แผนการสอนประจำบทเรียน

รายชื่ออาจารย์ผู้จัดทำ อาจารย์วิทวัส พันธุมจินดา

รายละเอียดของเนื้อหา

ตอนที่ 14.1 แนวคิดฐานข้อมูลเชิงวัตถุ

เรื่องที่ 14.1.1 โมเดลข้อมูลเชิงวัตถุ

เรื่องที่ 14.1.2 ภาษาเชิงวัตถุ

ตอนที่ 14.2 แนวคิดฐานข้อมูลเชิงวัตถุ-สัมพันธ์

เรื่องที่ 14.2.1 ความสัมพันธ์เชิงกลุ่ม

เรื่องที่ 14.2.2 ชนิดข้อมูลที่ซับซ้อนและแนวทางเชิงวัตถุ

เรื่องที่ 14.2.3 การสืบค้นข้อมูลชนิดที่ซับซ้อน

เรื่องที่ 14.2.4 การสร้างค่าข้อมูลที่ซับซ้อนและการสร้างวัตถุ

แนวคิด

1. ฐานข้อมูลเชิงวัตถุได้ถูกออกแบบมาเพื่อตอบสนองต่อความต้องการใหม่ ๆ ของโปรแกรมประยุกต์ในปัจจุบัน ซึ่งโมเดลข้อมูลแบบเชิงสัมพันธ์ไม่สามารถตอบสอนต่อความต้องการเหล่านั้นได้ และในการพัฒนาฐานข้อมูลเชิงวัตถุก็จะเกี่ยวข้องกับโมเดลข้อมูลเชิงวัตถุ และการโปรแกรมเชิงวัตถุ
2. ฐานข้อมูลเชิงวัตถุ-สัมพันธ์เป็นโมเดลของข้อมูลที่เพิ่มคุณสมบัติเชิงวัตถุ ให้กับโมเดลเชิงสัมพันธ์ เพื่อให้สนับสนุนชนิดข้อมูลได้หลากหลายขึ้น รวมไปถึงการเพิ่มแนวคิดเชิงวัตถุเข้าไปในภาษาสืบค้นข้อมูลด้วย

วัตถุประสงค์

1. ให้ผู้เรียนเข้าใจลักษณะของฐานข้อมูลเชิงวัตถุ ทั้งโมเดลข้อมูลเชิงวัตถุ หลักการเชิงวัตถุ และภาษาโปรแกรมเชิงวัตถุ
2. ให้ผู้เรียนเข้าใจในลักษณะความสัมพันธ์เชิงกลุ่ม(Nested Relations) ชนิดข้อมูลที่ซับซ้อน การจัดการข้อมูลที่ซับซ้อน และการสืบค้นข้อมูลจากความสัมพันธ์เชิงกลุ่ม โดยวิธีการเชิงวัตถุ

กิจกรรมการเรียนการสอน

1. ศึกษาเอกสารการสอน
2. ปฏิบัติกิจกรรมตามที่ได้รับมอบหมายในเอกสารการสอนแต่ละตอน

สื่อการสอน

1. เอกสารการสอนของชุดวิชา

2. แบบฝึกปฏิบัติ
3. บทความ/ข้อมูลทางคอมพิวเตอร์
4. การให้คำปรึกษาทางโทรศัพท์
5. CD-ROM
6. Homepage ของชุดวิชาผ่านทางอินเตอร์เน็ต

เอกสารประกอบการสอน

1. Abraham Silberschatz , Henry F. Korth , S. Sudarshan Database System Concepts, Third Edition, Mc Graw Hill, 1997.
2. Raghu Ramakrisanan, Johannes Gehrke Database Management Systems, Second Edition, Mc Graw Hill, 2000
3. Elemasri, Ramez and Navathe, Sinamkant B. Fundamentals of Database Systems, California: The Benjamin/Cummings Pub. Co., 1989.

ประเมินผล

ประเมินผลจากกิจกรรมที่ทำ

ประเมินผลจากคำถามท้ายบทเรียนและการสอนประจำภาคการศึกษา

ตอนที่ 14.1 แนวคิดฐานข้อมูลเชิงวัตถุ

หัวเรื่อง

เรื่องที่ 14.1.1 โมเดลข้อมูลเชิงวัตถุ

เรื่องที่ 14.1.2 ภาษาเชิงวัตถุ

เรื่องที่ 14.1.1 โมเดลข้อมูลเชิงวัตถุ

โมเดลของข้อมูลเป็นการแสดงให้เห็นองค์ประกอบสิ่งต่าง ๆ (objects)ที่เราสนใจ ซึ่งจะมีข้อกำหนดที่กำหนดให้กับองค์ประกอบเหล่านั้น และแสดงความสัมพันธ์ระหว่างองค์ประกอบเหล่านั้น

หลักการของโมเดลข้อมูลเชิงวัตถุจะประกอบไปด้วยองค์ประกอบเหล่านี้คือ

object and object identifier: หมายถึงอ๊อปเจ็กใด ๆ ที่อยู่ในรูปแบบที่กำหนด และมีการกำหนดค่า id ที่ไม่ซ้ำกันเพื่อใช้ในการอ้างถึงอ๊อปเจ็กนั้น

attributes and methods: อ๊อปเจ็กทุก ๆ อ๊อปเจ็กจะมีคุณลักษณะ(state) คือเซตของค่าข้อมูลของ แอตทริบิวต์ของอ๊อปเจ็ก และการกระทำ(behavior) เป็นเซตของวิธีการ ส่วนของโปรแกรม ซึ่งมีการดำเนินการบนลักษณะของอ๊อปเจ็ก ซึ่งลักษณะและการกระทำจะถูกห่อหุ้มไว้ในอ๊อปเจ็ก และจะถูกอ้างถึงหรือถูกเรียกให้ทำงานจากภายนอกผ่านทางข่าวสาร(message)

class: หมายถึงการรวมกลุ่มของอ๊อปเจ็กทั้งหมดที่มีลักษณะและวิธีการที่เหมือนๆ กัน อ๊อปเจ็กหนึ่งๆ จะถูกสร้างขึ้นจากคลาสเพียงหนึ่งคลาส ซึ่งคลาสมีลักษณะเหมือนกับ abstract data type คลาสอาจจะเป็นข้อมูลพื้นฐานก็ได้เช่น integer string หรือ boolean เป็นต้น

Class hierarchy and inheritance: เป็นการสร้างคลาสใหม่(subclass)จากคลาสเดิม(superclass)ที่มีอยู่ก่อน คลาสที่สร้างขึ้นใหม่จะสืบทอดคุณลักษณะและวิธีการทั้งหมดจากคลาสเดิม โดยคลาสใหม่สามารถที่จะเพิ่มคุณลักษณะ และวิธีการใหม่ เพิ่มเข้าไปได้ จะแบ่งเป็นสองประเภทคือ single inheritance และ multiple inheritance

1. Object Structure

แนวทางเชิงวัตถุมีพื้นฐานมาจากการห้อหุ้มข้อมูล และโปรแกรม(encapsulating) ดังนั้นการติดต่อสื่อสารกันระหว่างอ๊อปเจ็กจะทำผ่านทางข่าวสาร(messages)เสมอ

โดยทั่วไปแล้ว อ๊อปเจ็กจะเกี่ยวข้องกับ

เซตของตัวแปรที่บรรจุข้อมูลของอ๊อปเจ็ก

เซตของข่าวสารซึ่งอ๊อปเจ็กจะตอบสนองต่อข่าวสารที่ได้รับจากภายนอก

เซตของวิธีการ ซึ่งแต่ละวิธีการจะเป็นโปรแกรมที่ดำเนินการกับข่าวสาร และส่งข้อมูลกลับ

สิ่งที่ทำให้ต้องมีการใช้ข่าวสารและวิธีการระหว่างอ๊อปเจ็ก พิจารณาเอนตินี้ employee ในระบบธนาคาร ทุก ๆ คนจะมีการคำนวณเงินรายได้ทั้งปี แต่มีความแตกต่างกันในเรื่องของการคำนวณ ตามประเภทของพนักงาน เช่น ผู้จัดการ ได้รับโบนัสตามกำไรที่ธนาคารทำได้ หรือพนักงานเคาเตอร์ จะได้รับโบนัสตามชั่วโมงการทำงาน เป็นต้น เราสามารถซ่อนโปรแกรมในส่วนของการคำนวณเงินเดือน โดยอ๊อปเจ็กจะรู้ว่าต้องคำนวณเงินรายได้ทั้งปีอย่างไร โดยที่อ๊อปเจ็กของพนักงานทุกคนมี interface เดียวกัน ในระบบเชิงวัตถุ เราสามารถที่จะแก้ไขนิยามของวิธีการ ตัวแปร โดยไม่มีผลกระทบต่อระบบ ซึ่งคุณสมบัตินี้เป็นข้อดีของการเขียนโปรแกรมเชิงวัตถุ

วิธีการของอ๊อปเจ็กสามารถจำแนกได้ 2 กลุ่มคือ ประเภทอ่านอย่างเดียว หรือ ประเภท Update ถ้าเป็นวิธีการแบบอ่านอย่างเดียวจะไม่มีผลกระทบต่อข้อมูลในอ๊อปเจ็ก ในขณะที่ Update จะทำการเปลี่ยนแปลงข้อมูลภายในอ๊อปเจ็กได้ ซึ่งข่าวสารที่อ๊อปเจ็กจะตอบสนองว่าเป็นแบบอ่านอย่างเดียวหรือแบบ update นั้นขึ้นอยู่กับวิธีการ implement ข่าวสาร

สำหรับแอตทริบิวต์ที่ได้จากการคำนวณนั้น ในระบบเชิงวัตถุจะใช้ข่าวสารแบบอ่านอย่างเดียว กล่าวคือทุก ๆ แอตทริบิวต์ของเอนติตี้ต้องประกอบไปด้วยข่าวสาร 2 อันคือ ข่าวสารสำหรับการอ่านข้อมูลของแอตทริบิวต์ และอีกข่าวสารสำหรับการปรับปรุงข้อมูล

2. Object Classes

โดยปกติ ในฐานข้อมูลจะมีอ๊อปเจ็กที่คล้าย ๆ กัน หมายความว่าอ๊อปเจ็กมีการตอบสนองต่อข่าวสารเดียวกัน ใช้วิธีการเดียวกัน และมีตัวแปรชื่อเดียวกัน ชนิดข้อมูลเดียวกัน ดังนั้นเราจึงจัดกลุ่มของอ๊อปเจ็กที่ลักษณะคล้าย ๆ กัน แล้วกำหนดเป็นคลาส ๆ หนึ่ง และแต่ละอ๊อปเจ็กที่สร้างมาจากคลาสก็เรียกว่า เป็น instance ของคลาส อ๊อปเจ็กทุกตัวจะมีลักษณะเหมือนกัน จะแตกต่างกันตรงข้อมูลที่ถูกกำหนดให้กับตัวแปรในอ๊อปเจ็ก

เราจะนิยามคลาส employee ในรูปของคำสั่งลำลอง ซึ่งแสดงตัวแปรและข่าวสารซึ่งอ๊อปเจ็กจะตอบสนอง ในตัวอย่างนี้ยังไม่มีวิธีการที่จะมาจักการกับข่าวสาร

```
class employee {
/* Variables */
   string name;
   string address;
   date start-date;
   int salary;
/* Messages */
   int annual-salary();
   string get-name();
   string get-address();
   int set-address(string new-address);
```

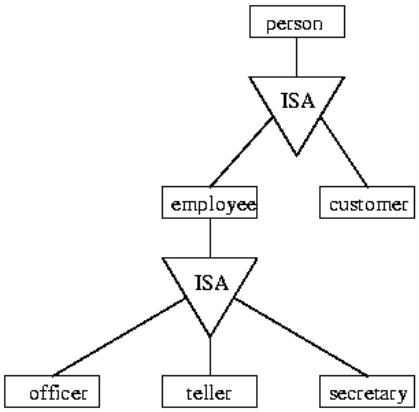
```
int employment-length();
};
```

จากนิยามของคลาส employee จะประกอบไปด้วยตัวแปร name และ address ซึ่งมีชนิดข้อมูลเป็น string start-date มีชนิดข้อมูลเป็น date และ salary มีชนิดข้อมูลเป็น integer แต่ละอ๊อปเจ็กจะตอบสนองข่าวสาร 5 ข่าวสาร คือ annual-salary get-name get-address set-address และ employment-length ข่าวสารแต่ละอันมีการกำหนดชนิดข้อมูลไว้ด้านหน้า หมายถึงชนิดของการตอบสนองข้อมูลกลับไปยังข่าวสารนั้น

3. Inheritance

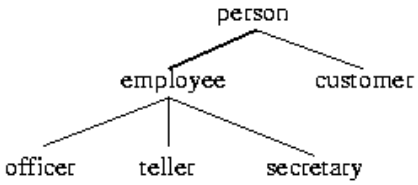
ในระบบฐานข้อมูลเชิงวัตถุจะประกอบไปด้วยคลาสจำนวนมาก อย่างไรก็ตามคลาสหลาย ๆ คลาสจะมีลักษณะคล้าย ๆ กัน ตัวอย่างเช่น พนักงานของธนาคาร จะมีลักษณะที่คล้ายกับลูกค้า เช่น ชื่อ ที่อยู่ โทรศัพท์ เป็นต้น

เพื่อแสดงให้เห็นถึงลักษณะที่คล้ายกันของคลาส เราจะแสดง E-R ไดอะแกรมในลักษณะของลำดับชั้นของ specialization (ความสัมพันธ์แบบ ISA) ดังรูปที่ 1



รูปที่ 1 แสดงลำดับชั้นของคลาสแบบ Specialization

หลักการของลำดับชั้นของคลาสจะเหมือนกับ specialization ใน E-R โมเดล รูปที่ 2 แสดงลำดับชั้นของคลาสที่เหมือน E-R ไดอะแกรมในรูปที่ 1



รูปที่ 2 แสดงลำดับชั้นของคลาสที่เหมือนกับ E-R ในรูปที่ 1

ลำดับชั้นของคลาสสามารถกำหนดเป็นคำสั่งลำลองได้ดังรูปที่ 3 ในที่นี่ยังไม่ขอกล่าวถึงวิธีการของแต่ละคลาส

```
class person {
        string  name;
        string  address;
        };
class customer isa  person {
        int credit-rating;
        };
class employee  isa person {
        date  start-date;
        int  salary;
        };
class officer  isa employee {
        int  office-number;
        int  expense-account-number;
        };
class  teller  isa employee {
        int  hours-per-week;
        int  station-number;
        };
class  secretary  isa  employee {
        int  hours-per-week;
        string  manager;
        };
```

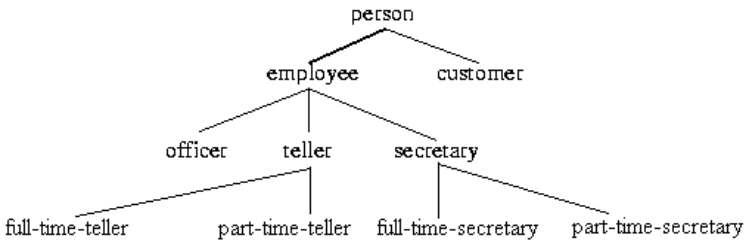
รูปที่ 3 แสดงคำสั่งลำลองในการนิยามลำดับชั้นของคลาส

คำว่า isa ใช้สำหรับบอกว่าคลาสนั้นมีลักษณะเช่นเดียวกันกับอีกคลาสหนึ่ง โดยเราเรียกคลาสที่ specialization ว่า subclasses ตัวอย่างเช่น employee เป็น subclass ของ person และ teller เป็น subclass ของemployee ในทางกลับกัน employee ก็เป็น superclass ของ teller

ข้อดีของการสืบทอดคุณสมบัติในระบบเชิงวัตถุคือ มีคุณลักษณะของ code-reuse นั่นคือวิธีการใด ๆ ของคลาส A สามารถที่จะถูกเรียกใช้โดยคลาส B ซึ่งเป็น subclass A ได้ทันที ทำให้ไม่ต้องเขียนคำสั่งในคลาส B อีก

4. Multiple Inheritance

โดยทั่วไปแล้วโครงสร้างของคลาสแบบลำดับชั้นก็เพียงพอต่อการแสดงโมเดลของข้อมูลแล้ว แต่ในบางกรณี เช่นเราต้องการจำแนกระหว่าง teller และ secretaries แบบ full-time และ part-time ในรูปที่ 2 ซึ่งเราก็สามารถสร้าง subclass ของ part-time-teller full-time-teller part-time-secretary และ full-time-secretary ดังรูปที่ 4

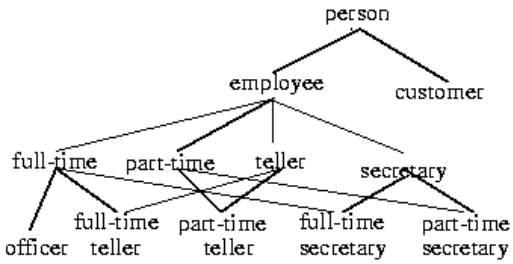


รูปที่ 4 แสดงลำดับชั้นของคลาส ของพนักงาน full-time และ part-time

แต่จะเกิดปัญหาสองอย่างคือ (1) ปัญหาความไม่สอดคล้องในการปรับปรุงข้อมูล เนื่องมาจาก ตัวแปรและวิธีการของพนักงาน full-time ต้องถูกกำหนดสองครั้ง คือ full-time-teller และ full-time-secretary ในทำนองเดียวกันกับพนักงาน part-time เมื่อไรก็ตามที่ต้องมีการเปลี่ยนแปลงข้อมูลของพนักงาน ก็ต้องมีการเปลี่ยนแปลงสองที่ (2) ลำดับชั้นของคลาสไม่สามารถแสดงพนังงานกลุ่มอื่นที่ไม่ใช่ full-time หรือ part-time ได้

วิธีการ Multiple inheritance เป็นวิธีการที่ยอมให้คลาสหนึ่งสืบทอดคุณสมบัติและวิธีการจากหลาย ๆ คลาสได้

ความสัมพันธ์ระหว่าง class และ subclass แสดงได้โดย directed acyclic graph (DAG) ซึ่งใช้ในกรณีที่คลาส มี superclass มากกว่าหนึ่งคลาส



รูปที่ 5 แสดงคลาส DAG

จากรูปที่ 5 เรากำหนดคลาส part-time และคลาส full-time ขึ้นมาเพื่อเก็บข้อมูลและวิธีการของพนักงาน part-time และ full-time จากนั้น สร้างคลาส part-time-teller ซึ่งเป็น subclass ของ teller และ part-time คลาส part-time-teller จะสืบทอดคุณสมบัติ และวิธีการของทั้ง teller และ part-time มา ซึ่งจะเห็นว่าไม่เกิดความซ้ำซ้อนอย่างที่เกิดขึ้นในรูปที่4 แล้ว

เมื่อมีการสืบทอดแบบหลายคลาส อาจจะมีความคลุมเคลือเกิดขึ้นในกรณีที่ตัวแปรหรือวิธีการที่สืบทอดมาจากหลายคลาสมีชื่อเดียวกัน

ตัวอย่างเช่น เรากำหนดการจ่ายเงิน pay ของพนักงานแต่ละประเภทดังนี้

full-time: จ่ายเงินอยู่ระหว่าง 0 ถึง 100,000 ของเงินรายได้ทั้งปี

part-time: จ่ายเงินอยู่ระหว่าง 0 to 20 เป็นค่าแรงต่อชั่วโมง.

teller: จ่ายเงินอยู่ระหว่าง 0 to 20,000 ของเงินรายได้ทั้งปี

secretary: จ่ายเงินอยู่ระหว่าง 0 to 25,000 ของเงินรายได้ทั้งปี

พิจารณาคลาส part-time-secretary ซึ่งสืบทอดตัวแปร pay มาจากคลาส part-time หรือ secretary ผลลัพธ์ที่ได้มีหลายกรณี ขึ้นอยู่กับวิธีการว่าจะจัดการอย่างไร

สืบทอดมาทั้งสองคลาส แล้วเปลี่ยนชื่อของ เป็น part-time-pay และ secretary-pay

เลือกตัวใดตัวหนึ่ง โดยดูจากลำดับของการสร้างคลาส

บังคับให้เลือกเลยว่าจะใช้ pay ตัวไหน

แสดงผลว่าเกิดความผิดพลาด

5.Object Identity

Object identity: อ๊อปเจ็กจะยังคงรักษาความเป็น identity แม้ว่าข้อมูลของตัวแปรหรือวิธีการของ อ๊อปเจ็กมีการเปลี่ยนแปลง แนวคิดของ identity นี้ไม่ได้นำไปใช้กับทูเปิลของฐานข้อมูลเชิงสัมพันธ์ ซึ่งในระบบเชิงสัมพันธ์นั้น ทูเปิลของรีเลชันจะแตกต่างกันด้วยข้อมูลที่ถูกเก็บอยู่

รูปแบบต่าง ๆ ของ identity

Value: หมายถึงค่าของข้อมูลที่ใช้ในการ identity เช่น primary key

Name: เป็นชื่อที่ผู้ใช้กำหนดเพื่อใช้ในการ identity เช่น ชื่อแฟ้มในระบบแฟ้มข้อมูล

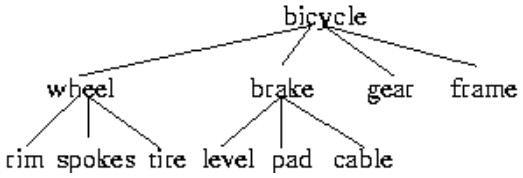
Built-in: เป็นการกำหนด identity โดยอัตโนมัติให้กับข้อมูล หรือภาษาโปรแกรม โดยผู้ใช้ไม่จำเป็นต้องกำหนด identifier รูปแบบนี้ถูกใช้ในระบบเชิงวัตถุ แต่ละอ๊อปเจ็กจะถูกกำหนด identifier โดยอัตโนมัติ เมื่ออ๊อปเจ็กถูกสร้างขึ้น

ในทางปฏิบัติ Object identity จะถูกกำหนดให้มีค่าไม่ซ้ำกัน เรียกว่า OID ซึ่งค่าของ OID นี้จะไม่สามารถมองเห็นได้โดยผู้ใช้ทั่วไป แต่จะถูกใช้โดยระบบเพื่อระบุอ๊อปเจ็กแต่ละอ๊อปเจ็ก

6. Object Containment

อ๊อปเจ็กที่สามารถบรรจุอ๊อปเจ็กอื่น ๆ ได้จะถูกเรียกว่า อ๊อปเจ็กเชิงซ้อน(complex object)หรืออ๊อปเจ็กผสม(composite object) ซึ่งสามารถบรรจุซ้อนกันได้หลายชั้น นั่นคือเป็นลักษณะความสัมพันธ์ที่อ๊อปเจ็กหนึ่งจะประกอบด้วยอ๊อปเจ็กหลาย ๆ อ๊อปเจ็กประกอบเข้าด้วยกัน

ตัวอย่าง A bicycle design database



รูปที่ 6 แสดงลำดับชั้นความสัมพันธ์ระหว่างอ๊อปเจ็ก

รูปที่ 6 แสดงลำดับชั้นความสัมพันธ์ระหว่างอ๊อปเจ็ก ซึ่งความสัมพันธ์ในลักษณะนี้ เป็นลักษณะของ is-part-of ซึ่งเป็นคนละแบบกับ is-a ที่เกิดจากการสืบทอดคลาส

ลักษณะการประกอบกันของอ๊อปเจ็กนี้เป็นหลักการที่สำคัญอย่างหนึ่งในระบบเชิงวัตถุ เนื่องจากจะทำให้การมองข้อมูลเริ่มมองจากจุดเล็ก ๆ ก่อน เช่น คนออกแบบล้อ ก็จะมุ่งไปกับการออกแบบล้อ โดยไม่สนใจกับส่วนอื่นเช่น เกียร์หรือ เบรก จากนั้นพนักงานฝ่ายการตลาดก็จะทำการกำหนดราคาจักรยาน โดยนำเอาข้อมูลส่วนต่าง ๆ ที่เป็นส่วนประกอบของจักรยาน มาคำนวณราคาต่อไป

เรื่องที่ 14.1.2 ภาษาเชิงวัตถุ

1. Object-Oriented Languages

ในการแสดงให้เห็นถึงแนวทางเชิงวัตถุ หลักการต่าง ๆ จะถูกนำไปใช้ในการเขียนโปรแกรมซึ่งสามารถทำได้วิธีใดวิธีหนึ่งในนี้

นำหลักการเชิงวัตถุไปใช้เป็นเครื่องมือในการออกแบบ เพื่อดำเนินการในขั้นต่อไป ตัวอย่างเช่น การออกแบบฐานข้อมูลเชิงสัมพันธ์ เริ่มจากการออกแบบ E-R ไดอะแกรม จากนั้นก็แปลงไปเป็นรีเลชัน

หลักการเชิงวัตถุถูกนำไปรวมในภาษาใดภาษาหนึ่ง เพื่อใช้ในการจัดการฐานข้อมูล ด้วยวิธีการนี้ก็มีอยู่หลาย ภาษาที่สามารถรวมหลักการเชิงวัตถุได้ ดังนี้

เพิ่มความสามารถเชิงวัตถุเข้าไปใน ภาษา SQL เพื่อให้สามารถจัดการกับชนิดข้อมูลที่ซับซ้อนได้ ซึ่งจะเรียกว่า Object-relational system

นำภาษาโปรแกรมที่มีลักษณะเชิงวัตถุอยู่แล้วมาเพิ่มความสามารถทางด้านฐานข้อมูลเข้าไป ซึ่งจะถูกเรียกว่า persistent programming languages

2. Persistent Programming Languages

Persistent data: คือข้อมูลที่ยังคงอยู่แม้ว่าโปรแกรมที่สร้างข้อมูลนั้นขึ้นมาได้ถูกปิดไปแล้วก็ตาม

Persistent programming language คือโปรแกรมภาษาที่เพิ่มส่วนสำหรับจัดการ persistent data ซึ่งแตกต่างจากภาษาที่รวม(embedded)เอา SQL เข้าไว้ในตัวภาษาอย่าน้อย 2 อย่างคือ

1. ในภาษาแบบ embedded ชนิดข้อมูลของ host language จะแตกต่างชนิดข้อมูลของภาษาจัดการข้อมูล ดังนั้นจึงเป็นหน้าที่ของนักเขียนโปรแกรมที่จะทำการแปลงชนิดข้อมูลระหว่าง host language และภาษาจัดการข้อมูล ซึ่งมีอุปสรรคหลายอย่างคือ

โปรแกรมที่ใช้ในการแปลงระหว่างอ๊อปเจ็กและทูเปิล จะไม่ได้มีการดำเนินการในเชิงวัตถุ ซึ่งอาจจะก่อให้เกิดความผิดพลาดขึ้นได้

การแปลงข้อมูลระหว่างข้อมูลที่อยู่ในรูปแบบเชิงวัตถุกับข้อมูลที่อยู่ในรูปแบบเชิงสัมพันธ์ จะต้องเขียนโปรแกรมเป็นจำนวนมาก

ในทางกลับกัน persistent programming language และภาษาสืบค้นข้อมูลถูกรวมเข้าด้วยกัน กับ host language และทั้งสองภาษาต่างก็มีการใช้งานชนิดข้อมูลชนิดเดียวกัน ดังนั้นอ๊อปเจ็กสามารถถูกสร้างและถูกจัดเก็บในฐานข้อมูล โดยไม่ต้องมีการเปลี่ยนแปลงรูปแบบใด ๆ ทั้งสิ้น

2. นักเขียนโปรแกรมใช้งานลักษณะ embedded SQL เพื่อไปดึงข้อมูลจากฐานข้อมูลเข้าสู่หน่วยความจำ ถ้ามีการปรับปรุงข้อมูลก็จะเขียนคำสั่งเพื่อสั่งให้นำข้อมูลบันทึกกลับลงไปสู่ฐานข้อมูล

ส่วนใน persistent programming language นั้น นักเขียนโปรแกรมสามารถจัดการกับ persistent data ได้โดยไม่ต้องมีการเขียนโปรแกรมเพื่อดึงข้อมูลหรือบันทึกข้อมูลกลับลงไปในฐานข้อมูล

2.1 Persistence of Objects

ภาษาโปรแกรมเชิงวัตถุ สนับสนุนหลักการของอ๊อปเจ็กทั้งการนิยามอ๊อปเจ็ก การสร้างอ๊อปเจ็ก แต่อย่างไรก็ตามอ๊อปเจ็กเหล่านี้ก็เกิดขึ้นเพียงชั่วคราวเท่านั้น เมื่อปิดโปรแกรม อ๊อปเจ็กเหล่านั้นก็จะหายไป ดังนั้นขั้นตอนแรกคือต้องหาวิธีทำให้อ๊อปเจ็กยังคงอยู่ ไม่หายไปเมื่อปิดโปรแกรม ซึ่งก็มีอยู่หลายวิธีคือ

Persistence by class เป็นวิธีที่ง่ายที่สุดแต่ไม่เหมาะที่จะนำมาใช้งาน วิธีการนี้เป็นการประกาศว่าคลาสสร้างสร้างขึ้นเป็นแบบ persistent นั่นคืออ๊อปเจ็กทุกอ๊อปเจ็กของคลาสจะเป็น persistent อ๊อปเจ็กไปโดยปริยาย ซึ่งวิธีการนี้ไม่มีความยืดหยุ่น เนื่องจากมันถูกใช้ทั้งแบบชั่วคราวและ persistent ในคลาสเพียงคลาสเดียว ในระบบจัดการฐานข้อมูลเชิงวัตถุ การประกาศคลาสให้เป็นแบบ persistent จะถูกมองว่าเป็นคลาสที่สามารถทำให้ persist ได้ แทนที่จะมองว่าเป็นคลาสที่ persist

Persistence by creation วิธีการนี้เป็นการกำหนดรูปแบบในการสร้าง persistent อ๊อปเจ็ก โดยการเพิ่มรูปแบบสำหรับสร้างอ๊อปเจ็กชั่วคราว ดังนั้นอ๊อปเจ็กที่ถูกสร้างขึ้นจะเป็นแบบ persist หรือแบบชั่วคราว ก็ขึ้นอยู่กับตอนที่สร้างอ๊อปเจ็ก ว่าจะสร้างอย่างไร

Persistence by marking วิธีนี้จะทำการกำหนดให้อ๊อปเจ็กเป็น persistent หลังจากที่ได้ถูกสร้างขึ้นมา ทุก ๆ อ๊อปเจ็กถูกสร้างขึ้นมาแบบชั่วคราว แต่ถ้าอ๊อปเจ็กไหนต้องการที่จะให้คงอยู่หลังจากจบโปรแกรม ก็จะทำการกำหนดก่อนที่จะจบโปรแกรม

Persistence by reference อ๊อปเจ็กหนึ่งหรือหลาย ๆ อ๊อปเจ็กจะถูกกำหนดให้ persist ในลักษณะเป็น root ของอ๊อปเจ็ก ดังนั้นอ๊อปเจ็กทุกตัวที่ถูกอ้างถึงจาก root ก็จะเป็นแบบ persistent

2.2 Object Identity and Pointers

เมื่อ persistent object ถูกสร้าง มันจะถูกกำหนด persistent object identifier

แนวคิดของ object identifier มีลักษณะคล้าย กับ pointer แนวทางง่าย ๆ ในการสร้าง build-in identity คือ ใช้ pointer ชี้ไปที่ ตำแหน่งทางกายภาพของแหล่งเก็บข้อมูล โดยเฉพาะภาษาเชิงวัตถุอย่าง C++ มีการสร้าง

ความสัมพันธ์ระหว่างอ๊อปเจ็กกับตำแหน่งทางกายภาพในแหล่งเก็บข้อมูล อาจมีการเปลี่ยนแปลงได้ตลอดเวลา อย่างไรก็ตาม ได้มีการแบ่งระดับความความถาวรไว้ดังนี้

intraprocedure: Identity จะคงอยู่ในขณะที่กำลังทำคำสั่งของหนึ่งกระบวนความ(procedure)เช่น ตัวแปรชนิด local ภายในกระบวนความ

intraprogram: Identity จะคงอยู่ในขณะที่กำลังทำคำสั่งของหนึ่งโปรแกรมหรือคิวรี เช่นตัวแปรชนิด global

interprogram: Identity จะคงอยู่จากโปรแกรมหนึ่งไปสู่อีกโปรแกรมหนึ่ง เช่น pointers ไปยังข้อมูลในแฟ้มบนดิสก์ แต่มันสามารถเปลี่ยนแปลงได้ขึ้นอยู่กับวิธีการจัดเก็บข้อมูลในแฟ้มถ้ามีการเปลี่ยนแปลงระบบแฟ้ม

persistent: Identity จะคงอยู่ไม่เฉพาะตอนที่โปรแกรมกำลังทำงานเท่านั้น แต่จะเกี่ยวข้องกับกรปรับเปลี่ยนโครงสร้างของข้อมูลด้วย ซึ่งการคงอยู่ในลักษณะนี้จะถูกใช้ในระบบเชิงวัตถุ

2.3 Storage and Access of Persistent Objects

อ๊อปเจ็กถูกจัดเก็บในฐานข้อมูลอย่างไร ข้อมูลของแต่ละอ๊อปเจ็กก็จะถูกจัดเก็บแยกเฉพาะไปแต่ละอ๊อปเจ็ก และส่วนของโปรแกรมที่ใช้เขียนเมธอดก็ควรถูกจัดเก็บไว้ในส่วนของ database schema รวมไปถึงการกำหนดชนิดข้อมูลต่าง ๆ ของคลาส อย่างไรก็ตามส่วนมากจะมีการจัดเก็บไว้ในไฟล์แยกออกมาจากฐานข้อมูล

การค้นหาอ๊อปเจ็กในฐานข้อมูลทำได้หลายวิธีคือ

บอกชื่อของอ๊อปเจ็ก เหมาะกับจำนวนอ๊อปเจ็กน้อย ๆ

ใช้ object identifiers หรือ persistent pointers ไปที่อ๊อปเจ็ก ซึ่งสามารถถูกจัดเก็บไว้ภายนอกได้ ไม่เหมือนกับการบอกชื่อ ตัวชี้เหล่านี้ไม่ต้องมีการทำอะไรอีก มันสามารถที่จะชี้ไปยังตำแหน่งที่เก็บข้อมูลในฐานข้อมูลได้ทันที

เก็บคอลเลกชันของอ๊อปเจ็ก และยอมให้โปรแกรมทำการค้นหาอ๊อปเจ็กที่ต้องการในคอลเลกชันได้ คอลเลกชันของอ๊อปเจ็กสามารถมองเป็น collection type ได้ ประกอบไปด้วย sets multiset list และอื่น ๆ และมีคลาสพิเศษที่เรียกว่า class extent ซึ่งเป็นคอลเลกชันของอ๊อปเจ็กทุกตัวของคลาส ซึ่งเมื่อไรก็ตามที่อ๊อปเจ็กถูกสร้าง อ๊อปเจ็กนั้นก็จะถูกเพิ่มเข้าไปใน class extent ทันที และ เมื่อไรที่อ๊อปเจ็กถูกลบ อ๊อปเจ็กนั้นก็จะถูกลบออกจาก class extent เช่นกัน

ระบบฐานข้อมูลเชิงวัตถุแทบทุกระบบสนับสนุนการเข้าถึง persistent objects ทั้งสามวิธี โดยที่อ๊อปเจ็กทุกตัวต้องมี object identifiers การกำหนดชื่อให้กับ class และ คอลเลกชันของอ๊อปเจ็กอื่น ๆ แต่อ๊อปเจ็กเกือบทั้งหมดไม่ได้ถูกกำหนดชื่อ ดังนั้นเราจะใช้ class extents เพื่อจัดการกับคลาสทุกคลาสที่มี persistent objects

ตอนที่ 14.2 แนวคิดฐานข้อมูลเชิงวัตถุ

หัวเรื่อง

เรื่องที่ 14.2.1 ความสัมพันธ์เชิงกลุ่ม

เรื่องที่ 14.2.2 ชนิดข้อมูลที่ซับซ้อนและแนวทางเชิงวัตถุ

เรื่องที่ 14.2.3 การสืบค้นข้อมูลชนิดที่ซับซ้อนฯ

เรื่องที่ 14.2.4 การสร้างค่าข้อมูลที่ซับซ้อนและการสร้างวัตถุ

เรื่องที่ 14.2.1 ความสัมพันธ์เชิงกลุ่ม

Nested Relations

ลักษณะของรีเลชันที่ 1NF นั้น ค่าของข้อมูลของแต่ละแอตทริบิวต์ในหนึ่งทูเปิลต้องมีค่าเดียวจากโดเมนของแอตทริบิวต์นั้น และโดเมนของแอตทริบิวต์นั้นไม่สามารถแบ่งย่อยได้อีก โดยโมเดลความสัมพันธ์เชิงกลุ่มนี้เป็นส่วนขยายของโมเดลเชิงสัมพันธ์ ซึ่งโดเมนอาจจะเป็นได้ทั้ง atomic หรือเป็นรีเลชันก็ได้ ดังนั้นภายในทูเปิลหนึ่ง ค่าข้อมูลของแอตทริบิวต์หนึ่งอาจมีได้หลายค่า และอาจจะเป็นรีเลชันซ้อนอยู่ในทูเปิลได้ พิจารณา รีเลชัน R(a, b) มีค่าข้อมูลของ b เป็น รีเลชัน(x, y, z) ดังรูป ที่ 1

a	b		
	x	y	z
-	-	-	-
	-	-	-
	-	-	-
	x	y	z
-	-	-	-
	x	y	z
-	-	-	-
	-	-	-

รูปที่ 1 แสดงลักษณะของความสัมพันธ์เชิงกลุ่ม

ยกตัวอย่างความสัมพันธ์เชิงกลุ่มของข้อมูลพนักงาน โดยมีการเก็บข้อมูลดังนี้

employee no รหัสพนักงาน

employee name ชื่อพนักงาน

child name ชื่อบุตรของพนักงาน(มีได้หลายคน)

training list ประวัติการอบรม(มีได้หลายครั้ง)

ถ้าเรากำหนดให้รีเลชันเพียงรีเลชันเดียวเก็บข้อมูลดังกล่าวข้างต้น จะพบว่าข้อมูล children และ training จะมีลักษณะของโดเมนที่ nonatomic ดังรูปที่ 2 และรูปที่ 3 แสดงรีเลชัน employee ที่มีลักษณะของโดเมนที่ atomic ซึ่งมีคุณสมบัติ 1NF

empno	name	childname	training(tno, tdate)
105	John	(Jane, Eric)	(125, 02/05/80) (168, 07/06/81) (198, 03/04/83)
123	Anna	(Maria)	(132, 10/04/80) (145, 06/10/81)

รูปที่ 2 แสดงรีเลชัน Employee ที่ไม่มีคุณสมบัติ 1NF

empno	name	childname	tno	tdate
105	John	Jane	125	02/05/80
105	John	Jane	168	07/06/81
105	John	Jane	198	03/04/83
105	John	Eric	125	02/05/80
105	John	Eric	168	07/06/81
105	John	Eric	198	03/04/83
123	Anna	Maria	132	10/04/80
123	Anna	Maria	145	06/10/81

รูปที่ 3 แสดงรีเลชัน Employee ที่มีคุณสมบัติ 1NF

จะพบว่ารีเลชัน employee ในรูปที่ 3 มีลักษณะของฟังก์ชันการขึ้นต่อกันเชิงกลุ่ม(Multivalued dependencies) ดังนี้

empno → name

empno →→ childname

empno →→ tno, tdate

ดังนั้นเราสามารถแยกรีเลชันให้อยู่ใน 4NF ดังนี้

(empno, name)

(empno, childname)

(empno, tno, tdate)

รูปที่ 4 แสดงรีเลชันที่ได้จากการแตกรีเลชัน employee

empno	Name
105	John
123	Anna

empno	childname
105	Jane
105	Eric
123	Maria

empno	tno	Tdate
105	125	02/05/80
105	168	07/06/81
105	198	03/04/83
123	132	10/04/80
123	145	06/10/81

รูปที่ 4 แสดงรีเลชันที่ 4 NF

รีเลชันที่มีคุณสมบัติ 1 NF ในรูปที่ 3 สามารถแสดงให้เราเห็นถึงลักษณะของข้อมูลที่มีการจัดเก็บในฐานข้อมูล แต่ ถ้าพิจารณารีเลชันในรูปที่ 2 จะพบว่าสามารถที่จะทำความเข้าใจในข้อมูลได้ง่ายกว่า เนื่องจากการสืบค้นข้อมูลโดยทั่วไป จะไม่ได้ สนใจว่าข้อมูลที่ต้องจะต้องอยู่ในรูปของ 1 NF(ค่าข้อมูลเพียงค่าเดียว) แต่จะสนใจกับข้อมูลเป็นกลุ่มมากกว่า ซึ่งจะพบว่ารีเลชันที่มีคุณสมบัติ 4 NF ในรูปที่ 4 จะมีผลทำให้การสืบค้นข้อมูลมีความซับซ้อนมากกว่าเนื่องจากต้องมีการ join ระหว่างรีเลชันมากกว่า

เรื่องที่ 14.2.2 ชนิดข้อมูลที่ซับซ้อนและแนวทางเชิงวัตถุ

ในหัวข้อนี้จะกล่าวถึงส่วนขยายของคำสั่ง SQL ที่ยอมให้มีชนิดข้อมูลที่ซับซ้อน รวมถึงการรีเลชันเชิงกลุ่ม และรูปแบบเชิงวัตถุ โดยรูปแบบของคำสั่งจะอ้างอิงมาตรฐาน SQL-3 ซึ่งคำสั่งจะอยู่ในรูปแบบคำสั่งของ Illustra database system โดย Illustra นี้เป็นเวอร์ชันทางการค้าของ Postgres database system ที่มีการพัฒนาที่มหาวิทยาลัยแห่งแคลิฟอร์เนีย ที่ Berkley ซึ่งเป็นระบบจัดการฐานข้อมูลที่ขยายความสามารถเชิงวัตถุเข้าไปในโมเดลเชิงสัมพันธ์

Structured and collection Types

พิจารณาคำสั่งต่อไปนี้ เป็นการกำหนดรีเลชัน employee ด้วยแอตทริบิวต์เชิงซ้อน

```
create type mystring char varying
create type mytraining
    (tno integer,
    tdate date)
create type emp
    (empno as integer,
    name as mystring,
    childname setof(mystring),
    training setof(mytraining))
create table employee of type emp
```

คำสั่งแรกเป็นการกำหนดชนิดข้อมูล mystring ซึ่งเป็นชนิดสตริงแบบ variable length character คำสั่งที่สองเป็นการกำหนดชนิดข้อมูล mytraining ซึ่งประกอบไปด้วย tno และ tdate และคำสั่งที่สามเป็นการกำหนดชนิดข้อมูล emp ซึ่งประกอบไปด้วย empno name กลุ่มของ children และ กลุ่มของ training และคำสั่งสุดท้ายเป็นการสร้างตาราง employee ซึ่งจากคำสั่งดังกล่าวจะพบว่ามีความแตกต่างจาก การสร้างตารางในระบบฐานข้อมูลเชิงสัมพันธ์ เนื่องจากเรายอมให้แอตทริบิวต์มีลักษณะเป็นเซตได้ และด้วยคุณสมบัตินี้จะยอมให้เราสามารถกำหนดชนิดของข้อมูลเป็นแอตทริบิวต์แบบผสม(composite attribute) และแอตทริบิวต์แบบหลายค่า(multivalued attribute)ได้โดยตรง

เราสามารถสร้างตาราง employee ได้โดยตรง โดยไม่ต้องสร้างชนิดข้อมูลของ emp ขึ้นมาก่อนก็ได้

```
create table employee
    (empno as integer,
     name as mystring,
     childname setof(mystring),
```

training setof(mytraining))

โดยทั่วไปแล้ว ลักษณะชนิดข้อมูลเชิงซ้อนนี้ จะสนับสนุนชนิดข้อมูลแบบคอลเลคชัน(collection : คือข้อมูลที่ไม่มีลำดับ และสามารถมีข้อมูลได้หลาย ๆ ค่าข้อมูล) เช่นอะเรย์(array) และมัลติเซต(multiset) ตัวอย่างเช่น สมมุติว่าเราเพิ่มแอตทริบิวต์ telephone ให้กับรีเลชัน employee ซึ่งพนักงานแต่ละคนสามารถมีโทรศัพท์ได้หลายหมายเลข เราจะเพิ่มแอตทริบิวต์ telephone ดังนี้

telephone mystring[3]

ซึ่งหมายถึงอะเรย์ของหมายเลขโทรศัพท์ของพนักงานแต่ละคนนั่นเอง ซึ่งถ้าเราต้องการทราบว่าหมายเลขโทรศัพท์หมายเลขที่หนึ่งคือหมายเลขอะไร ก็สามารถบอกได้ทันที เพราะมีการอ้างถึงข้อมูลแบบอะเรย์ แต่ถ้าเรากำหนดข้อมูลโทรศัพท์นี้เป็น setof(mystring) เราจะไม่สามารถบอกได้เลยว่าหมายเลขโทรศัพท์ใดเป็นลำดับที่เท่าไร

Inheritance

การสืบทอดก็สามารถดำเนินการกับชนิดของข้อมูลได้เช่นกัน ตัวอย่างเช่น สมมุติว่าเราได้กำหนดชนิดข้อมูลของบุคคลดังนี้

```
create type person
    ( name mystring,
    social-security integer)
```

จากนั้นเราต้องการเก็บข้อมูลเกี่ยวกับนักเรียน และครู ซึ่งทั้งนักเรียนและครู ต่างก็เป็นบุคคลด้วยกันทั้งคู่ ดังนั้นเราสามารถใช้คุณสมบัติการสืบทอดมากำหนดชนิดข้อมูลนักเรียน และครูได้ดังนี้

```
create type student
    ( degree mystring,
    department mystring )
    under person
create type teacher
    ( salary integer,
    department mystring )
    under person
```

ทั้งนักเรียนและครูต่างสืบทอดคุณสมบัติของ person คือ name และ social-security กล่าวได้ว่า student และ teacher เป็น subtype ของ person และ person เป็น supertype ของ student และ teacher

สมมุติว่า เราต้องการเก็บข้อมูลของผู้ช่วยสอน(teacher assistant) ซึ่งเป็นทั้งนักเรียนและครูในเวลาเดียวกัน และอาจจะอยู่ต่างแผนกกัน แนวคิดนี้เรียกว่า multiple inherit ถ้าระบบสนับสนุน multiple inherit เราสามารถกำหนดชนิดข้อมูล teaching assistant ได้ดังนี้

```
create type teachingassistant
    under student, teacher
```

teachingassistant จะสืบทอดคุณสมบัติทั้งหมดของ student และ teacher จะพบว่าแอตทริบิวต์ social-security name และ department ปรากฏอยู่ในทั้ง student และ teacher ซึ่งแอตทริบิวต์ social-security และ name ได้ถูกสืบทอดมาจาก person เหมือนกัน ดังนั้นจึงไม่เกิดความขัดแย้งจากการสืบทอดจาก student และ teacher แต่ แอตทริบิวต์ department อาจเกิดความขัดแย้งได้เนื่องจาก teaching assistant อาจจะเป็นนักเรียนในแผนกหนึ่ง แต่อาจารย์จะอยู่อีกแผนกหนึ่ง ดังนั้นในการหลีกเลี่ยงความขัดแย้งนี้ เราจะใช้วิธีการเปลี่ยนชื่อ ดังนี้

```
create type teachingassistant
    under student with (department as student-department) ,
    teacher with (department as teacher-department)
```

Reference Type

ภาษาเชิงวัตถุสนับสนุนการอ้างอิงอ๊อปเจ็ก โดยที่แอตทริบิวต์หนึ่ง ๆ ของ ชนิดข้อมูลสามารถที่จะอ้างอิงไปอีกอ๊อปเจ็กหนึ่งได้ ตัวอย่างเช่นการอ้างอิงไปที่รีเลชัน person ของแอตทริบิวต์ childname สามารถเขียนใหม่ได้ดังนี้

```
childname   setof(ref(person))
```

หมายถึงแอตทริบิวต์เป็นเซตของการอ้างอิงไปที่อ๊อปเจ็ก person เราสามารถจัดการกับการอ้างอิงไปที่ทูเปิลของรีเลชัน person โดยการใช้คีย์หลักของรีเลชัน person มาเป็นข้อมูลใน childname เพื่อทำหน้าที่ foreign key หรืออีกวิธีหนึ่งคือ ใช้ tuple identifer ของแต่ละทูเปิลเป็นตัวอ้างอิงแทน

เรื่องที่ 14.2.3 การสืบค้นข้อมูลชนิดที่ซับซ้อน

ในหัวข้อนี้จะแสดงให้เห็นถึงการใช้คำสั่ง SQL เพื่อสืบค้นข้อมูลที่มีชนิดข้อมูลที่ซับซ้อนในลักษณะต่างๆ

Relation-Valued Attributes

จากรีเลชัน employee ถ้าต้องการค้นหาชื่อของพนักงานที่มีบุตรชื่อ Jane เราสามารถเขียนเป็นคำสั่ง select ได้ดังนี้

```
select name
```

from employee

where “Jane” in childname

เราเรียกแอตทริบิวต์ childname ว่า relation-valued attribute ซึ่งจะปรากฏอยู่ในส่วนของ where ในคำสั่ง Select หรือ จงแสดงชื่อของพนักงาน และชื่อของบุตร พนักงานแต่ละคน สามารถเขียนคำสั่ง select ได้ดังนี้

select E.name, C.childname

from employee as E, E.childname as C

เนื่องจากแอตทริบิวต์ childname ของรีเลชัน employee เป็นฟิลด์ประเภท set-valued ดังนั้นจึงสามารถใช้ในส่วนของ fromได้

เราสามารถใช้ aggregate ฟังก์ชันเพื่อดำเนินการกับกลุ่มข้อมูลได้ เช่นจงแสดงชื่อพนักงาน และจำนวนบุตรของพนักงานแต่ละคน สามารถเขียนเป็นคำสั่ง select ได้ดังนี้

select name, count(childname)

from employee

Path Expression

เราใช้สัญลักษณ์จุด (.) เพื่อการอ้างอิงแอตทริบิวต์ที่ประกอบกันเป็นแอตทริบิวต์ผสมได้ เช่น สมมุติเรามีรีเลชันที่เก็บข้อมูล phd-student และรีเลชัน person โดยกำหนดนิยามของรีเลชันดังนี้

create table phd-students

(advisor ref(person))

under person

ดังนั้นถ้าเราต้องการค้นหาข้อมูลชื่อของ advisor ของนักเรียนปริญญาเอกทั้งหมด สามารถเขียนเป็นคำสั่ง ได้ดังนี้

select phd-students.advisor.name

from phd-students

เนื่องจาก phd-students.advisor มีการอ้างถึงแอตทริบิวต์ name ในตาราง people ซึ่งการอ้างอิงนี้ใช้เพื่อซ่อนการ join ระหว่างรีเลชัน จากตัวอย่างข้างต้น ถ้าเราไม่ได้ใช้วิธีการอ้างอิง ฟิลด์ advisor ของรีเลชัน phd-student จะต้องเป็น foreign ไปที่รีเลชัน person จากนั้นก็จะทำการ join ระหว่างรีเลชัน phd-student กับ person ด้วย foreign key โดยเราจะเรียกนิพจน์ phd-students.advisor.name นี้ว่า path expression

Nesting and Unnesting

การทำ unnesting คือการแปลงความสัมพันธ์เชิงกลุ่มให้อยู่ในรูปแบบ 1 NF พิจารณารีเลชัน employee มีแอตทริบิวต์ childname และ training ที่เป็นความสัมพันธ์เชิงกลุ่ม สมมุติว่าเราต้องการแปลงรีเลชัน employee ให้อยู่ในรูปแบบที่ไม่มีรีเลชันเชิงกลุ่ม สามารถทำได้โดย

```
select empno, name, C as childname, T.tno as tno, T.tdate as tdate
    from employee As E, E.childname As C, E.training As T
```

โดยที่ตัวแปร E ในประโยค from เป็นการประกาศขอบเขตของรีเลชัน employee ตัวแปร C กำหนดขอบเขตแอตทริบิวต์ childname และตัวแปร T ก็จะกำหนดขอบเขตแอตทริบิวต์ training

ในทางกลับกันถ้าทำการแปลงจากรีเลชันที่อยู่ในรูป 1 NF ไปเป็นรีเลชันเชิงกลุ่มจะเรียกว่าการทำ nesting ในการทำ nesting จะใช้วิธีการจัดกลุ่ม ซึ่งเมื่อทำการจัดกลุ่มแล้ว จะเหมือนกับมีการสร้าง multiset รีเลชันของแต่ละกลุ่มขึ้นมา และทำการส่ง multiset กลับคืนมาแทนที่จะใช้ฟังก์ชัน aggregate ตัวอย่างเช่น จากรูปที่ 3 เป็นรีเลชันที่อยู่ในรูป 1 NF เราต้องการแปลงให้อยู่ในรูปของ ความสัมพันธ์เชิงกลุ่ม สามารถเขียนเป็นคำสั่ง ได้ดังนี้

```
select empno, empname, set(childname) as childname-list, set(training) as training-list
    from employee
    group by empno, empname
```

เรื่องที่ 14.2.4 การสร้างค่าข้อมูลที่ซับซ้อนและการสร้างวัตถุ

ที่ผ่านมาเป็นการกำหนดชนิดข้อมูลที่ซับซ้อน และการสืบค้นข้อมูลในรีเลชันที่มีการกำหนดชนิดข้อมูลที่ซับซ้อน ในหัวข้อนี้จะเป็นการสร้างและปรับปรุงข้อมูลในรีเลชันที่มีชนิดข้อมูลที่ซับซ้อน

เราสามารถเพิ่มข้อมูลหนึ่งทูเปิลในรีเลชัน employee ได้ดังนี้

(105, “John”, set(“Jane”, “Eric”), set((125, “02/05/80”), (168, “07/06/81”), (198, “03/04/83”)))

จากคำสั่งดังกล่าว จะเห็นว่าเราสร้างค่าข้อมูลของแอตทริบิวต์ผสม(composite attribute) โดยการกำหนดเป็นรายการไว้ในวงเล็บ และใช้คำหลัก set เป็นตัวบอกว่า เป็นชนิดข้อมูลที่ซับซ้อน ดังนั้นถ้าจะเขียนให้อยู่ในรูปของคำสั่ง SQL สามารถเขียนได้ดังนี้

```
insert into employee values (
    105, “John”, set(“Jane”, “Eric”),
    set((125, “02/05/80”), (168, “07/06/81”), (198, “03/04/83”)) )
```

อีกทั้งเราสามารถที่จะใช้ข้อมูลที่ซับซ้อนในการสืบค้นข้อมูลได้อีกด้วย ดังตัวอย่างเช่น

```
select  empno, name
```

```
from employee
where name in set("John","Anna","Tom")
```

คำสั่งข้างต้นเป็นการสืบค้นข้อมูล รหัสพนักงานและชื่อพนักงาน จากรีเลชัน employee ที่มีชื่อเป็น “John” หรือ “Anna” หรือ “Tom”

เราสามารถสร้างค่าข้อมูลแบบ multiset โดยใช้คำสั่ง multiset แทน set ได้ ซึ่งผลลัพธ์จะเป็นการสร้างรายการหรืออะเรย์ของข้อมูลภายในทูเปิลหนึ่ง ๆ

ในการสร้างอ๊อปเจ็กของข้อมูล เราจะใช้ฟังก์ชัน constructor ซึ่งฟังก์ชัน constructor ของชนิดข้อมูล T คือ T() เมื่อ constructor ถูกเรียกขึ้นมาสร้างอ๊อปเจ็กใหม่ของ T ก็จะทำการกำหนด oid ให้กับอ๊อปเจ็กและส่งอ๊อปเจ็ก กลับมา จากนั้นข้อมูลแต่ละตัวของอ๊อปเจ็กก็จะถูกกำหนดค่าข้อมูล

เราสามารถใช้คำสั่ง update เพื่อปรับปรุงข้อมูลในรีเลชันที่ซับซ้อนได้ เหมือนกับคำสั่ง update ที่ใช้ในฐานข้อมูลเชิงสัมพันธ์