บทที่ 9
ระบบฐานข้อมูลเชิงวัตถุ
(Object-Oriented Database Systems)

The Object-Oriented Data Model

Object-Oriented Database Systems คือส่วนที่เพิ่มเติมจาก basic features ของ Object-Oriented Programming languages โดยลักษณะต่างๆ จะอธิบายในหัวข้อต่อไปนี้

9.1 โครงสร้างของวัตถุ (Object Structure)

หากจะพูดอย่างง่ายๆ object ก็จะตรงกับ entity ใน E-R model ซึ่งแนวคิดของ Object-Oriented จะมีพื้นฐานบนหลักของการซ่อนสารสนเทศ (Encapsulation) ของข้อมูลและ code ที่เกี่ยวข้องกับหนึ่ง object ในหนึ่ง Single unit ตามแนวคิดแล้วการติดต่อต่างๆ ระหว่าง object และส่วนต่างๆ ของระบบจะผ่านทาง messages นั่นคือการติดต่อระหว่างหนึ่ง Object และส่วนต่างๆ ของระบบจะถูกกำหนดโดยเซตของ messages ได้รับอนุญาติแล้ว

โดยทั่วไปวัตถุหนึ่งๆ จะประกอบด้วย

- เซตของ variables ซึ่งเก็บ data สำหรับ object หนึ่งๆ variables อาจเทียบได้กับ attributes ใน E-R model
- เซตของ messages ซึ่ง object ต้องใช้ในการได้ตอบกับ objects อื่นๆ หรือใช้ในการจัดการกับ variables โดยแต่ละ messages อาจมีจำนวน parameters เป็นศูนย์, หนึ่ง หรือมากกว่านั้น
- เซตของ method ซึ่งแต่ละส่วนคือ body ของ code ที่ใช้ในการสร้างหนึ่ง message ซึ่ง method จะส่งคืนค่าหนึ่งค่าสำหรับการตอบรับ message

หมายเหตุ : ในบางครั้งเราเรียก methods ว่า operations ส่วน messages บางครั้งถูกเรียกว่าเป็น interfaces ของ objectsและส่วนของ variables บางครั้งถูกเรียกว่าเป็น object states หรือ values สำหรับ methods ที่ใช้ในการสร้าง object เราเรียกว่า constructors

"message ใน Object-Oriented context หมายถึงการส่งผ่านค่า request ระหว่าง objects โดยไม่ต้องระบุถึง implementation details สำหรับศัพท์ invoke a method ใช้สำหรับแสดงถึงการส่งหนึ่ง message ไปยังหนึ่ง object แล้ว method ที่ตรงกันกับ message นั้นจะเกิดการทำงาน"

CT 316 (S)

ส่งหนึ่ง message ไปยังหนึ่ง object แล้ว method ที่ตรงกันกับ message นั้นจะเกิดการทำงาน"

"นั่นคือ objects มีการติดต่อกันโดยผ่านทาง messages เมื่อ object ได้รับ message แล้ว method ของ object นั้นๆ ที่ตรงกันกับ message ที่ได้รับจะเกิดการทำงาน"

เราสามารถอธิบายการเคลื่อนที่ในแนวนี้ได้ด้วยการพิจารณาตัวอย่างของ employee entities ใน bank ฐานข้อมูล ซึ่งโบนัสของผู้ที่ทำงานแต่ละคนจะถูกคำนวณต่างกัน ตัวอย่างเช่น ผู้จัดการได้รับโบนัสโดยขึ้นอยู่กับประสิทธิภาพในการบริหารงานในขณะที่พนักงานรับจ่ายเงินจะได้รับโบนัสโดยขึ้นอยู่กับจำนวนชั่วโมงที่ทำงาน โดยแนวคิดเราจะซ่อน code สำหรับการคำนวณเงินเดือนสำหรับพนักงานแต่ละคนเช่นเดียวกับ method ที่ใช้ในการตอบรับต่อ annual-salary message

ทุก employee objects ตอบรับไปยัง annual-salary message เหมือนกันแต่จะทำงานในลักษณะที่ต่างกัน (ด้วยการซ่อนรายละเอียดของการคำนวณ annual salary ภายใน employee object ทำให้ employee object ทั้งหมดมี interface เหมือนกัน) และเนื่องจาก external interfaces ของ object คือเซตของ messages ซึ่ง object นั้นๆ ต้องตอบรับ จึงเป็นไปได้ที่จะแก้ไขการกำหนดของ methods และ variables ได้ โดยไม่มีผลกระทบ ต่อส่วนที่เหลือของระบบ ซึ่งข้อดีนี้มาจากผลของการซ่อนรายละเอียดซึ่งเราเรียกว่า encapsulation โดยเป็นส่วนสำคัญหลักสำหรับแนวคิดของ object-oriented programming

Methods ของ object หนึ่งแบ่งได้เป็น read-only หรือ update ซึ่ง read-only method จะไม่มีผลกระทบต่อมูลค่าของตัวแปร (หรือ state) ใน object หนึ่งๆ ในขณะที่ update method อาจเปลี่ยนมูลค่าของตัวแปรได้ โดย messages ซึ่ง object ตอบรับสามารถจำแนกชนิดเป็น read-only หรือ update ได้เช่นเดียวกัน ทั้งนี้ขึ้นอยู่กับ method ที่สร้างขึ้นสำหรับ message นั้นๆ

Derived attributes ของ entity ใน E-R model จะถูกแสดงใน object-oriented model เป็น read-only messages ตัวอย่างเช่น derived attribute employment-length ของหนึ่ง employee entity จะถูกแสดงเป็น employment-length message ไปยังหนึ่ง employee object (method ที่สร้างขึ้นเพื่อตอบรับ message นี้ ใช้สำหรับแสดงอายุการทำงานซึ่งคำนวณได้โดยการนำ start-date มาลบออกจาก current-date ของ employee)

หากพูดอย่างเข้มงวดแล้วสำหรับ object-oriented model ทุกๆ attribute ของหนึ่ง entity ต้องถูกแสดงเป็น variable คู่กับ messages ที่ใช้กับ object นั้น โดย variable จะถูกใช้สำหรับเก็บค่าของ attribute และหนึ่ง message สำหรับการอ่าน attribute value ส่วน method อื่นๆ จะถูกใช้สำหรับการ update values ของ objects ตัวอย่างเช่น attribute address ของ employee entity ใน

E-R model สามารถถูกแสดงโดย :

- A variable address
- message get-address ที่เข้ากันกับ address
- message set-address ซึ่งใช้ parameter new-address ในการ update address

อย่างไรก็ตามเพื่อความง่าย มี object-oriented data model จำนวนมากอนุญาติให้ variables สามารถ read หรือ update ได้โดยตรง (นั่นคือไม่ต้องมีการกำหนด messages สำหรับการ read หรือ update อีก)

ประเภทการทำงานของวัตถุ (Classification of Operations)

1. Primitive Constructor. (ตัวสร้างชนิดเบื้อต้น) จัดเป็นการทำงานที่ใช้สำหรับสร้าง instance ของ object type แต่ละชนิดตัวอย่างเช่น matrix$create(i) ใช้สำหรับสร้าง unity matrix ที่มีมิติ = i x i

2. Constructors. การทำงานของตัวสร้างชนิดนี้ต่างจากตัวสร้างชนิดเบื้องต้น ตรงที่มีการสร้าง instance ใหม่ เพื่อทำงานบางอย่าง เช่น m.inverse() จะสร้าง matrix instance ตัวใหม่ ซึ่ง derive มาจาก matrix ที่มีอยู่เพื่อเป็นผลลัพธ์สำหรับการหา inverse ของ matrix

3. Observer functions. เป็นการทำงานซึ่งส่งสารสนเทศที่มีอยู่ใน internal state (variable) ของ object instance (ในลักษณะ copy) ไปยังการทำงานอื่นๆ ที่ต้องการประยุกต์ใช้ เช่น การทำงาน elem จะนำอาร์กิวเมนต์สามตัวได้แก่ m(a matrix) และ i,j (two integers) จากนั้นจะส่งคืนค่าตำแหน่ง (i,j) ของ matrix instance m การเรียกใช้จะมีลักษณะดังนี้ m.elem(m,i,j)

4. Mutators. เป็นการทำงานซึ่งเปลี่ยน internal state ของ object instance ซึ่งเรียกใช้การทำงานเหล่านี้ เช่น การทำงาน m,.add(m_2) จะทำการบวก matrix m_2 เข้ากับ matrix m_1 ซึ่งเรียก method add โดย matrix m_1 และ m_2 จะต้องมีมิติที่เท่ากัน

การทำงานในสามจำพวกแรกมักถูกเรียกว่า function สำหรับในภาษาโปรแกรมเพราะการทำงานเหล่านี้จะมีการส่งค่ากลับและปราศจาก side effect ซึ่งกรณีนี้หมายความว่าการทำงานเหล่านั้นจะไม่ทำให้เกิดการเปลี่ยนแปลงสถานะภายในของวัตถุที่เรียกใช้มันสำหรับ mutator เป็นการทำงานที่มักถูกเรียกว่า procedure เพราะว่าเปลี่ยนสถานะภายในบางอย่างของวัตถุในฐานข้อมูล และเราเรียก object type ที่มีการประกาศ mutator ว่า mutable ส่วน object type ที่ไม่มีประกาศ mutator เลยจะเรียกว่า immutable

// ในภาษาโปรแกรมบางภาษา เช่น C++ มักเรียก object type ว่า class

9.2 กลุ่มวัตถุ (Object Classes)

โดยปกติ objects จำนวนหนึ่งในฐานข้อมูลมักคล้ายกัน หมายถึง objects เหล่านั้นต่างตอบรับต่อ message ที่เหมือนกัน (โดยใช้ method เดียวกัน) และมี variables ซึ่งมีชื่อและ type เดียวกัน จึงเป็นการสิ้นเปลืองมากที่จะนิยามแต่ละ object แยกกัน ดังนั้นเราจึงจัดให้ objects ที่เหมือนกันมีรูปแบบเดียวกันซึ่งเรียกว่า class โดยแต่ละ object จะถูกเรียกว่าเป็น instance ของ class นั้นๆ objects ทั้งหมดใน class หนึ่งๆ จะใช้การนิยามรวมกัน อย่างไรก็ตามค่าที่กำหนดให้ variables จะต่างกันไป

แนวความคิดของ class ใน object-oriented data model ตรงกับแนวคิดของ entity-set ใน E-R model ตัวอย่างเช่น classes ใน bank ฐานข้อมูล ได้แก่ employee, customers, accounts และ loans

ตัวอย่างต่อไปนี้เป็นการกำหนด class employee โดยใช้ pseudocode การกำหนดแสดงถึง variables และ messages ซึ่ง objects ของ class ต้องตอบรับ สำหรับ method ซึ่งใช้สร้าง messages ไม่ถูกแสดงไว้ในที่นี้

```
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-addrtess());
        int employemnt-length(); };
```

ในการนิยามข้างบนแต่ละ object ของ class employee ประกอบด้วย variables name และ address ซึ่งเป็น string , start-date เป็น date และ salary ซึ่งเป็น integer แต่ละ object จะตอบรับห้า messages ดังแสดงไว้โดยมีชื่อดังนี้ annual-salary, get-name, get-address, set-address, employment-length

สำหรับ type name ก่อน message mane แสดงถึง type ที่ใช้ในการตอบกลับของ message สำหรับ message set-address จะใช้ parameter new-address ซึ่งระบุถึงมูลค่าใหม่ของ address

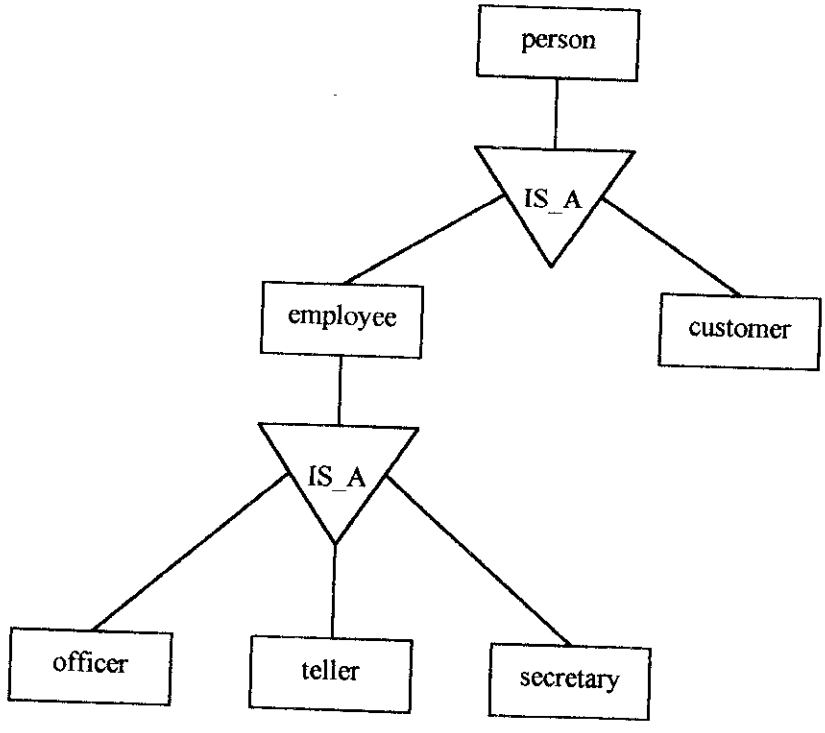
หลักการของ class คล้ายกับหลักการของ abstract data types อย่างไรก็ตามมีหลักเกณฑ์หลายอย่างที่เพิ่มขึ้น ในการแสดงถึงสมบัติที่เพิ่มเติมเหล่านี้ เราจะปฏิบัติกับ class เช่นเดียวกับ object ของมัน โดยหนึ่ง class object ประกอบด้วย set-valued variable ซึ่งมีมูลค่าคือ set ของ object ทั้งหมดที่เป็น instances ของ class นั้นๆ และสร้างหนึ่ง method สำหรับตอบรับ message new ซึ่งใช้สร้าง instance ใหม่ของ class

9.3 การสืบทอด (Inheritance)

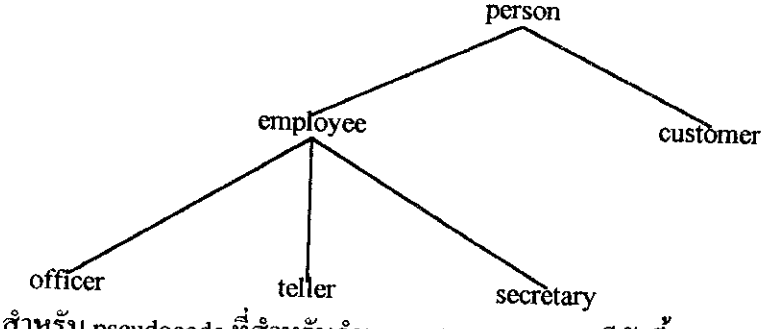
บ่อยครั้งที่ classes มักจะมีลักษณะคล้ายกัน ตัวอย่างเช่น สมมุติว่าเรามี object-oriented database สำหรับ bank application เราอาจคาดได้ว่า class ของ bank customers มีลักษณะเหมือนกัน class ของ bank employees ซึ่งทั้งสองมีการกำหนด variables สำหรับ mane, address และอื่นๆ อย่างไรก็ตาม variables ที่ใช้สำหรับระบุว่าเป็น employee (ตัวอย่างเช่น salary) และ variables ที่ใช้สำหรับระบุว่าเป็น customers (ตัวอย่างเช่น credit-rating) มีความต่างกัน เราสามารถกำหนด common variables ไว้ในที่เดียวกัน แล้วจึงกำหนดลักษณะเฉพาะเพิ่มเติมในภายหลังได้ ซึ่งสมบัตินี้เรียกว่า Inheritance

9.3.1 การสืบทอดแบบปกติ (Normal Inheritance)

เราสามารถได้โดยใช้ E-R diagrams ซึ่งมี specialization hierarchy เพื่อแสดงถึงความสัมพันธ์ของ entity types ต่างๆ ใน bank ฐานข้อมูล ได้ดังรูปนี้



และด้วยสมบัติของการ inheritance จะได้ class hierarchy ดังรูปต่อไปนี้



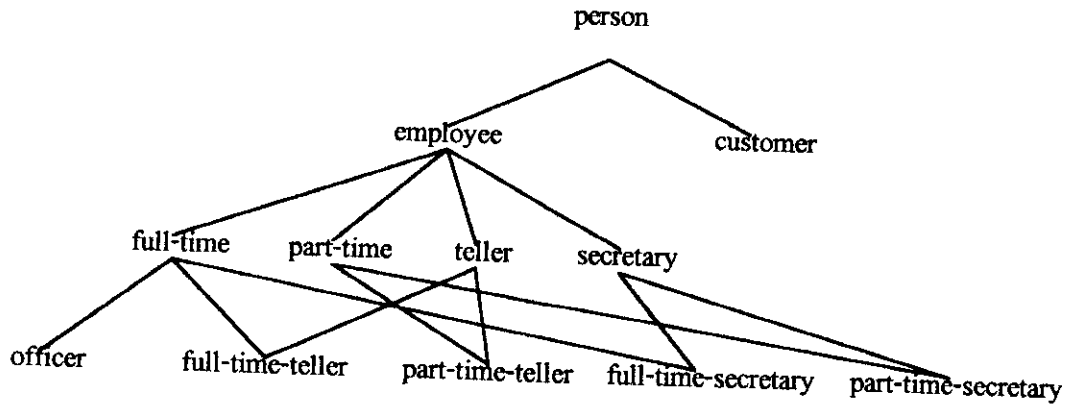
สำหรับ pseudocode ที่สำหรับกำหนด class hierarchy มีดังนี้

```
class  person {
        string  name
        string  address
};
```

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

9.3.2 การสืบทอดแบบหลายส่วนและแบบเลือก (Multiple Inheritance and Selective Inheritance)

การสืบทอดแบบหลายส่วนในแบบลำดับชั้นเกิดขึ้นเมื่อมีหนึ่ง subclass เป็นคลาสย่อยของสอง (หรือมากกว่า) คลาสที่ต่างกันและได้หน้าที่ (คุณลักษณะและการทำงาน superclass ทั้งสอง) การสืบทอดในลักษณะนี้จะทำให้เกิดการสร้าง class แบบโครงข่ายมากกว่าที่จะเป็นการสร้าง class แบบลำดับชั้นซึ่งจะถูกแสดงโดย direct a cyclic graph (DAGO ตัวอย่างเช่น Class DAG สำหรับ bank example ดังต่อไปนี้



ส่วนการสืบทอดแบบเลือกเกิดขึ้นเมื่อ subclass รับหน้าที่เพียงบางส่วนของ supberclass เท่านั้น โดยหน้าที่อีกส่วนหนึ่งไม่ได้ถูกสืบทอดมา

9.4 สิ่งระบุวัตถุหรือเอกลักษณ์ของวัตถุ (Object Identity)

ฐานข้อมูลเชิงวัตถุจะมีการกำหนดเอกลักษณ์พิเศษ (unique identity) ให้แต่ละวัตถุที่เก็บอยู่ในฐานข้อมูล เอกลักษณ์พิเศษนี้จะถูกสร้างขึ้นอัตโนมัติโดยระบบ สิ่งที่ได้เรียกว่า สิ่งระบุวัตถุ (object identifier) หรือ OID โดยที่มูลค่าของ OID จะไม่สามารถเห็นได้จากผู้ใช้ แต่จะถูกใช้ภายในระบบเพื่อบ่งชี้ถึงวัตถุแต่ละชนิดหรือเพื่อสร้างและจัดการการอ้างอิงระหว่างวัตถุ

สิ่งที่สำคัญสำหรับ OID คือจะต้องไม่มีการเปลี่ยนแปลง (immutable) นั่นคือมูลค่าของ OID สำหรับวัตถุแต่ละชนิดจะไม่ถูกเปลี่ยนแปลง (เพื่อประกันการระบุวัตถุที่ถูกต้อง) ซึ่งเป็นข้อดี คือ OID แต่ละค่าจะถูกใช้เพียงครั้งเดียวถึงแม้ว่าวัตถุจะถูกย้ายออกไปจากฐานข้อมูล OID ของวัตถุนั้นก็จะไม่ถูกกำหนดค่าให้กับวัตถุอื่น ดังนั้น OID ไม่ควรขึ้นอยู่กับ attribute ของวัตถุเพราะค่า attribute ของวัตถุสามารถเปลี่ยนแปลงได้

"นั่นคือวัตถุทุกชนิดที่ถูกสร้างขึ้นในระบบฐานข้อมูลเชิงวัตถุจะมีการกำหนดตัวบ่งชี้ (OID) โดยอัตโนมัติซึ่ง OID นี้ก็ทำหน้าที่คล้ายกับ primary key ใน relational databases แต่ผู้ใช้จะไม่สามารถมองเห็นได้และไม่ได้เป็นส่วนหนึ่งของ attribute ที่ใช้เก็บข้อมูลของ user"

รูปแบบต่างๆ ของสิ่งระบุวัตถุ (Object Identity form)

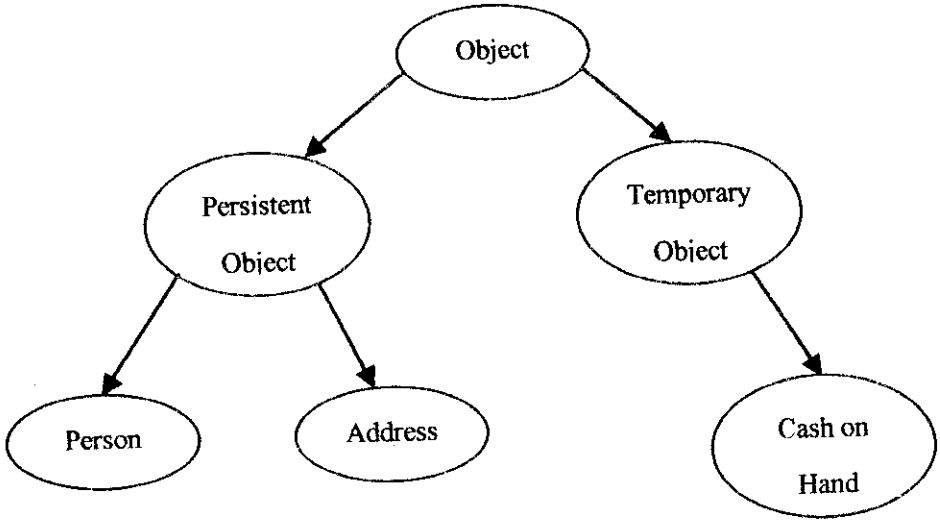
Value. เป็นสิ่งที่ถูกใช้ใน Relational systems เช่น มูลค่าของ primary key ของ tuple จะเป็นตัวระบุถึง tuple

Name. เป็นสิ่งที่ใช้ในระบบไฟล์ เช่น ทุกครั้งที่กำหนดชื่อไฟล์จะเป็นการระบุถึงไฟล์

Built-in. เป็นสิ่งที่ใช้ในระบบ OO เช่น ทุกวัตถุจะถูกกำหนดค่าโดยอัตโนมัติเมื่อวัตถุถูกสร้างขึ้น

9.5 ความทนทานของวัตถุและการอ้างอิงอรรถศาสตร์ (Persistence of Objects and Semantic Reference)

object ในฐานข้อมูลถูกแบ่งออกเป็น 2 ประเภทคือ วัตถุทนทาน (persistent object) และวัตถุไม่ทนทาน (transient (temporary) object) ซึ่งแสดงตัวอย่างได้ดังรูปต่อไปนี้



วัตถุทนทานคือวัตถุที่จะถูกเก็บไว้ใน ฐานข้อมูล และจะไม่หายไปกับโปรแกรมที่จบการทำงาน ส่วนวัตถุไม่ทนทานจะหายไปเมื่อโปรแกรมจบการทำงาน

9.5.1 แนวทางในการทำให้วัตถุมีความคงที่

- **Persistence by class.** คือแนวคิดที่ทำให้คลาสเป็นสิ่งที่ทนทาน ทุกวัตถุของคลาสจะทนทานโดยปริยาย ส่วนวัตถุของคลาสที่ไม่คงที่ก็จะหายไปเมื่อโปรแกรมจบการทำงาน

- **Persistence by creation.** เป็นการกำหนดความสัมพันธ์ (syntax) แบบใหม่ในการประกาศให้วัตถุเป็นแบบคงที่
- **Persistency by making.** เป็นการกำหนดวัตถุให้เป็นแบบคงที่ก่อนที่โปรแกรมจะจบการทำงาน
- **Persistency by reference.** วัตถุหนึ่งหรือมากกว่าจะถูกประกาศอย่างชัดแจ้งให้เป็นวัตถุคงที่ (root persistent object) วัตถุอื่นๆ จะเป็นแบบคงที่ทั้งหมดถ้าหากว่าวัตถุเหล่านั้นถูกอ้างอิงโดยตรงหรือทางอ้อมจากวัตถุที่เป็นวัตถุคงที่

9.5.2 สิ่งระบุวัตถุและตัวชี้ (Object Identity and Pointers)

แนวทางหนึ่งในการสร้าง built-in identify คือการกำหนดตัวชี้ (pointer) ไปยังตำแหน่งทางกายภาพ (physical location) ของหน่วยจัดเก็บ (storage) อย่างไรก็ตามการรวมกันของวัตถุกับตำแหน่งทางกายภาพอาจจะมีการเปลี่ยนแปลงได้เมื่อเวลาผ่านไป ซึ่งระดับความคงทนของเอกลักษณ์ (identity) มีดังนี้

- **Intraprocedure.** เอกลักษณ์จะคงอยู่ในขณะที่มี procedure ใด procedure หนึ่งทำงานอยู่เท่านั้น เช่น local variables ใน procedure
- **Interprogram.** เอกลักษณ์จะคงอยู่เฉพาะตอนที่มีการทำงานของโปรแกรมหรือดิวรี่หนึ่งๆ เช่น global variables ในภาษาโปรแกรม
- **Interprogram.** เอกลักษณ์จะคงอยู่ในขณะที่โปรแกรมมีการทำงานไปยังส่วนอื่นๆ เช่น pointers ที่ชี้ไปยังข้อมูลของระบบไฟล์หรือดิสก์แต่จะถูกเปลี่ยนแปลงเมื่อแนวทางในการจัดเก็บไฟล์ที่เก็บไว้ในระบบถูกเปลี่ยนแปลง
- **Persistent.** เอกลักษณ์จะไม่เพียงปรากฏอยู่ระหว่างที่โปรแกรมทำงานเท่านั้นแต่จะยังคงอยู่แม้มีการเปลี่ยนโครงสร้างของข้อมูลใหม่ ซึ่งเป็นความคงทนที่ต้องการสำหรับ object-oriented systems

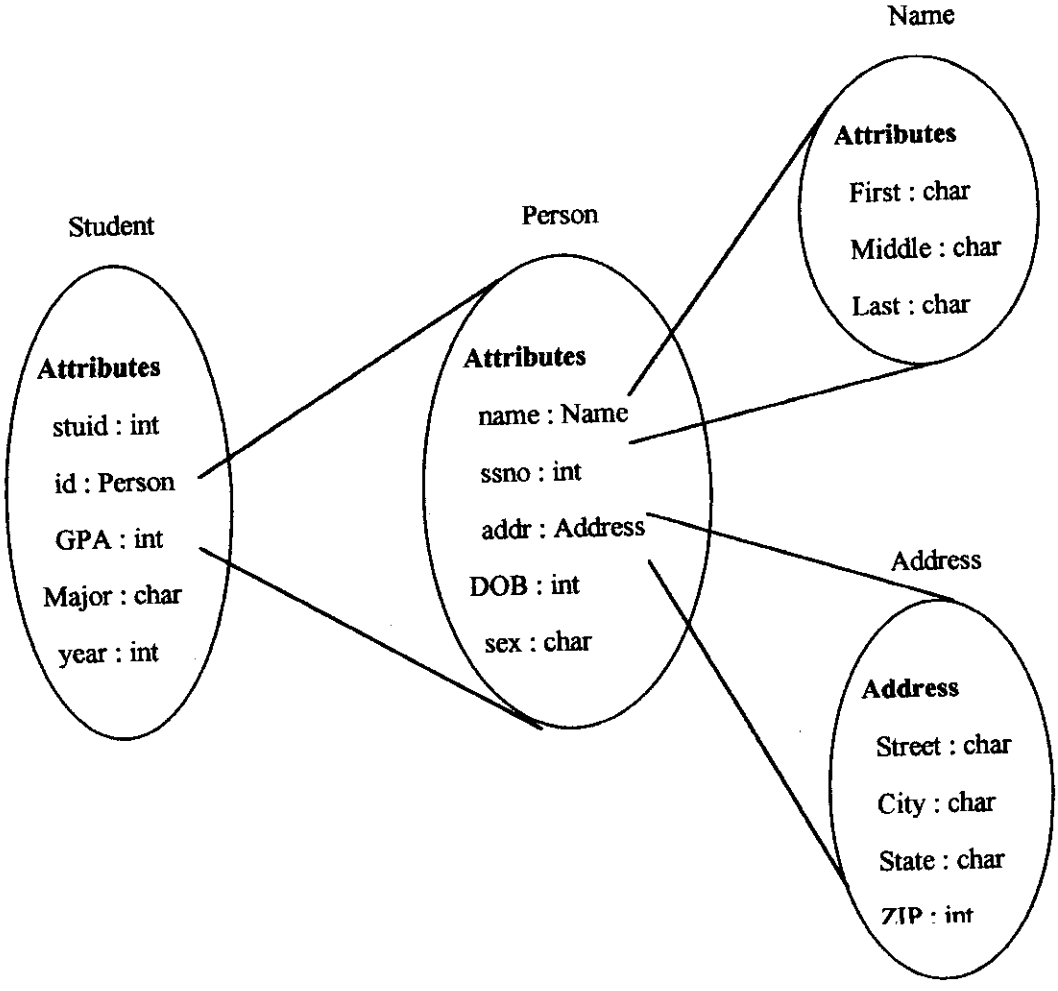
9.5.3 การอ้างอิงความหมาย (Sematic Reference)

การอ้างอิงถึงความหมายของวัตถุเชิงซ้อนและส่วนประกอบของวัตถุมี 2 แบบ แบบแรกเรียกว่า ownership semantics ซึ่งจะประยุกต์ใช้เมื่อมีวัตถุย่อยของวัตถุที่เชิงซ้อนซ่อนอยู่ภายในวัตถุเชิงซ้อนนั้นๆ และวัตถุย่อยนั้นจะถูกพิจารณาว่าเป็นส่วนหนึ่งของวัตถุที่ซับซ้อน ดังนั้นวัตถุย่อยจะถูกลบออกเมื่อวัตถุที่ซับซ้อนนั้นๆ ถูกลบออกไป ส่วนแบบที่สองเรียกว่า reference semantic จะถูก

ประยุกต์ใช้เมื่อส่วนประกอบของวัตถุที่เชิงซ้อนเป็นอิสระโดยตัวเองและถูกตีความเป็นส่วนประกอบของวัตถุเชิงซ้อนเมื่อถึงช่วงเวลาหนึ่งๆ (run time) ซึ่งการลบวัตถุที่เชิงซ้อนจะไม่ทำให้ส่วนประกอบที่เป็นอิสระถูกลบหายไป

9.6 วัตถุเชิงซ้อน (Complex Object)

วัตถุเชิงซ้อนคือวัตถุที่มีวัตถุย่อย (subobjects) เป็นส่วนประกอบโดยวัตถุนั้นอาจมีวัตถุย่อยอื่นๆ ซ้อนอยู่ภายในได้อีก วัตถุเชิงซ้อนสามารถแสดงได้ดังตัวอย่างต่อไปนี้



วัตถุที่เชิงซ้อนแบ่งได้เป็น 2 แบบ ชนิดแรกคือแบบไม่มีโครงสร้างหมายถึงวัตถุที่ต้องการหน่วยจัดเก็บเป็นจำนวนมาก เช่น แบบข้อมูลในการแสดงรูปภาพหรือวัตถุที่เป็นข้อความขนาด

ใหญ่ส่วนชนิดที่สองคือแบบมีโครงสร้างซึ่งเป็นวัตถุที่สร้างจากการนำเอาวัตถุต่างๆ มาประกอบเข้าด้วยกันโดยใช้ตัวสร้างรูปแบบวนซ้ำไปมาในระดับต่างๆ

9.6.1 วัตถุเชิงซ้อนแบบไม่มีโครงสร้างและรูปแบบที่เพิ่มเติมได้ (Unstructured Complex Objects and Type Extensibility)

ตัวอย่างของวัตถุประเภทนี้ได้แก่ รูปภาพ Bitmap และข้อความที่ยาวมากๆ วัตถุพวกนี้รู้จักกันในชื่อของ binary large objects หรือ BLOBs วัตถุเหล่านี้ไม่มีรูปแบบในทางความหมายเพราะ DBMS ไม่ทราบถึงโครงสร้างของวัตถุว่าเป็นอย่างไร มีเพียงโปรแกรมประยุกต์ที่เรียกใช้วัตถุเหล่านี้เท่านั้นที่สามารถตีความหมายของวัตถุได้ อย่างเช่น โปรแกรมประยุกต์อาจมีหน้าที่สำหรับแสดงรูปภาพหรือค้นหาคำหลักในข้อความยาวๆ ได้ วัตถุเหล่านั้นถูกมองว่ามีความซับซ้อนเพราะมีความต้องการเนื้อที่ในการจัดเก็บเป็นจำนวนมากและไม่ใช่ส่วนหนึ่งของตัวแบบข้อมูลมาตรฐานที่กำหนดโดย DBMS ปกติ

ซอฟต์แวร์ DBMS ไม่มีความสามารถที่จะประมวลผลการเลือกและการทำงานที่ขึ้นอยู่กับมูลค่าของวัตถุเหล่านี้ นอกเสียจากว่าโปรแกรมประยุกต์จะให้รหัสในการทำงานเปรียบเทียบซึ่งจำเป็นสำหรับการเลือก, เปรียบเทียบและแสดงวัตถุที่แน่นอนใน OODBMS สิ่งนี้สามารถสำเร็จได้โดยการกำหนดรูปแบบโครงสร้างเชิงนามธรรมแบบใหม่สำหรับวัตถุที่ไม่สามารถตีความได้และจึงให้การดำเนินการสำหรับเลือกเปรียบเทียบและแสดงวัตถุต่างๆ

เนื่องจาก OODBMS อนุญาตให้ผู้ใช้กำหนดรูปแบบชนิดใหม่ได้ และรูปแบบประกอบทั้งโครงสร้างและการทำงาน เราจึงมอง OODBMS ว่าเป็นระบบที่มีความสามารถในการขยายเพิ่มเติมรูปแบบได้ OODBMSs ส่วนใหญ่จะกำหนดให้วัตถุแบบไม่มีโครงสร้างเป็น character string หรือ bit string เพื่อให้สามารถส่งไปยังโปรแกรมประยุกต์เพื่อตีความ

9.6.2 วัตถุเชิงซ้อนแบบมีโครงสร้าง (Structured Complex Objects)

วัตถุเชิงซ้อนแบบมีโครงสร้างต่างจากแบบไม่มีโครงสร้างตรงที่โครงสร้างของวัตถุจะถูกกำหนดโดยใช้โปรแกรมประยุกต์ของตัวสร้างรูปแบบที่มีอยู่ใน OODBMS โดยวนซ้ำกัน ดังนั้นโครงสร้างของวัตถุจึงเป็นที่เข้าใจโดย OODBMS

9.7 แบบรวบรวมวัตถุ (Collection Types)

Collection Types คือแบบชนิดในการรวบรวมและจัดชนิดของวัตถุรวมถึงรูปแบบความเกี่ยวเนื่องระหว่างวัตถุ สำหรับกลุ่มของวัตถุที่ถูกรวบรวมโดย collection type จะเรียกว่า collection ซึ่ง collection สามารถกำหนด properties และ operations บนกลุ่มของตัวเองได้ ตัวอย่างเช่น เราใช้ type set เป็น collection type และเราใช้ type นี้ไปยัง group ของ student instances และ supplier instances ผลที่ได้คือเรามี collections สองกลุ่มที่แตกต่างกันคือ เซตของ students และ set ของ supplier ซึ่ง collection แต่ละกลุ่มจะมี embedded operations ที่กำหนดเพิ่มเติมได้ เช่น total_student() หรือ count_supplier() และ common operations ที่ได้รับการสืบทอดมาจาก supertype (จากตัวอย่างคือ set) เช่น union, intersection, difference

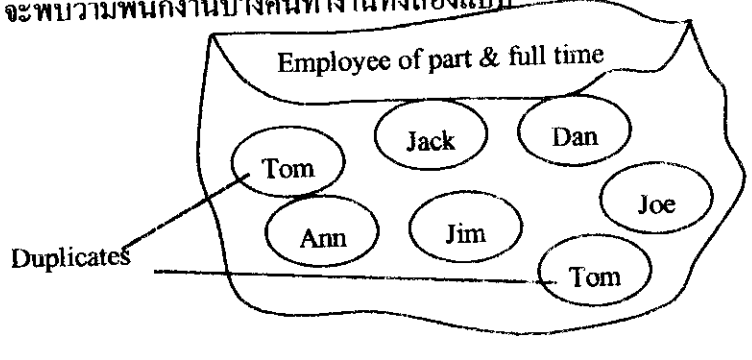
Collection types ที่ Object-oriented databases ให้การสนับสนุนคือ set, list, bag และ array โดยที่แต่ละชนิดมีลักษณะดังนี้

- Set. คือ collection type ที่มีลักษณะเป็น unordered collection (นั่นคือวัตถุที่จัดเก็บจะไม่มีการเรียงลำดับ) และไม่อนุญาตให้มีสมาชิกซ้ำกัน โดย operations ที่มีได้แก่ union, intersection, difference, cony, subset determination, proper subset determination, superset determination และ proper superset determination ตัวอย่างของการใช้ operation เหล่านี้เช่น การนำสองเซตของ employees union กันแล้วได้ผลลัพธ์เป็นหนึ่งเซต employees ซึ่งทำให้ไม่มีการเก็บข้อมูลที่ซ้อนกัน

SET { (S1), , ..., (Sn) }

Set collection type

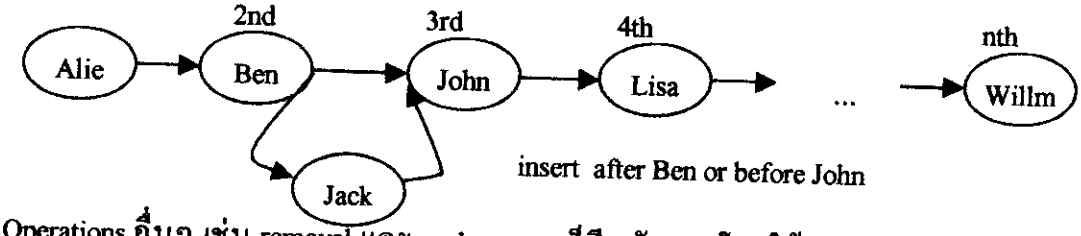
- Bag. คือ unordered collection type ซึ่งอนุญาติให้เก็บสมาชิกซ้ำกันได้ ซึ่ง bag สนับสนุน union, intersection และ difference operators บน compatibles sets (เซตที่มีสมาชิกประเภทเดียวกัน) ตัวอย่างเช่นถ้าเรารวมพนักงานประเภท part-time และ full-time บางครั้งจะพบว่ามีพนักงานบางคนทำงานทั้งสองแบบ



Bag collection type

- List. คือ ordered group ของ objects ซึ่งอนุญาติให้มีการเก็บข้อมูลซ้ำกันได้ ซึ่งลำดับของวัตถุภายใน list จะถูกกำหนดโดยลำดับของการใส่ไปยัง collection ไม่ใช้ index หรือการ sort แต่จะเรียงโดย programmer ซึ่งใช้ operations insertion, removal และ replacment objects ใน list ซึ่งตำแหน่งของ list จะเป็นจุดบอกให้ operations ทำงาน ตัวอย่างเช่นในรูปข้างล่าง หาก new objects ถูกใส่ไปใน list หลัง Ben แต่ก่อน John เราระบุโดย

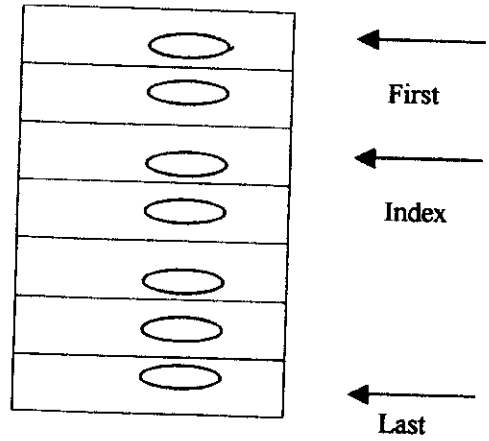
  Insert_element_after(child,2)



Operations อื่นๆ เช่น removal และ replacement ก็มีหลักการโดยใช้ relative position (เช่น first, last, second, etc.) ในทำงานเช่นเดียว

- Array. คือ collection type ที่มี one dimension ซึ่งความยาวผันแปรได้ตามจำนวนของสมาชิก array collection type สามารถให้ objects ถูกจัดเรียงได้ถ้ามี objects เหล่านั้นมีอยู่ใน indexed list Operations ที่มีได้แก่ insert, remove, replace หรือ retieve objects จากตำแหน่งที่ระบุใน array collection

List collection type



Array collection type

9.8 ภาษาเชิงวัตถุ (Object-Oriented Languages)

ขณะนี้เราได้ศึกษาถึงหลักการเบื้องต้นของ object orientation ในระดับของ abstract level ในการปฏิบัติในด้านระบบฐานข้อมูล หลักการเหล่านี้จะต้องแสดงโดยบางภาษาซึ่งมี 2 แนวทางคือ

- แนวคิดของ object orientation จะต้องถูกใช้เป็นเพียงเครื่องมือในการออกแบบ ซึ่งในทางปฏิบัติจะต้องแปลงไปสู่ฐานข้อมูลในระบบอื่นเช่น relational database เป็นต้น
- แนวคิดของ object orientation ถูกรวมเข้าไปกับภาษาหนึ่งที่ใช้ในการจัดการข้อมูลด้วย แนวทางนี้มีหลายภาษาที่เป็นไปได้ในการรวมเข้ากับแนวคิดของ object orientation.
  - แนวทางหนึ่งคือการขยาย data-manipulation language อย่างเช่น SQL โดยการเพิ่ม complex types และ object-orientation ระบบซึ่งให้ object-oriented extensions แก่ relational systems เรียกว่า object-relational systems ซึ่งเป็นหัวข้อที่จะพูดถึงในบทต่อไป
  - อีกทางเลือกหนึ่งคือเลือก object-oriented programming language ที่มีอยู่แล้วเพิ่มความสามารถในการจัดการด้านฐานข้อมูลเข้า ซึ่งภาษาเหล่านี้จะถูกเรียกว่า **persistent programming languages**

9.9 Persistent Programming Languages

Database languages ซึ่งต่างจาก programming language แบบเดิมตรงที่เพิ่มการจัดการกับข้อมูลที่เป็นแบบทนทาน (persistent) (นั่นคือข้อมูลจะยังคงปรากฏอยู่ถึงแม้ว่าโปรแกรมที่สร้างมันได้จบการทำงานไปแล้ว) ตัวอย่างของ persistent data ได้แก่ relation ในฐานข้อมูลและ tuples ในหนึ่ง relation เมื่อเปรียบเทียบแล้ว traditional programming languages มี persistent data ที่จัดการโดยตรงคือ files

Persistent programming language คือ programming language ที่เพิ่มความสามารถในการจัดการกับ persistent data ได้โดยตรง แทนที่จะต้องเขียน embedded SQL ใน code ของภาษาโปรแกรมแบบก่อน ๆ ซึ่งข้อแตกต่างที่เห็นได้ชัดคือ

1. สำหรับ embedded language นั้น type system ของ host language มักจะต่างไปจาก type system ของ data manipulation language ดังนั้นหน้าที่ของ programmer คือจะต้องรับผิด

ชอบในการแปลง type ระหว่าง host language และ data manipulation language ซึ่งการกำหนดให้ programmer รับภาระในส่วนนี้มีข้อเสียหลายอย่างคือ

- สำหรับการแปลง tuples และ objects ทำงานอยู่ object-oriented type systems ดังนั้นจึงมีโอกาสเกิดข้อผิดพลาดที่ไม่สามารถตรวจสอบได้สูงและการแปลงระหว่าง object-oriented format และ relational format จะต้องใช้ code จำนวนมากสำหรับโปรแกรมประยุกต์

  ในทางกลับกัน persistent programming language จะรวมเอา query language เข้าไว้กับ host language ซึ่งทั้งสองต่างใช้ type system ร่วมกัน ดังนั้น objects จึงสามารถสร้างและจัดเก็บในฐานข้อมูลได้โดยไม่ต้องมีการแปลงชนิดหรือรูปแบบ

2. programmer ที่ใช้ embedded query language จะต้องรับผิดชอบในการเขียน explicit code เพื่อ fetch data จาก ฐานข้อมูล ไปยัง main memory เมื่อไรที่มีการ update เกิดขึ้น programmer ก็จะต้องเขียน code เพื่อทำการจัดเก็บ update data ลง ฐานข้อมูล

Persistent versions ของ programming languages เช่น Pascal ได้ถูกเสนอขึ้นนานแล้วและต่อมาก็คือ C++ และ Smalltalk ก็ได้รับการพิจารณามากกว่า เพราะภาษาเหล่านี้ให้ programmer สามารถจัดการกับข้อมูลได้โดยตรงจาก programming language โดยไม่จำเป็นที่จะต้องใช้ data manipulation language เช่น SQL

อย่างไรก็ตามยัง persistent programming language ยังคงมีข้อจำกัดจำนวนหนึ่งเนื่องจาก programming languages มีพลังมาก ดังนั้นการเขียนโปรแกรมที่ผิดพลาดอาจเป็นอันตรายต่อฐานข้อมูลได้, ความซับซ้อนของภาษาโปรแกรมทำให้ automatic high-level optimization (เช่น การลด disk I/O) ทำได้ยากและการสนับสนุน declarative querying เป็นส่วนสำคัญ applications จำนวนมากแต่ persistent programming languages ในปัจจุบันยังไม่สนับสนุน declarative querying ได้ดีเท่าไรนัก

9.10 Persistent C++ Systems

เมื่อหลายปีที่ผ่านมา มี object-oriented database จำนวนมากปรากฏขึ้นโดยมีพื้นฐานบน persistent extensions ของ C++ แต่ยังคงมีความแตกต่างในเรื่องของ system architecture เพราะส่วนใหญ่ยังคงมี common features ใน term ของ programming language

object-oriented features จำนวนมากของ C++ ช่วยสนับสนุนการจัดการในเรื่องของ

persistence ได้ดีโดยไม่ต้องเปลี่ยนภาษาของตัวมันเอง ยกตัวอย่างเช่น เราสามารถประกาศคลาส ซึ่งเรียกว่า Persisten_Object ด้วย attributes และ methods ที่สนับสนุนในเรื่อง persistence โดยที่ class อื่นๆ สามารถเป็น persistent ได้ ถ้าหากสร้างหนึ่ง subclass จาก class นี้ ได้ (เนื่องจาก inherit ส่วนสนับสนุนสำหรับ persistence มา) อีกทั้ง C++ languages ยังให้ความสามารถในการกำหนดชื่อ ของ standard functions และ operators ซ้ำได้เช่น +, -, pointer dereference operator -> และอื่นๆ โดยขึ้นอยู่กับชนิดของ operand ที่จะถูกนำมาไปใช้ ความสามารถในส่วนนี้เรียกว่า overloading ใช้ เพื่อกำหนด operators ซ้ำในการทำงานกับ persistent objects ได้ตามต้อง

การให้การสนับสนุน persistent โดย class libraries มีข้อดีคือทำให้เกิดการเปลี่ยนแปลงโครงสร้าง ของ C++ น้อยสุดและยังง่ายในการสร้างด้วย อย่างไรก็ตามยังคงมีข้อจำกัดอยู่ คือ programmer จะ ต้องใช้เวลาจำนวนมากในการเขียน program เพื่อจัดการกับ persistent objects และมันไม่ง่ายเลยที่ จะระบุ integrity constraints บน schema หรือให้การสนับสนุนในเรื่องของ declarative querying

9.10.1 The ODMG C++ Object-Definition Language

Object Database Management Group (ODMG) ได้ทำงานเพื่อจัดมาตรฐานของส่วนขยาย C++ และ Smalltalk เพื่อสนับสนุน persistent และบนการกำหนด class libraries เพื่อสนับสนุน persistent ซึ่ง ODMG standard พยายามที่จะขยายโครงสร้างของ C++ ให้น้อยที่สุด แต่จะให้ functionality โดย class libraries ให้มากที่สุด

ODMG C++ extension ประกอบด้วย 2 ส่วน คือ (1) C++ Object Definition Language (C++ ODL) และ (2) C++ Object Manipulation Language (C++ OML) ซึ่ง C++ ODL ขยาย type definition syntax.

ตัวอย่างต่อไปนี้คือ code ที่เขียนขึ้นใน ODMG C++ ODL โดย schema ของสี่ classes ถูก กำหนดไว้ใน code แต่ละ class ถูกกำหนดเป็น subclass ของ Persistent_Object ดังนั้น objects ใน class สามารถทำให้เป็น persistent ได้ Classes Person, Branch และ Account เป็น direct subclasses ของ Persistent_Object ส่วน class Customer คือ subclass ของ Person เพราะฉะนั้นจึงเป็น subclass ทางอ้อมของ Persistent_Object

```
class Person : public Persistent_Object {
public :
        String name;
        String address; };
```

```
class Customer : public Person {
public :
        Date member from;
        Int customer_id;
        Ref<Branch> home_branch;
        Set<Ref<Account>>accounts inverse Account :: owners;
};

class Brahch : public Persistent_Object {
public :
        String name;
        String address;
        Int assets;
};
class Account : public Persistent_Object {
private :
        int balance;
public :
        int number;
        Set<Ref<Customer>>owners inverse Custormer :: accounts;
        int find_balance();
        int update_balance(int delta);
};
```

Example of ODMG C++ Object Definition Language

C++ ไม่ได้ให้การสนับสนุนแนวคิดของ messages โดยตรง แต่ methods กลับถูกเรียกโดยตรงในลักษณะของ procedure calls แทน keyword **private** แสดงถึง attributes หรือ methods ต่อไปนี้จะมองเห็นได้เฉพาะ methods ภายใน class ส่วน keyword **public** แสดงถึงว่า attributes หรือ methods นั้นสามารถเห็นได้ code ของ class อื่นๆ

Attribute definitions ของ Type int, String และ Date เป็นส่วนหนึ่งของ Standard C++ syntax สำหรับ Type Ref<Branch> คือ *reference* (หรือ persistent pointer) ไปยัง object ของ type Branch Type Set<Ref<Account>> คือ set ของ persistent pointers ไปยัง objects ของ type Account เราใช้สัญลักษณ์ **inverse** เพื่อระบุถึง referential integrity constraints พิจารณาจาก Customer object การระบุ **inverse** Account :: owners สำหรับ attribute accounts หมายถึงว่า สำหรับแต่ละ account object ที่ถูกอ้างอิงใน account set ของ Customer object, field owner ของ account object จะต้องมี reference ย้อนกลับไปยัง Customer object.

Classes Ref<T> และ Set<T> คือ *template* classes ซึ่งถูกกำหนดใน ODMG standard class เหล่านี้ถูกกำหนดในลักษณะของ type-independent (ไม่ยึดติดกับชนิด) และพร้อมรับ type ต่างๆ ตามต้องการ ความต้องการของ user (เช่น Ref<Account>) สำหรับ class Set<T> เป็น template ที่มีการทำงานเช่นแบบเซต เช่น insert_element และ add_element

การประกาศ class Account แสดงถึง encapsulation features ของ C++ attributes balance ของ Account ถูกกำหนดเป็น private ซึ่งหมายถึงว่าไม่มี functions อื่น นอกจาก methods ของ class สามารถอ่านหรือเขียนมันได้ class นี้มี 2 methods คือ find_balance() และ update_balance(int delta) methods เหล่านี้สามารถอ่านหรือเขียน balance attribute ได้

9.10.2 The ODMG C++ Object Manipulation Language

ตัวอย่างของ code ต่อไปนี้คือตัวอย่างของ ODMG C++ Object Manipulation Language โดยขั้นแรก ฐานข้อมูล จะถูกเปิดขึ้นและ เริ่มต้นทำงาน transaction

```
int create_account_owner(String name, String address) {
        Database *bank db;
Bank_db = Database::open("Bank-DB");
        Transaction Trans;
Trans.begin();
        Ref<Account>account = new(bank_db) Account;
Ref<Customer> cust = new(bank_db) Customer;
cust -> name – name;
        cust -> address = address;
cust -> accounts.insert_element(account);
```

```
        cust -> ownerrs.insert_element(cust);
        ... Code to initialize customer_id, account number etc.
        trans.comit();
}
```

transaction คือลำดับของขั้นตอน ซึ่งกำหนดโดยการเรียก begin และการเรียก commit หรือ abort ของ transaction ลำดับขั้นตอนของการเรียก transaction จะถูกจัดการเป็น atomic unit เพื่อประกันว่าทุกลำดับขั้นตอนได้ทำงานอย่างสมบูรณ์หรือถ้าขั้นตอนใดขั้นตอนหนึ่งไม่สามารถทำงานได้ด้วยเหตุผลใดก็ตาม ผลจากการทำงานของขั้นตอนทั้งหมดที่ได้ทำมาแล้วจะถูกยกเลิกซึ่งไม่ทำให้เกิดผลต่อ ฐานข้อมูล ในกรณีที่ทำงานทั้งหมดเป็นผลสำเร็จ transaction จะถูกบันทึกผล (commit) ถ้ามีความผิดพลาดในขั้นตอนใดขั้นตอนหนึ่ง การทำงานของ transaction นั้นก็จะถูกยกเลิก (abort)

ลำดับถัดมา account และ owner object จะถูกสร้างและตั้งต้นเป็นส่วนหนึ่งของ transaction ในท้ายสุด transaction จะถูก commit สำหรับ class Persitent_Object ได้สร้างหลาย methods ประกอบด้วย persistent version ของ C++ memory-allocation operator **new** ซึ่งถูกใช้ใน code ตัวอย่าง new operator ในที่นี้ใช้สำหรับการสร้าง object ใน ฐานข้อมูล ที่ระบุ (ไม่ใช่ใน memory) เราใช้ method insert_element ของ template class SET<> เพื่อใส่ customer และ account references ในเซตที่เหมาะ สมหลังจากสร้าง customer และ account objects ถ้าท การ insert ถูกกระทำเพียงบางส่วน (ทำเพียงแค่หนึ่งในสอง) ความผิดพลาดของ referential-integrity จะเกิดขึ้นเมื่อมีการทำ transaction commit ดังนั้น transaction จะถูกยกเลิก

ตัวอย่างข้างต้นยังไม่สมบูรณ์เนื่องจากหลังจากจบ transaction เราจะไม่สามารถย้อนกลับไปยัง customer และ account objects ที่เพิ่มถูกสร้างได้ แต่เราสามารถแก้ปัญหานี้ได้โดยเพิ่มการอ้างอิง (references) ไปยัง customer และ account objects ใน pesistent sets ซึ่งได้รวม customer และ account objects ไว้ทั้งหมด เราสามารถตั้งชื่อและ look up เซตเหล่านี้ได้โดยชื่อที่ตั้งให้ หรือเราสามารถรวมเซตเหล่านี้เข้าไว้ภายใน class Customer และ Account โดยการเพิ่มหนึ่ง attribute ของการประกาศซึ่งทำได้โดย statement ต่อไปนี้

```
static Ref<Set<Ref<Customer>>> all_customer;
```

ไปยัง Customer class และ Account class โดย collection จะถูกตั้งต้นโดยการจองพื้นที่ใน ฐานข้อมูล และกำหนดให้เป็น empty เราสามรถใส่ customer object ลงใน set ดังกล่าวโดยใช้ statement ที่

อยู่ในรูป

```
Customer::all_customers.insert_element(cust);
```

Constructor ของ class หนึ่งคือ method พิเศษซึ่งทำหน้าที่ในการตั้งต้นวัตถุเมื่อวัตถุเหล่านั้นถูกสร้างขึ้น และจะถูกเรียกโดยอัตโนมัติเมื่อ new operator ถูก execute ในทำนองเดียวกัน destuctor ของหนึ่ง class คือ method พิเศษที่ถูกเรียกเมื่อ objects ของ class นั้นๆ ถูกลบด้วยการเพิ่ม statement ดังกล่าวไปยัง constructors ของ class และการเพิ่ม delete_element statement ไปยัง destructor ของ class, programmer สามารถมั่นใจได้ว่า collection Customer::all_customers จะถูกดูแลอย่างถูกต้องในบาง persistent C++ extensions, class extents (ซึ่งก็คือ collections ซึ่งรวม persistent objects ของหนึ่ง class ไว้ทั้งหมด) จะถูกสร้างและบำรุงโดยอัตโนมัติสำหรับทุกๆ class ที่สามารถมี persistent objects ได้ ดังนั้น programmer ไม่ต้องเขียน code เพื่อ insert หรือ delete objects จาก class extents

เราสามารถทำซ้ำ collection ของการ reference ได้โดยใช้ *iterator* ดังในตัวอย่างต่อไปนี้

```
int print_customers() {
        Database *bank db;
        bank_db = Database::open("Bank-DB");
        Transaction Trans;
        Trans.begin();

        Iterator<Ref<Customer>> iter = Customer::all_customers.creat_iterator();
        Ref<Customer> p;
        while(iter.next(p)) {
                print_cust(p);
        }
        Trans.commit();
}
```

จากตัวอย่างเราสร้าง iterator โดยใช้ method create_iterator() ซึ่งมีให้โดย class Collection และโดย class ของ subclass class นี้ ตัวอย่างเช่น Set เราใช้ method next() ซึ่งได้จาก iterator เพื่อก้าวไปยัง

elements ต่างๆ ที่มีอยู่ใน collection ของ customers สำหรับแต่ละ customer method print cust (ซึ่งเราสมมุติว่าได้ถูกกำหนดไว้ในตำแหน่งอื่น) จะถูกเรียกเพื่อแสดง customer

ในปัจจุบัน Object-Oriented Database Systems มี query language ซึ่งมีลักษณะคล้ายกับ SOL. ใน relational database เรียกว่า OQL ซึ่งผู้ที่กำหนดมาตรฐานนี้คือ object data management group (ODMG) สำหรับรายละเอียดหาเพิ่มเติมที่ *Paul* ในบรรณานุกรม

9.11 แนวคิดเชิงวัตถุอื่นๆ (Other Object-Oriented Concepts)

ในหัวข้อนี้เราจะพูดถึงลักษณะต่างๆ ที่สำคัญของ Object-oriented Database Systems ซึ่งมีดังนี้

- **การซ่อนสารสนเทศ (Encapsulation)**

Encapsulation คือ การซ่อนรายละเอียด code จากผู้ใช้งานเพื่อป้องกันการแก้ไข code และความเป็นมาตรฐานในการติดต่อกับระบบ ตัวอย่างเช่น โครงสร้างวัตถุหนึ่งๆ จะถูกแบ่งออกเป็นลักษณะที่เห็นได้ (visible attributes) คือสามารถเข้าถึงได้จาก method ภายนอกส่วนลักษณะที่ถูกซ่อน (hidden attribute) จะเป็นส่วนที่ต้องเข้าถึงโดย method ของวัตถุนั้นๆ ที่ได้กำหนดไว้แล้ว

อีกตัวอย่างหนึ่งคือ class จะมีการกำหนดชื่อของ method และ interfaces ไว้ในตอนที่กำหนด class ส่วน code ของ methods จะต้องถูกกำหนดไว้ในส่วนอื่น

- **การมีหลายรูปแบบ (Polymorphism or Operator Overloading)**

แนวคิดนี้กำหนดให้ตัวดำเนินการซึ่งมีชื่อหรือสัญลักษณ์เดียวกันสามารถมีขอบเขตในการทำให้เกิดผลตั้งแต่ 2 แบบขึ้นไป ขึ้นอยู่กับชนิดของวัตถุที่จะถูกดำเนินการ ตัวอย่างเช่น ภาษาโปรแกรมบางภาษาเครื่องหมาย + สามารถใช้ได้กับข้อมูลประเภทจำนวนเต็มเพื่อทำการหาผลบวกหรือประเภทเซตเพื่อนำเซตมารวมกันได้

- **การแปลและองค์ประกอบ (Versions and Configurations)**

การแปล (versions) คือรุ่นของการตีความสำหรับวัตถุต่างๆ ที่อยู่ในฐานข้อมูลซึ่งโปรแกรมประยุกต์ต่างๆ ที่ใช้ OO systems มักมีความต้องการ version ของวัตถุต่างๆ ที่สอดคล้องกัน

กิจกรรมบำรุงรักษาจะถูกประยุกต์เข้ากับระบบซอฟท์แวร์เพื่อความก้าวหน้าของระบบ ซึ่งโดยปกติจะทำให้การออกแบบและการทำให้เกิดผลของ module บางส่วนเปลี่ยนไป ถ้าระบบ

ได้ทำงานอยู่แล้วแต่มีความจำเป็นที่จะต้องเปลี่ยนแปลง modules กลุ่มหนึ่ง ผู้ออกแบบควรที่จะสร้างการแปลใหม่ (new version) ของแต่ละ module เหล่านั้น เพื่อที่จะทำให้การเปลี่ยนแปลงเกิดผลขึ้นโดยที่การแปลรุ่นเดิมควรที่จะเก็บไว้จนกว่าการแปล รุ่นใหม่ได้ถูกทดสอบและปรับปรุงจนเป็นที่เรียบร้อยก่อน แล้วจึงทำการแปลใหม่แทนที่การแปลเดิม ซึ่งวัตถุหนึ่งๆ อาจมีการแปลได้มากกว่า 2 การแปลขึ้นไป OODBMS ควรที่จะสามารถจัดการกับการแปลหลายๆ รุ่นของวัตถุเดียวกันได้

องค์ประกอบ (configuration) ของวัตถุเชิงซ้อนคือการรวมกันของลำดับการแปลที่ถูกต้องและสอดคล้องกันของแต่ละ module การแปลใหม่หรือองค์ประกอบของวัตถุที่เชิงซ้อนไม่ได้รวมถึงการแปลครั้งใหม่ของทุกๆ module ดังนั้น version ของ module เก่าอาจเป็นองค์ประกอบได้มากกว่า 1 องค์ประกอบของวัตถุเชิงซ้อน ซึ่งองค์ประกอบก็คือการรวมกันของ versions ของวัตถุต่างๆ ที่รวมกันเป็นวัตถุเชิงซ้อน

9.12.ข้อเปรียบเทียบระหว่าง Relational Database และ Object-oriented Databases (Comparison of Relational Databases and Object-oriented Databases)

<u>ฐานข้อมูลเชิงสัมพันธ์</u> (Relational Databases)

- จุดประสงค์หลัก (Primary Goal)

ความเป็นอิสระของข้อมูล ข้อมูลสามารถที่จะจัดกลุ่มทางกายภาพใหม่ได้โดยไม่มีผลว่าจะเรียกใช้อย่างไร มีผลกระทบว่าจะถูกใช้อย่างไร

- ข้อมูลเพียงอย่างเดียว (Data Only)

ฐานข้อมูลจะเก็บเฉพาะข้อมูลเท่านั้น ไม่เก็บวิธีการทำงาน

<u>ฐานข้อมูลเชิงวัตถุ</u> (Object-Oriented Database)

- จุดประสงค์หลัก (Primary Goal)

ซ่อนรายละเอียดของการทำให้เกิดผล (encapsulation) และความเป็นอิสระของคลาส (class) สามารถที่จะจัดเรียงรูปแบบใหม่โดยไม่มีผลว่าจะถูกใช้อย่างไร

- ข้อมูลและวิธีการทำงาน (Data Plus Methods)

ฐานข้อมูลจะเก็บข้อมูลและวิธีการทำงาน (method) ไว้ร่วมกัน

- ข้อมูลมีการใช้ร่วมกัน (Data Sharing)

ข้อมูลสามารถที่จะเข้าถึงโดยกระบวนการ (process) ต่างๆ ข้อมูลออกแบบมาเพื่อการใช้งานประเภทต่างๆ

- ข้อมูลอยู่นิ่ง (Passive Data)

ข้อมูลจะอยู่นิ่งมีเพียงการทำงานจำกัดซึ่งอาจจะเกิดขึ้นโดยอัตโนมัติเมื่อข้อมูลถูกใช้

- การเปลี่ยนแปลงอย่างต่อเนื่อง (Constant Change)

การดำเนินงาน (process) จะทำให้ข้อมูลมีการเปลี่ยนแปลงอย่างต่อเนื่อง

- ความเรียบง่าย (Simplicity)

ผู้ใช้จะได้รับข้อมูลในรูปของแถวลำดับตามแนวตั้ง,แนวนอนและตาราง

- ตารางแยกจากกัน (Separate Table)

ตารางแต่ละตารางจะแยกจากกัน การดำเนินการเชื่อมโยง (join) จะถูกใช้ในการอ้างอิงข้อมูลระหว่างตาราง

- ซ่อนรายละเอียดของการทำให้เกิดผล (Encapsulation)

ข้อมูลสามารถถูกใช้โดยวิธีการทำงานของคลาสเท่านั้น ข้อมูลถูกออกแบบมาเพื่อใช้กับวิธีการทำงานที่เฉพาะเจาะจงเท่านั้น

- วัตถุมีการทำงาน (Active Object)

วัตถุจะทำงานตามคำร้องขอ (request) จะทำให้วัตถุเรียกวิธีการทำงานมาดำเนินการ

- คลาสและการนำกลับมาใช้ใหม่ (Classes and Reusability)

วัตถุออกแบบมาเพื่อการนำกลับมาใช้ใหม่และการเปลี่ยนแปลงไม่บ่อยนัก

- ความซับซ้อน (Complexity)

โครงสร้างของข้อมูลค่อนข้างซับซ้อน แต่ผู้ใช้ไม่ต้องกังวลถึงสิ่งเหล่านี้เพราะมีการซ่อนรายละเอียดของการทำให้เกิดผล

- ข้อมูลเชื่อโยงระหว่างกัน (Interlinked Data)

ข้อมูลสามารถที่จะเชื่อมโยงระหว่างกันได้โดย method ของ class จึงทำให้ประสิทธิภาพดีขึ้น

- ไม่มีข้อมูลซ้ำซ้อนกัน (Nonredundant Data)

กระบวนการทำให้เป็นมาตรฐาน (Normalization) ของข้อมูลจะทำให้ลดความซ้ำซ้อนในส่วนของข้อมูลได้ แต่ยังไม่ช่วยลดความซ้ำซ้อนในการพัฒนาโปรแกรมประยุกต์

- ภาษาข้อมูล (Data Language)

ภาษา SQL จะเป็นภาษาที่ใช้ในการจัดการกับตาราง (Tables)

-ประสิทธิภาพ (Performance)

ประสิทธิภาพจะขึ้นอยู่กับโครงสร้างของข้อมูลที่มีระดับความซับซ้อนสูง

- ตัวแบบทางความคิดแตกต่างกัน (Different Conceptual Model)

ตัวแบบของโครงสร้างข้อมูลและการเข้าถึงถูกแสดงโดยตาราง (Table) และการดำเนินการเชื่อมโยง (Join Operation) ตามลำดับซึ่งเป็นสิ่งที่แตกต่างกันทั้งในทางวิเคราะห์, ออกแบบและเขียนโปรแกรม การออกแบบจะต้องถูกแปลงให้อยู่ในรูปของตารางสัมพันธ์และการเข้าถึงในแบบของ SQL

-ไม่มีวิธีการทำงานที่ซ้ำซ้อนกัน (Nonredundant Method)

การสืบทอดคุณสมบัติ (Inheritance) จะช่วยลดความซ้ำซ้อนของวิธีการทำงานได้ และการใช้คลาสซ้ำจะช่วยลดเวลาในการพัฒนาโปรแกรมประยุกต์ได้ ปัญหาความซ้ำซ้อนของข้อมูลได้ถูกแก้ไขโดยการซ่อนวิธีการทำให้เกิดผล (Encapsulation)

- คำร้องเชิงวัตถุ (Object-Oriented Requests)

คำร้องขอทำให้เกิดการทำงานของmethodวิธีการทำงานที่หลากหลายสามารถใช้ได้

- การเล็งผลที่ดีที่สุดของคลาส (Class Optimization)

ข้อมูลของหนึ่งวัตถุสามารถที่จะเชื่อมโยงระหว่างกันได้ดังนั้นประสิทธิภาพจึงดีกว่า

- ตัวแบบซึ่งลงรอยกันทางความคิด (Consistent Conceptual Model)

ระเบียบวิธีซึ่งใช้ในการวิเคราะห์, ออกแบบและเขียนโปรแกรมรวมทั้งวิธีการเข้าถึงข้อมูลและโครงสร้างข้อมูลเป็นวิธีเดียวกัน แนวคิดในการปฏิบัติได้ถูกแสดงโดยตรงด้วยคลาสในฐานข้อมูลเชิงวัตถุ ในการสร้างโปรแกรมประยุกต์และโครงสร้างข้อมูลที่ซับซ้อนนั้น ระเบียบวิธีดังกล่าวจะเป็นส่วนที่ช่วยลดเวลาและค่าใช้จ่ายในการพัฒนาโปรแกรมประยุกต์

Summary

จุดประสงค์หลักของเทคโนโลยีฐานข้อมูลแบบเดิมคือความเป็นอิสระของข้อมูล โครงสร้างข้อมูลจึงควรจะเป็นอิสระจากการทำงานซึ่งเรียกใช้ข้อมูลนั้น ข้อมูลสามารถที่จะนำไปใช้ในลักษณะต่างๆ ได้ตามที่ผู้ใช้ต้องการในทางกลับกันจุดประสงค์หลักของเทคโนโลยีฐานข้อมูลเชิงวัตถุคือการซ่อนรายละเอียดของการทำให้เกิดผล (Encapsulation) จากผู้ใช้ การซ่อนรายละเอียดจากผู้ใช้หมายถึง ข้อมูลซึ่งอยู่รวมในวัตถุสามารถถูกเรียกใช้โดยวิธีการทำงานซึ่งเป็นส่วนหนึ่งของคลาสเท่านั้น

Object-oriented class มีจุดมุ่งหมายในการนำกลับมาใช้ใหม่ ดังนั้นจุดประสงค์อีกอย่างหนึ่งของแนวคิดเชิงวัตถุคือการนำกลับมาใช้อย่างสูงสุด ด้วยเหตุผลนี้คลาสจึงไม่ควรมี bug และจะถูกแก้ไขต่อเมื่อจำเป็นเท่านั้น แนวคิดฐานข้อมูลเชิงสัมพันธ์ถูกคิดค้นเพื่อสนับสนุนวิธีการทำงาน (process) ซึ่งจะมีการแก้ไขอย่างไม่มีการสิ้นสุด ดังนั้นความเป็นอิสระของข้อมูลจึงเป็นสิ่งสำคัญ ส่วนแนวคิดฐานข้อมูลเชิงวัตถุสนับสนุนคลาส ซึ่งมีการเปลี่ยนแปลงน้อยครั้ง การเปลี่ยนเกิดจากการเชื่อมโยงกันระหว่างคลาสในแบบต่างๆ โครงสร้างข้อมูล ในฐานข้อมูลเชิงวัตถุควรที่จะทำให้เหมาะสมเพื่อสนับสนุนคลาสซึ่งข้อมูลเหล่านี้ถูกซ่อนอยู่

จุดประสงค์หลักของฐานข้อมูลเชิงวัตถุและฐานข้อมูลเชิงสัมพันธ์มีความแตกต่างกัน จุดประสงค์หลักของฐานข้อมูลเชิงสัมพันธ์คือทำให้ข้อมูลเป็นอิสระจากโปรแกรมประยุกต์ที่เรียกใช้งานซึ่งจะทำให้เหลือข้อมูลเป็นจำนวนมากไว้สำหรับให้โปรแกรมประยุกต์อื่นๆ เรียกใช้ เรามีความต้องการฐานข้อมูลซึ่งสามารถจะนำงานไปใช้ได้ในลักษณะต่างๆ ซึ่งไม่สามารถบอกถึงงานที่จะทำล่วงหน้าในอนาคตได้ในทางกลับกันฐานข้อมูลเชิงวัตถุจะกำหนดข้อมูลซึ่งสามารถเรียกใช้ได้โดยวิธีการทำงานซึ่งนิยามไว้แล้วเท่านั้น

ตัวอย่างของ commercial OODBMSs ในปัจจุบัน ได้แก่ GemStone, Itasca, O2, Objectivity, Ontos, ObjectStore, Ode, Open-OODB, Orion, Versant และอื่นๆ