
4005100275	PH113	P	1/41
4005100275	BI116	P	2/41
4005100275	CT211	P	2/41
4005100275	CT212	P	2/41
4005100275	CT214	-	2/41
4005100275	EN201	G	2/41
4005100275	MA213	P	2/41
4005100275	CT214	-	S/41
4005100275	PS110	G	S/41
4005100275	MA226	G	S/41
4005100275	CT214	P	1/42
4005100275	CT215	-	1/42
4005100275	CT216	G	1/42
4005100275	OR203	P	1/42
4005100275	PH114	P	1/42
4005100275	ST204	G	1/42
4005100275	CT215	P	2/42
4005100275	CT313	P	2/42
4005100275	CT314	-	2/42
4005100275	CT315	-	2/42
4005100275	CT316	-	2/42
4005100275	GM103	G	2/42
4005100275	CT314	-	S/42
4005100275	CT315	G	S/42

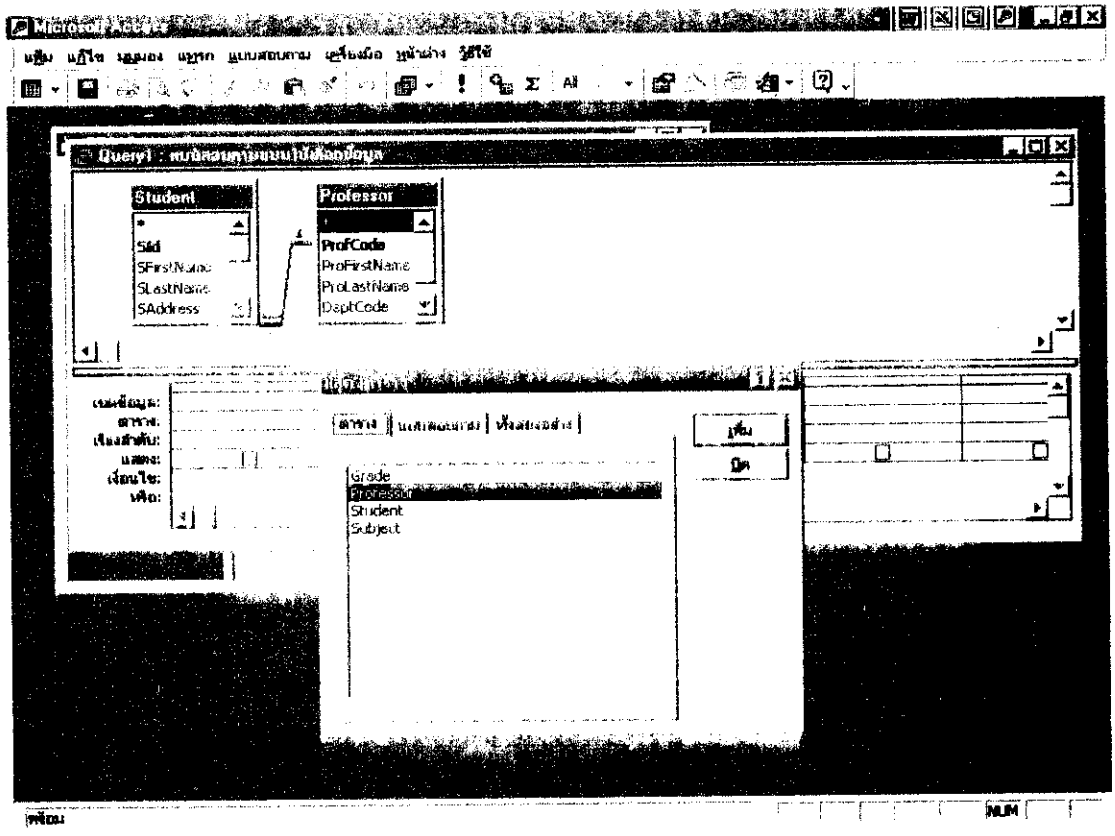
SId	SubjectCode	Grade	Semester
4005100275	GM315	G	S/42
4005100275	CT314	P	1/43
4005100275	CT316	-	1/43
4005100275	CT317	-	1/43
4005100275	CT417	-	1/43
4005100275	CT455	P	1/43
4005100275	GM203	G	1/43
4005100275	CT316	P	2/43
4005100275	CT317	-	2/43
4005100275	CT414	-	2/43
4005100275	CT417	G	2/43
4005100275	CT489	-	2/43
4005100275	GM204	G	2/43
4005100275	CT414	-	S/43
4005100275	CT489	G	S/43
4005100275	GM406	G	S/43
4005100275	CT317	-	1/44
4005100275	CT414	P	1/44
4005100275	CT478	G	1/44
4005100275	CT479	-	1/44
4005100275	CT484	-	1/44
4005100275	GM403	G	1/44
4005100275	CT317	-	2/44
4005100275	CT415	-	2/44
4005100275	CT479	P	2/44
4005100275	CT484	-	2/44
4005100275	LW104	G	2/44
4005100275	PY101	G	2/44
4005100275	CT317	P	S/44
4005100275	CT415	-	S/44
4005100275	CT484	-	S/44
4005100275	CT415	-	1/45
4005100275	CT484	-	1/45
4005100275	CT490	-	1/45
4005100275	CH114	P	1/45
4005100275	CT415	-	2/45
4005100275	CT484	-	2/45
4005100275	CT490	-	2/45
4005100275	CT415	-	S/45

SId	SubjectCode	Grade	Semester
4005100275	CT484	-	S/45
4005171513			
4005217951			
4005768539			
4102546937			
4105891374			
4106774985			
4301587621			
4305231190			
4305718913			
4305721511			
4305813528			
4307128471			
4402764559			
4404132155			
4405087659			
4405113716			
4405115732			
4505002462			
4505871132			

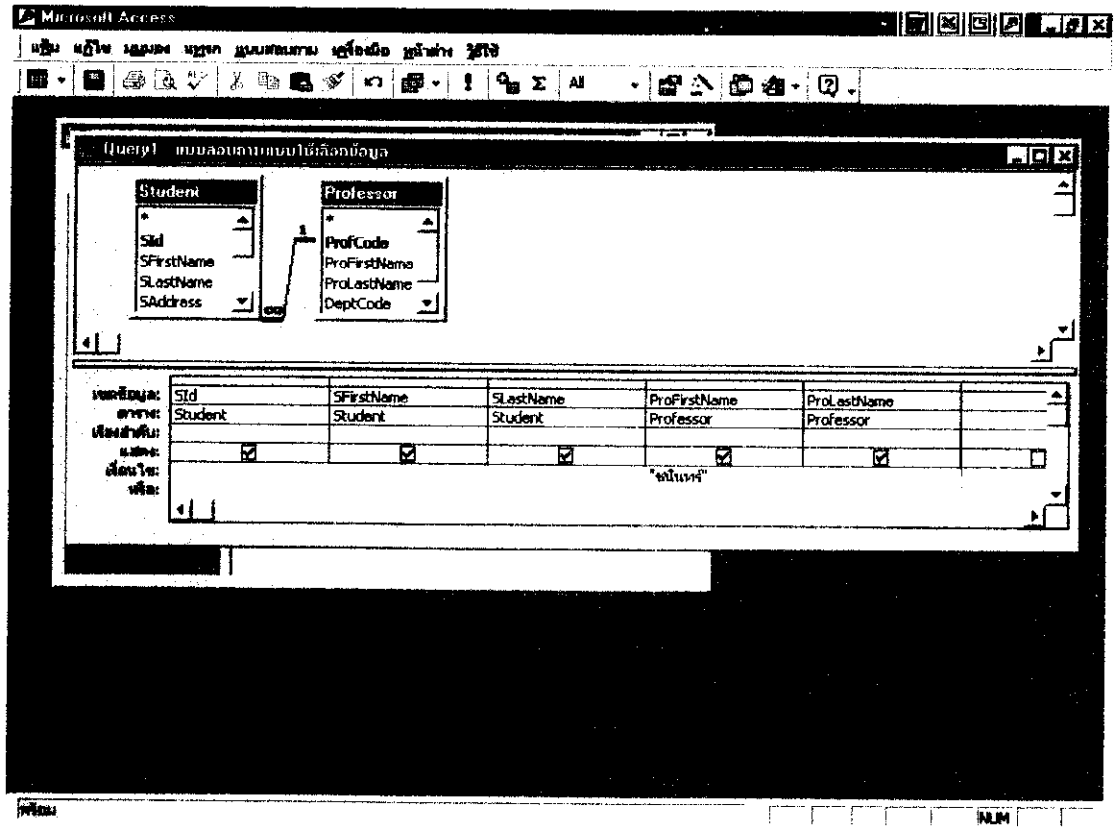
เมื่อมีข้อมูลทุกตารางแล้วก็สามารถสร้าง Query (แบบสอบถาม) ได้



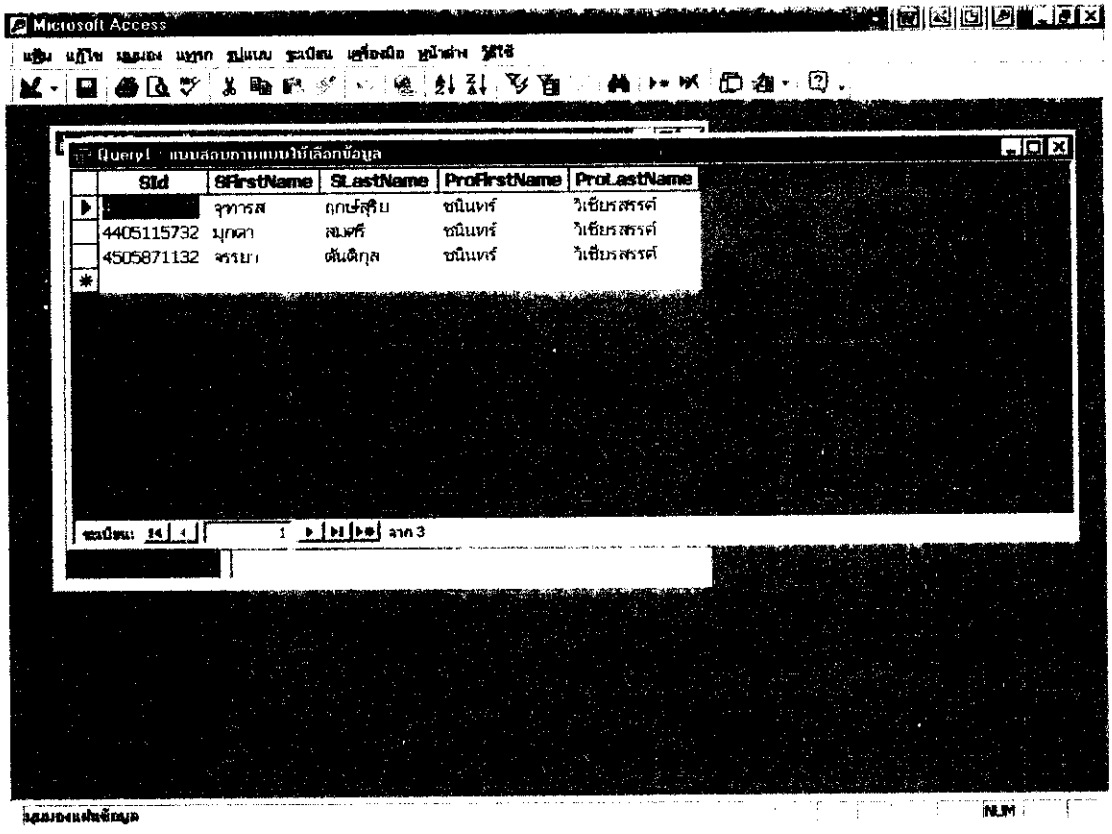
เช่นต้องการทราบว่านักศึกษาคนไหนมีอาจารย์ "ชนินทร์" เป็นอาจารย์ที่ปรึกษา ก็ต้องเลือกตาราง Student และตาราง Professor

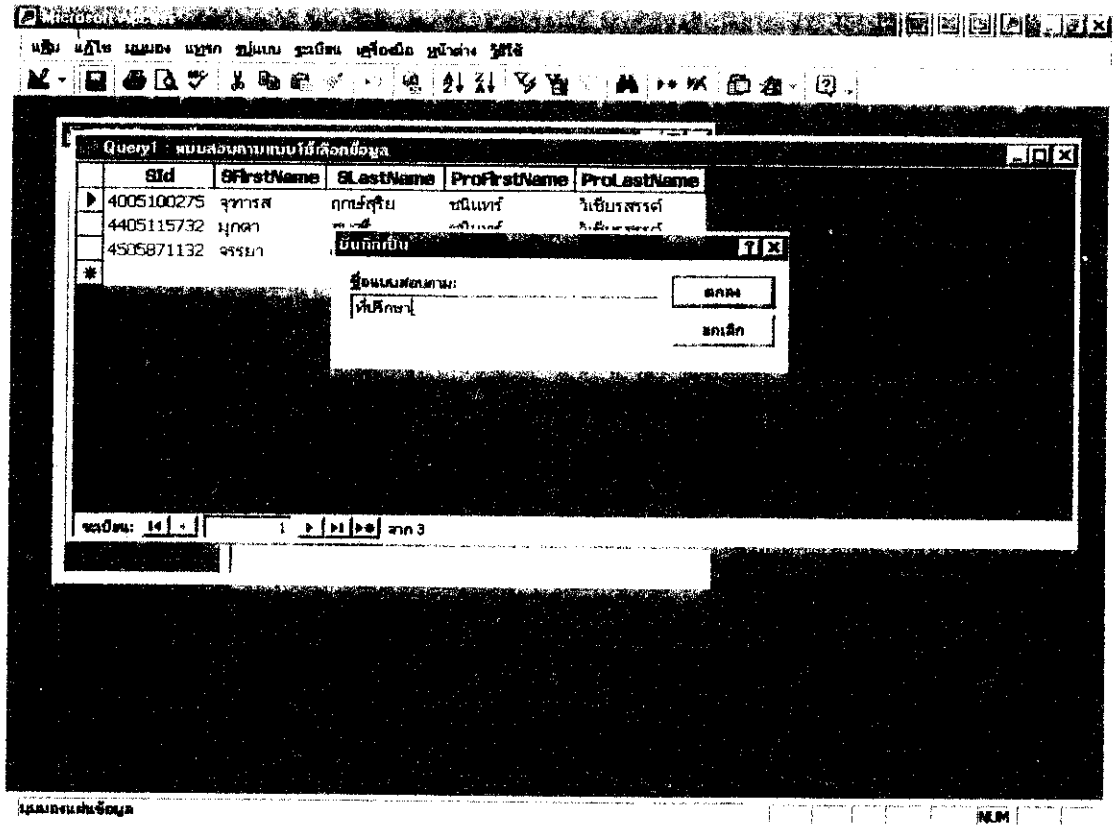


แล้วเลือกฟิลด์ที่ต้องการในที่นี้ จะเลือก SId, SFirstName, SLastName ของตาราง Student และเลือก ProFirstName, ProLastName ของตาราง Professor และใส่เงื่อนไข "ชนินทร์" ในช่องเงื่อนไขของฟิลด์ ProFirstName

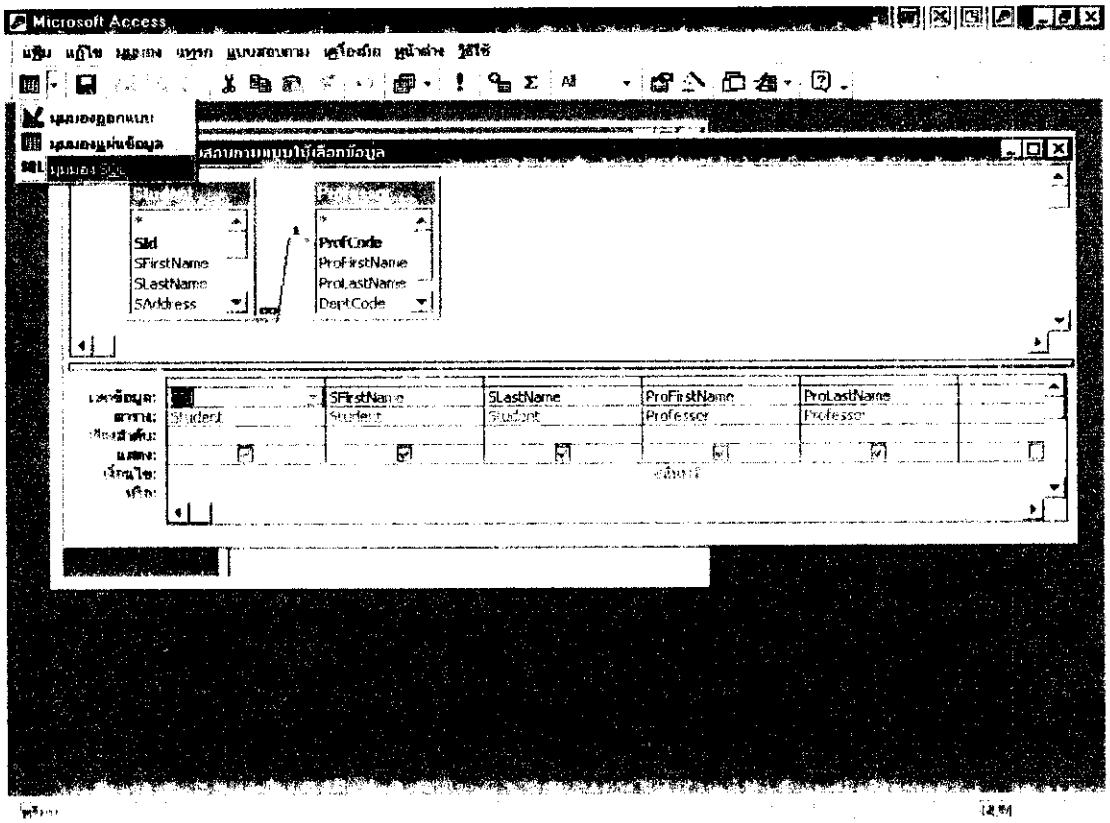


เมื่อกดปุ่มมุมมอง ก็จะได้ข้อมูลดังนี้

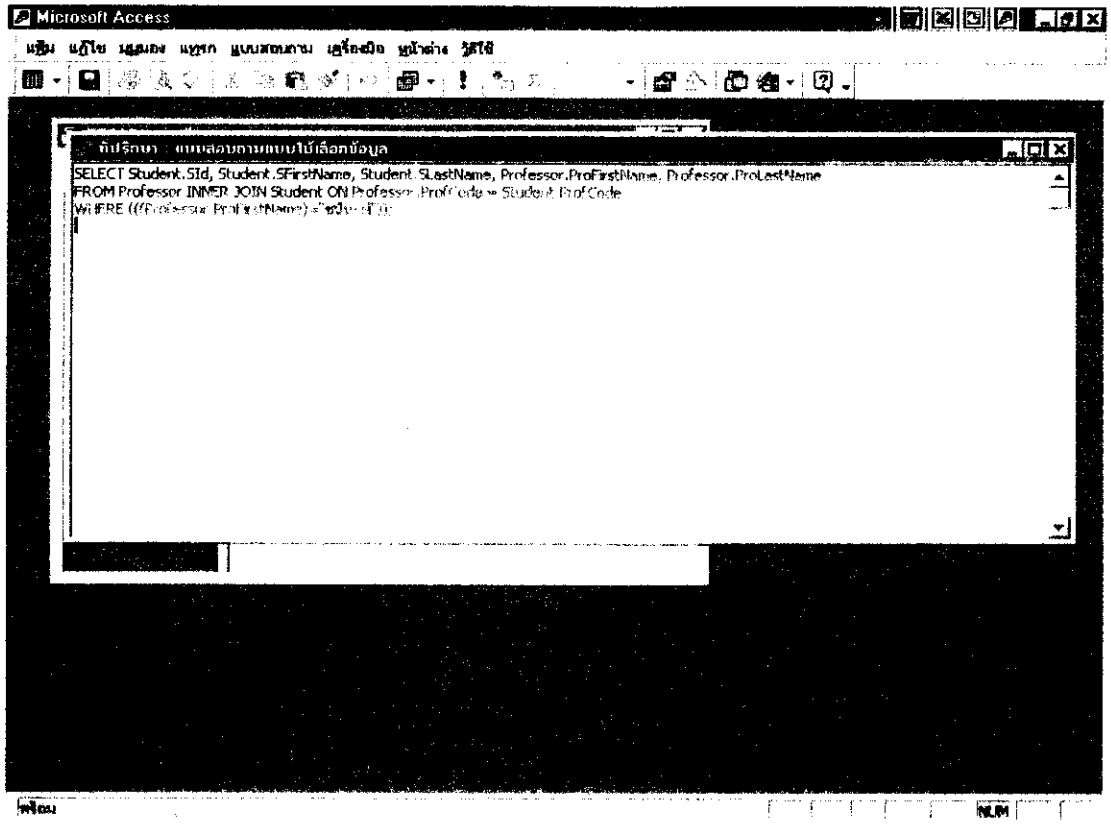



Query1 : แบบสอบถามแบบให้เลือกข้อมูล
SId
SFirstName
SLastName
ProFirstName
ProLastName
4005100275 จุฑารส กุกษ์สุริย ชนินทร์ ไชยรสรรค์
4405115732 มุกดา
4505871132 จรรยา
บันทึกเข้า
ตกลง
ยกเลิก
ระเบียน:
จาก 3

Query ใน Microsoft Access ก็คือการซ่อนภาษา SQL ถ้าต้องการให้ Microsoft Access แสดงภาษา SQL ก็เลือก SQL

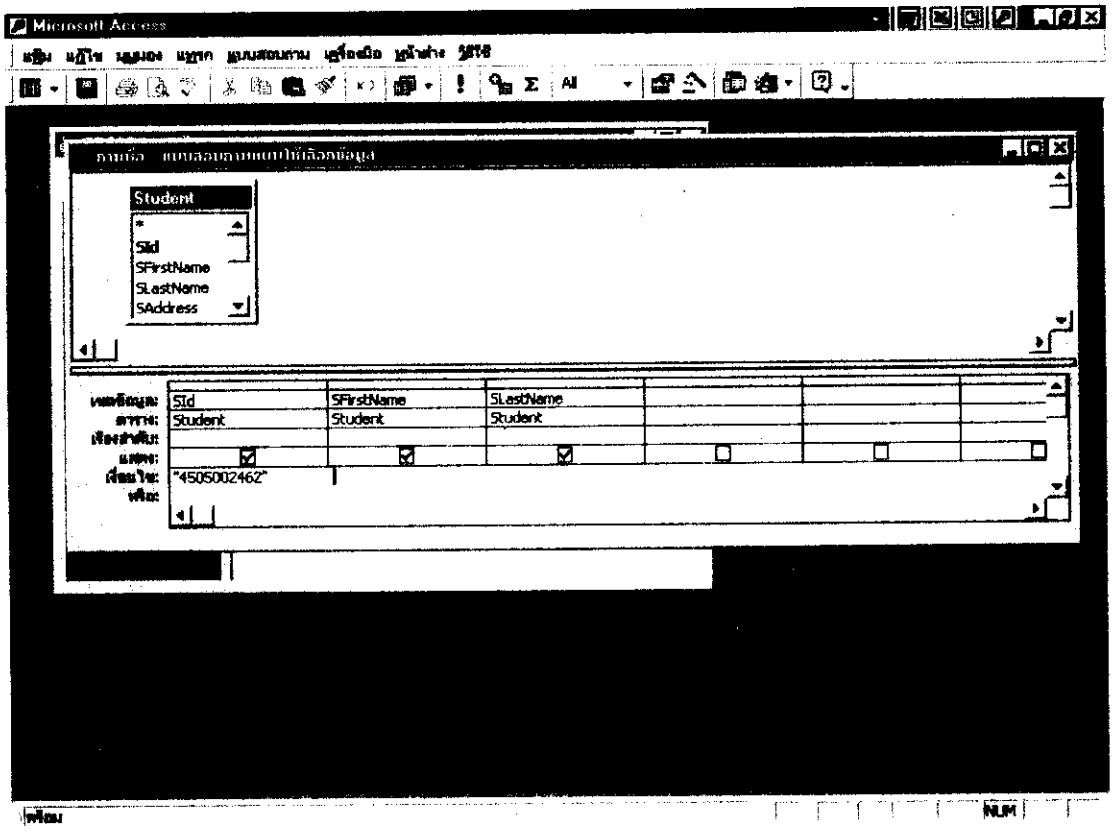


จะได้ดังรูป




```
SELECT Student.SId, Student.SFirstName, Student.SLastName, Professor.ProFirstName,
       Professor.ProLastName
FROM Professor INNER JOIN Student ON Professor.ProfCode = Student.ProfCode
WHERE (((Professor.ProFirstName) ="ชนินทร์"));
```

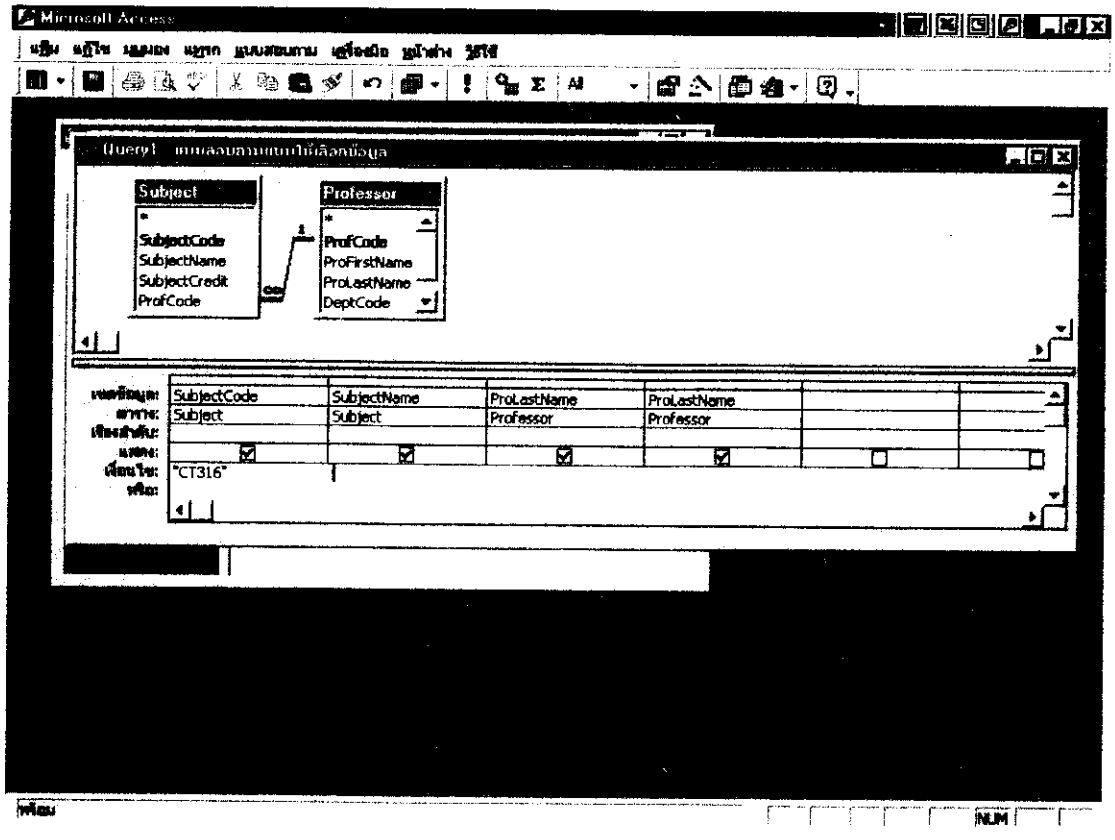
Query2 ให้หาว่านักศึกษารหัส "4505002462" คือใคร ซึ่งจะเรียกใช้เฉพาะตาราง Student




```
SELECT Student.SId, Student.SFirstName, Student.SLastName
FROM Student
WHERE (((Student.SId)="4505002462"));
```

SId	SFirstName	SLastName
4505002462	ธนจักร	ชำนาญ

Query3 วิชา "CT316" ใครสอน



SELECT Subject.SubjectCode, Subject.SubjectName, Professor.ProLastName, Professor.ProLastName

FROM Professor INNER JOIN Subject ON Professor.ProfCode = Subject.ProfCode

WHERE (((Subject.SubjectCode)="CT316"));

SubjectCode	SubjectName	Expr1002	ProLastName
CT316	Database Systems	วิเชียรสรรค์	วิเชียรสรรค์

คำศัพท์และความหมาย

abstract data types	แบบชนิดข้อมูลนามธรรม
abstract level	ระดับนามธรรม
ad hoc queries	คำถามตามต้องการ
Ad Hoc Query	คำสั่งที่จะใช้สั่งเรียกข้อมูลพิเศษที่ต้องการ
Adding Data to The Table	เพิ่มข้อมูลไปไว้ในตาราง
Addition Anomalies	ความผิดพลาดในการเพิ่มข้อมูล
Address	เลขที่อยู่
Advanced Data Management Commands	คำสั่งจัดการข้อมูลชั้นสูง
Aggregate functions	ฟังก์ชั่นทางคณิตศาสตร์
algorithm	ขั้นตอนวิธี
Anomaly	การผิดหลักการ
application programs	โปรแกรมการใช้งาน
Attribute Domain	ลักษณะประจำโดเมน
attribute value	ค่าของลักษณะประจำ
Attributes	ลักษณะประจำ
automatic high-level optimization	การทำให้เหมาะที่สุดชั้นสูงโดยอัตโนมัติ
Automatic Transmission Database	การส่งข้อมูลโดยอัตโนมัติ
backup	การทำสำรองข้อมูล
Backup and recovery management	การจัดการการทำสำรองและฟื้นสภาพ
Basic Data Management Command (กลุ่มของภาษา DML)	พื้นฐานคำสั่งจัดการข้อมูล
Basic Function	การทำงานพื้นฐาน
Basic Modeling Concepts	แนวคิดพื้นฐานในการสร้างแบบจำลอง

Batch	กลุ่ม
Binary	ฐานสอง
Binary Search	การค้นหาแบบทวิภาค
Binding of methods	วิธีการเชื่อมโยง
Candidate Key	ซูเปอร์คีย์ ที่ไม่มีการซ้ำ,คีย์ให้เลือก
Cardinality	คู่ลำดับ
Characteristics	ลักษณะกำหนด
Checking The Table Contents	ตรวจสอบเนื้อหาของตาราง
Child Segment	ส่วนของลูก
class object	กลุ่มวัตถุ
Classification of Operations	ประเภทการทำงานของวัตถุ
Collection hierarchies, type extents and queries	ลำดับชั้นของกลุ่ม, ขอบเขตของชนิดและคิวรี
Collection Types	แบบรวบรวมวัตถุ
Collision	การชนกัน
common variables	ตัวแปรร่วมกัน
Comparison of Object-Oriented and Object-Relational Databases	การเปรียบเทียบฐานข้อมูลเชิงวัตถุและฐานข้อมูลเชิงสัมพันธ์
Complex Data Relationship	ความสัมพันธ์ของข้อมูลที่ซับซ้อนขึ้น
Complex Object	วัตถุเชิงซ้อน
Complex Types and Object Orientation	ฐานข้อมูลเชิงวัตถุประเภทซับซ้อน
complex values constructed	โครงสร้างค่าซับซ้อน
Composite Entity	การประกอบเป็นเอนทิตี้
Composite Key	ลักษณะประจำ หลายๆ ตัวที่ร่วมกันทำหน้าที่เป็น คีย์
Conceptual Model	แบบจำลองของแนวคิด

Conceptual Level	ระดับของแนวคิด
Conceptual models	รูปแบบแนวคิด
configuration	องค์ประกอบ
Connectivity	ความสัมพันธ์ระหว่างเอนทิตี้
Constructed types	ชนิดที่ถูกสร้าง
Constructed types and ADTs	ชนิดและข้อมูลเชิงนามธรรม
constructors	ตัวสร้าง
Conversion To First Normal Form	การแปลงไปสู่รูปแบบบรรทัดฐานแรก
Conversion to Second Normal Form	การแปลงไปสู่รูปแบบบรรทัดฐานที่สอง
Conversion to Third Normal Form	การแปลงไปสู่รูปแบบบรรทัดฐานที่สาม
Create View	สร้างวิว
data access	การเข้าถึงข้อมูล
Data Anomalies	ที่ข้อมูลเดียวกันแต่อยู่ต่างแฟ้มกันกลับมีค่าของข้อมูลไม่เหมือนกัน
Data Anormalies	ข้อมูลไม่ปกติ
Data Definition Commands (DDC)	
data definition language	กำหนดโครงสร้างข้อมูลที่จะจัดเก็บ
Data Definition Language (DDL)	คำสั่งการนิยามข้อมูล
Data Dependency	ข้อมูลที่น่าเชื่อถือ
Data Dictionary	เก็บรายลเอียดเกี่ยวกับ Tables ทั้งหมดที่มีอยู่ใน Database
Data Dictionary (DD)	จัดเก็บข้อมูลรวมทั้งความสัมพันธ์เกี่ยวกับข้อมูลทั้งหมด
Data Dictionary and System Catalog	พจนานุกรมข้อมูลและระบบรายระเอียด
Data dictionary management	การจัดการพจนานุกรมข้อมูล

Data Inconsistencies	ไม่สอดคล้องกันของข้อมูล
data inconsistency	ความขัดแย้งกับข้อมูล
Data Inconsistency	ความไม่มั่นคงของข้อมูล
Data integrity management	การจัดการบูรภาพของข้อมูล
Data Management	การจัดการข้อมูล
data management	การจัดการข้อมูล
Data Management Language (DML)	คำสั่งที่ใช้ในการกระทำกับข้อมูล
data manipulation language	กำหนดวิธีการนำข้อมูลเข้าไปจัดเก็บและนำออกมาใช้
Data Model	การแสดงอย่างง่ายและชัดเจนของโครงสร้างข้อมูลจริงที่ ซับซ้อน (โดยการใช้ Graphic ในการอธิบาย)
data operation	การปฏิบัติการข้อมูล
Data Redundancy	การเกิดการซ้ำกันของข้อมูล
Data storage management	การจัดการการจัดเก็บข้อมูล
Data transformation and presentation	การนำเสนอข้อมูลและการแปลงข้อมูล
Database Administrator:DBA	ผู้บริหารฐานข้อมูล
Database communication interfaces	การติดต่อฐานข้อมูล
Database design	การออกแบบฐานข้อมูล
Database design for an ORDBMS	การออกแบบฐานข้อมูลสำหรับ ORDBMS
database internal structure	โครงสร้างฐานข้อมูลภายใน
Database Language	ภาษาฐานข้อมูล
Database Model	รูปแบบฐานข้อมูล
Database System Environment	สภาพแวดล้อมของระบบฐานข้อมูล
Database Tables and Normalization	ฐานข้อมูล ตารางและการทำให้เป็นบรรทัดฐาน
Data-Sharing	การใช้ข้อมูลร่วมกัน

Defining methods of an ADT	การกำหนดการทำงานสำหรับข้อมูลนามธรรม
Defining types with inheritance	การกำหนดชนิดด้วยการสืบทอด
Degrees of Data Abstraction	ระดับของข้อมูลเชิงนามธรรม
Delete Table	ลบตาราง
Delete Table Rows	ลบแถวในตาราง
Deletion Anomalies	ความผิดพลาดในการลบข้อมูล
Denormalization	ไม่เป็นบรรทัดฐาน
Dependency Diagram	แผนผังความสัมพันธ์
Derived Attribute	ลักษณะประจำที่หาค่าได้โดยใช้ขั้นตอนวิธีช่วยจึงไม่จำเป็น ต้องเก็บไว้ในฐานข้อมูล
Derived attributes	ลักษณะประจำสืบทอด
Determinant	ลักษณะประจำใดๆ ที่มีค่าสามารถระบุไปถึงข้อมูลต่างๆ ในแถวข้อมูล
Determination	การตัดสินในได้
Difference	แสดงทุก Row ที่มีใน Table ที่ 1 แต่ไม่มีใน Table ที่ 2 และจะต้องเป็น Union Compatible
Digital data	ข้อมูลดิจิตอล
Direct Access	เข้าถึงโดยตรง
Distinct- ไม่มี Unique	แตกต่าง,ไม่ซ้ำ
document-retrieval system	ระบบค้นคืนเอกสาร
Domain	ลักษณะของโครงสร้างของข้อมูลที่กำหนดให้กับลักษณะประจำ หรือ ก็คือขอบเขตนั่นเอง
embedded query language	
embedded SQL	การใส่ SQL ในภาษาโปรแกรม
Encapsulation	การซ่อนสารสนเทศ

end user data	ข้อมูลดิบที่ผู้ใช้สนใจ
Entity	เอนทิตี้
Entity Supertypes and Subtypes	การแบ่งชั้นของเอนทิตี้
Entity Integrity	ความถูกต้องมั่นคงในเรื่องของ เอนทิตี้
Entity Relation Modeling	การสร้างแบบจำลองความสัมพันธ์ของเอนทิตี้
EntitySet	กลุ่มของเอนทิตี้
E-R Model	แบบจำลองของ E-R
Existence Dependency	การขึ้นอยู่กับ
expression	นิพจน์
extended SQL syntax	ไวยากร SQL เพิ่มเติม
Extending the ER model	ส่วนขยายของตัวแบบ ER
Extent	ขอบเขต
External Model	แบบจำลองภายนอก
external interfaces	การติดต่อจากภายนอก
File Systems Data Management	การจัดการข้อมูลในระบบแฟ้มข้อมูล
First Normal Form (1NF)	รูปแบบบรรทัดฐานที่หนึ่ง
Flexibility	ปรับเปลี่ยน
Foreign Key	ฟอร์เรนคีย์
Fourth Normal Form (4NF)	รูปบรรทัดฐานที่สี่
Fully Relational	ความสัมพันธ์เต็มรูปแบบ
Function	ฟังก์ชั่น
Hashing Function	ฟังก์ชั่นแบบแฮช
hidden attribute	ลักษณะที่ถูกซ่อน
Hierarchical Database Model	รูปแบบฐานข้อมูลลำดับชั้น
Hierarchical Path	เส้นทางลำดับชั้น

Hierarchical Structure	โครงสร้างลำดับชั้น
Higher-Level Normal Forms	บรรทัดฐานระดับสูง
High-Level Language	ภาษาโปรแกรมชั้นสูง
host language	ภาษาที่ใช้เป็นพื้นฐาน
immutable	เปลี่ยนรูปไม่ได้
immutable	เปลี่ยนรูปไม่ได้
imperative languages	ภาษาเชิงคำสั่ง
implementation details	รายละเอียดในการทำให้เกิดผล
Implementation models	รูปแบบนำไปใช้
Index	ดัชนี
Indexed – Sequential File	แฟ้มลำดับดัชนี
Indexing new types	การทำดรรชนีแบบชนิดใหม่
Information	สารสนเทศ
Information Manage System	การจัดการระบบสารสนเทศ
Information Technology	เทคโนโลยีสารสนเทศ
Inheritance	การสืบทอด
instance	ตัวอย่าง
Integrity and Consistency	ความ สอดคล้องกันของข้อมูล
Integrity Rules	กฎความมั่นคง
Interactive	เชิงโต้ตอบ
interface	การติดต่อ
Internal Model	แบบจำลองภายใน
Internal Structure	โครงสร้างภายใน
Interprogram	กระบวนคำสั่งภายนอก
Intersect	ใช้ในการแสดงเฉพาะ Row ที่มีในทั้ง 2 Tables

CT 316 (S)

Intraprocedure	กระบวนคำสั่งภายใน
invoke a method	อ้างถึงระเบียบ
Join	ใช้รวมข้อมูลของ 2 Tables เข้าด้วยกัน
Joining The Table	ใช้ตารางร่วมกัน
key	กุญแจหลัก
Key Attribute	ลักษณะประจำของกุญแจ
Key Attribute	ลักษณะประจำ ใดๆ ที่ทำหน้าที่เป็น คีย์
Keys	สิ่งที่ช่วยในการระบุ เอนทิตี้ และ ความสัมพันธ์ของเอนทิตี้
logical data	ข้อมูลที่ผู้ใช้ ใช้ในการทำงานจริง
logical view	มุมมองทางตรรก
Magnetic Disk	จานแม่เหล็ก
Maintain	การปรับเปลี่ยน
Making a Correction	ทำการปรับปรุง
Manipulating data of constructed types	การจัดการกับข้อมูลของชนิดที่ถูกสร้าง
Many-To-Many	หลาย-ต่อ-หลาย
messages	ข่าวสาร
metadata	ข้อมูลแสดงรายละเอียดเกี่ยวกับคุณสมบัติของข้อมูล
Minimal Relational	ความสัมพันธ์น้อยที่สุด
models	นามธรรมอย่างง่ายของเหตุการณ์หรือเงื่อนไขที่มีอยู่ในโลกของความเป็นจริง
models	รายละเอียดที่ใช้ในการอธิบายให้เห็นภาพพจน์ของสิ่งที่ไม่สามารถจะสังเกตุเห็นได้โดยตรง
More Complex Queries	คำถามที่ซับซ้อน
Multi valued Dependencies	ความสัมพันธ์หลายค่า

Multi valued Dependencies	ความน่าเชื่อถือของข้อมูลหลายๆ ค่า
Multiple Inheritance and Selective Inheritance	การสืบทอดแบบหลายส่วนและแบบเลือก
multi-user	หลายผู้ใช้
Multi-user access control	การเข้าถึงข้อมูลโดยผู้ใช้หลายๆ คน
Mutators	การเปลี่ยนรูป
Naive User	ผู้ใช้ที่ไม่มีประสบการณ์
Nested Relations	ความสัมพันธ์ซ้อนกัน
Nesting and Unnesting	การซ้อนกันและไม่ซ้อนกัน
Network Database Model	รูปแบบฐานข้อมูลเครือข่าย
Network Schema	เค้าร่างเครือข่าย
New challenges in implementingt an ORDBMS	แนวทางในการสร้าง ORDBMS
Nonkey Attribute	ลักษณะประจำที่ไม่เป็นกุญแจ
Nonprime Attribute	ลักษณะประจำรอง
non-procedural language	ระบุว่าสิ่งที่ต้องการ :what แต่ไม่ต้องระบุวิธีที่ต้องทำ :how
Non-Procedural Language	เป็นภาษาที่ผู้ใช้เพียงแค่ระบุถึงสิ่งที่ต้องการ แต่ไม่ต้องระบุวิธีการทำงาน
Normal Inheritance	การสืบทอดแบบปกติ
Normalization	การทำให้เป็นบรรทัดฐาน
Normalization and Database Design	การออกแบบฐานข้อมูลและรูปแบบบรรทัดฐาน
Notions of equality	แนวคิดของความเสมอภาค
Object Classes	คลาสของวัตถุ
Object Database Management Group (ODMG)	กลุ่มการจัดการฐานข้อมูลวัตถุ

object identifier	สิ่งระบุวัตถุ
Object Identity	สิ่งระบุวัตถุหรือเอกสักษณ์ของวัตถุ
Object identity	สิ่งระบุวัตถุ
Object Identity and Pointers	สิ่งระบุวัตถุและตัวชี้
Object Identity form	รูปแบบต่างๆ ของสิ่งระบุวัตถุ
object instance	ตัวอย่างวัตถุ
Object Structure	โครงสร้างของวัตถุ
Object Structure	การซ่อนสารสนเทศ
object type	ประเภทของวัตถุ
Object, object identity, and reference types	วัตถุ, สิ่งระบุวัตถุและแบบชนิดอ้างอิง
Object-Oriented context	เนื้อหาเชิงวัตถุ
object-oriented database	ฐานข้อมูลเชิงวัตถุ
object-oriented extensions	ส่วนขยายเชิงวัตถุ
object-oriented features	ลักษณะเชิงวัตถุ
object-oriented format	รูปแบบเชิงวัตถุ
Object-Oriented Languages	ภาษาเชิงวัตถุ
object-oriented model	แบบจำลองเชิงวัตถุ
object-oriented programming language	ภาษาโปรแกรมเชิงวัตถุ
object-oriented systems	ระบบเชิงวัตถุ
Object-relation systems	ระบบวัตถุเชิงสัมพันธ์
Object-relational data models	รูปแบบข้อมูลเชิงสัมพันธ์วัตถุ
Object-Relational Databases	ฐานข้อวัตถูมูลเชิงสัมพันธ์
Observer functions	การทำงานตรวจพิจารณา
office information systems	ระบบสารสนเทศสำนักงาน
On Delete Restrict	ลบข้อจำกัด

On Update Cascade	ปรับปรุงแบบเรียงต่อ
One-To-Many	หนึ่ง-ต่อ-หลาย
One-To-One	หนึ่ง-ต่อ-หนึ่ง
operations	ตัวดำเนินการ
Other Object-Oriented Concepts	แนวคิดเชิงวัตถุอื่นๆ
overloading	การโหลดเกิน
Parent	พ่อ
Parent Segment	ส่วนของพ่อ
Parent/Child Relationship	ความสัมพันธ์ พ่อ- ลูก
Partial Dependency	ความสัมพันธ์บางส่วน
Path Expression	เส้นทางของนิพจน์
Persistence by class	คลาสทนทาน
Persistence by creation	การสร้างที่คงที่ทนทาน
Persistence of Objects and Semantic Reference	ความทนทานของวัตถุและการอ้างอิงอรรถศาสตร์
Persistency by making	การกำหนดวัตถุแบบคงที่
Persistency by reference	การอ้างอิงที่ต่อเนื่องซ้ำ
Persistent	การต่อเนื่องซ้ำๆ
persistent object	วัตถุทนทาน
persistent programming languages	
Physical Details	รายละเอียดเชิงกายภาพ
Physical Level	ระดับกายภาพ
Pointer	ตัวชี้
Polymorphism or Operator Overloading	การมีหลายรูปแบบ
Portability	สามารถเคลื่อนย้ายได้

Primary and Foreign Key Designation	การกำหนดกุญแจหลักและกุญแจภายนอกใหม่
Primary Key	ซูเปอร์คีย์ และเป็นแคนดิเดทคีย์
Prime Attribute	ลักษณะประจำหลัก
Primitive Constructor	ตัวสร้างชนิดเบื้องต้น
Problem Oriented Language	ภาษาที่เชี่ยวชาญในปัญหาด้านใดด้านหนึ่ง
Procedural Oriented Language	ภาษาที่เชี่ยวชาญในการทำงาน
Product	แสดงทุกความเป็นไปได้ของ Row ที่จะจับคู่กันใน Table ทั้ง 2 Table
Program Dependency	โปรแกรมที่น่าเชื่อถือ
Programmer	นักเขียนชุดคำสั่ง
Project	แสดงข้อมูลของ ลักษณะประจำ ที่เราสนใจเท่านั้น (แสดงข้อมูลในลักษณะของ Column)
pseudocode	รหัสเทียม
Queries	คำถาม
Query Language	ภาษาสอบถาม
Query processing	การประมวลผลคิวรี
Querying with Complex Types	ประเภทของคำถามที่ซับซ้อน
Random File	แฟ้มสุ่ม
Random File (Direct File)	แฟ้มข้อมูลเชิงสุ่ม
read-only method	แบบที่อ่านได้อย่างเดียว
Record Key	กุญแจหลักของระเบียน
recovery	การฟื้นสภาพ
Recursive Entity	เอนทิตี้ที่ซึ่งมีความสัมพันธ์ได้กับเอนทิตี้ชนิดเดียวกัน โดยปกติจะพบในความสัมพันธ์เดี่ยวๆ
reference semantic	การอ้างถึงความหมาย

Reference Type	ประเภทของการอ้างอิง
references to objects	การอ้างถึงวัตถุ
Referential Integrity	ความมั่นคงในเรื่องของการอ้างอิงผ่านฟอร์เรนคีย์
Referential Integrity	ความคงสภาพของข้อมูล
Relational Database Model	แบบจำลองฐานข้อมูลเชิงสัมพันธ์
Relational Database Model	แบบจำลองฐานข้อมูลเชิงสัมพันธ์
Relational Database Operators	ตัวดำเนินการกับฐานข้อมูลเชิงสัมพันธ์
relational format	รูปแบบความสัมพันธ์
Relational systems	ระบบความสัมพันธ์
Relationally Complete	ความสัมพันธ์ที่สมบูรณ์
Relationship Participation	ความสัมพันธ์ที่มีส่วนร่วม
Relationships Within the Relational Database	ความสัมพันธ์ในฐานข้อมูลเชิงสัมพันธ์
Relation-Valued Attributes	ความสัมพันธ์ของค่าของลักษณะประจำ
Repeating Groups	กลุ่มข้อมูลที่ซ้ำ
request	ความต้องการ
Restore The Table Contents	คืนสู่เนื้อหาปกติของตาราง
root persistent object	วัตถุคงที่
routine	งานที่ทำประจำ
Saving The Table Contents	จัดเก็บเนื้อหาของตาราง
Schema	เค้าร่าง
Scheme	รูปแบบที่อธิบายด้วย Text
Second Normal Form (2NF)	รูปแบบบรรทัดฐานที่สอง
Secondary Key	คีย์ ที่กำหนดขึ้นมาใช้ในการทำการค้นคืนข้อมูลเท่านั้น

Security	ความปลอดภัย
Security management	การจัดการด้านความปลอดภัย
Segment	ส่วน
Select	แสดงข้อมูลทุก ลักษณะประจำ ที่มีใน Row ที่เราสนใจ (แสดงข้อมูลในลักษณะของ Row
Sematic Reference	การอ้างอิงความหมาย
Sequential Access	เข้าถึงโดยลำดับ
Sequential Search	การค้นหาตามลำดับ
Single Data Repository	การนำเอาข้อมูลทั้งหมดมาไว้ที่เดียวกัน
Single Entity	เอนทิตี้เดี่ยว
Single unit	หน่วยเดี่ยวๆ
special	พิเศษ
Special Operators	ตัวดำเนินการพิเศษ
specialization hierarchy	ลำดับชั้นพิเศษ
SQL Numeric Functions	ฟังก์ชั่นเกี่ยวกับการคำนวณในภาษาสอบถามเชิงโครงสร้าง
Standard Concepts	แนวคิดแบบมาตรฐาน
Standard Transmission Database	มาตรฐานในการส่งข้อมูล
state	สถานะ
Storage and access methods	หน่วยจัดเก็บและวิธีการเข้าถึง
Structural Dependency	โครงสร้างที่น่าเชื่อถือ
Structure Query Language (SQL)	ภาษาสอบถามเชิงโครงสร้าง
Structured and Collection Types	ประเภทของโครงสร้างและการจัดกลุ่ม
Structured Complex Objects	วัตถุเชิงซ้อนแบบมีโครงสร้าง
subclass	คลาสย่อย
subexpression	นิพจน์ย่อย

subobjects	วัตถุย่อย
Subschema	เค้าร่างย่อย
superclass	คลาสเสริม
Superkey	คีย์ ที่แสดง เอนทิตี้โดยไม่มีการซ้ำกัน
system architecture	สถาปัตยกรรมระบบ
System Catalog	เก็บ Metadata เช่นเดียวกับ Data Dictionary แต่เปรียบได้เป็น Data Dictionary ที่เก็บข้อมูลที่ละเอียดมาก คือ เก็บรายละเอียดเกี่ยวกับ Object ทุกอย่างที่มีใน Database
Systems Analysis	การวิเคราะห์ระบบ
Systems Development	การพัฒนาระบบ
Table	ที่เก็บข้อมูล บางครั้งเรียกว่า Relation
Tabular	จัดระเบียบ
Terminal	เทอร์มินัล
The BOYCE-CODD normal Form (BCNF)	บรรทัดฐานแบบ BOYCE-CODD
The Need For Normalization	ความต้องการการทำให้เป็นบรรทัดฐาน
The Object-Oriented Data Model	แบบจำลองข้อมูลเชิงวัตถุ
transaction	การเปลี่ยนแปลงข้อมูล
Transitive Dependency	ความสัมพันธ์ต่อผ่าน
Tuple	ที่อยู่ของสมาชิกใน Set (Row)
type constructors	การใช้ตัวสร้างชนิด
type system	ประเภทระบบ
UNICODE	มาตรฐานรหัสครอบจักรวาล
Union	แสดงรวมทุกแถว
unique identity	การกำหนดเอกลักษณ์พิเศษ

CT 316 (S)