CHAPTER 8

Object-Oriented Databases

Exercises

8.2 Answer: An entity is simply a collection of variables or data items. An object is an encapsulation of data as well as the methods (code) to operate on the data. The data members of an object are directly visible only to its methods. The outside world can gain access to the object's data only by passing pre-defined messages to it, and these messages are implemented by the methods.

8.3 Answer:

```
class vehicle {
    int        vehicle-id;
    string     license-number;
    string     manufacturer;
    string     model;
    date       purchase-date;
    string     color;
};

class truck isa vehicle {
    int        cargo-capacity;
};

class sports-car isa vehicle {
    int        horsepower;
    int        renter-age-requirement;
};

class van isa vehicle {
    int         num-passengers;
};

class off-road-vehicle isa vehicle {
    real        ground-clearance;
    string      drivetrain;
};
```

8.7 Answer: Creation, destruction and access will typically be more time consuming and expensive for persistent objects stored in the database, than for transient objects in the transaction's local memory. This is because of the over-heads in preserving transaction semantics, security and integrity. Since a transient object is purely local to the transaction which created it and does not enter the database, all these over-heads are avoided. Thus, in order to provide efficient access to purely local and temporary data, transient objects are provided by persistent programming languages.

8.8 Answer:

a. The schema definitions can be written in two different ways, one of which is a direct translation from the relational schema, while the other uses object-oriented features more directly.

- The first scheme is as follows:

```
class employee : public d_Object {
public:
    d_String person-name;
    d_String street;
    d_String city;
};

class company : public d_Object {
public:
    d_String company-name;
    d_String city;
};

class works : public d_Object {
public:
    d_Ref<employee> person;
    d_Ref<company> company;
    d_Long salary;
};

class manages : public d_Object {
public:
    d_Ref<employee> person;
    d_Ref<employee> manager;
};
```

- The second schema is as follows

```
class employee : public d_Object {
public:
    d_String person-name;
    d_String street;
    d_String city;
    d_Rel_Ref<company, _employees> company;
    d_Ref<employee> manager;
    d_Long salary;
};

class company : public d_Object {
public:
    d_String company-name;
    d_String city;
    d_Rel_Set<employee, _comp> employees;
};

const char _employees[] = ”employees”;
const char _comp[] = ”comp”;
```

b. We present queries for the second schema.

- Find the company with the most employees.

```
d_Ref<company> mostemployees(){
    d_Database emp_db_obj;
    d_Database * emp_db = &emp_db_obj;
    emp_db−>open(”Emp-DB”);
    d_Transaction Trans;
    Trans.begin();
    d_Extent<company> all_comps(emp_db);
    d_Iterator<d_Ref<company>> iter=all_comps.create_iterator();
    d_Iterator<d_Ref<employee>> iter2;
    d_Ref<company> c, maxc;
    d_Ref<employee> e;
    int count;
    int maxcount=0;
```

```
                while(iter.next(c)) {
                        iter2=(c->employees).create_iterator();
                        count=0;
                        while(iter2.next(e)) {
                                count++;
                        }
                        if(maxcount < count) {
                                maxcount=count;
                                maxc=c;
                        }
                }
                Trans.commit();
                return maxc;
        }
```

- Find the company with the smallest payroll.

```
d_Ref<company> smallestpay(){
        d_Database emp_db_obj;
        d_Database * emp_db = &emp_db_obj;
        emp_db->open("Emp-DB");
        d_Transaction Trans;
        Trans.begin();
        d_Extent<company> all_comps(emp_db);
        d_Iterator<d_Ref<company>> iter=all_comps.create_iterator();
        d_Iterator<d_Ref<employee>> iter2;
        d_Ref<company> c, minc;
        d_Ref<employee> e;
        d_Long sal;
        d_Long minsal=0;
        while(iter.next(c)) {
                iter2=(c->employees).create_iterator();
                sal=0;
                while(iter2.next(e)) {
                        sal+=e->salary;
                }
                if(minsal > sal) {
                        minsal=sal;
                        minc=c;
                }
        }
        Trans.commit();
        return minc;
}
```

- Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

```
d_Set<d_Ref<company>> highersal(){
        d_Database emp_db_obj;
        d_Database * emp_db = &emp_db_obj;
        emp_db−>open("Emp-DB");
        d_Transaction Trans;
        Trans.begin();
        d_Extent<company> all_comps(emp_db);
        d_Iterator<d_Ref<company>> iter=all_comps.create_iterator();
        d_Iterator<d_Ref<employee>> iter2;
        d_Ref<company> c, FBC=all_comps.select(
                "company-name='First Bank Corporation'");
        d_Set<d_Ref<company>> result;
        d_Ref<employee> e;
        int count;
        d_Long avsal=0, avFBCsal=0, sal=0;
        iter2=(FBC−>employees).create_iterator();
        while(iter2.next(e)) {
                count++;
                sal+=e−>salary;
        }
        avFBCsal=sal/count;
        while(iter.next(c)) {
                iter2=(c−>employees).create_iterator();
                sal=0; count=0;
                while(iter2.next(e)) {
                        sal+=e−>salary;
                        count++;
                }
                avsal=sal/count;
                if(avsal > avFBCsal) {
                        result.insert_element(c);
                }
        }
        Trans.commit();
        return result;
}
```

8.10 Answer: To represent ternary relationships, create a class corresponding to the relationship and refer to the entities in this class. For example, to represent the ternary relationship in Figure 2.13, we do the following:

```
class workson : public d_Object {
public:
    d_Ref<employee> emp;
    d_Ref<branch> branch;
    d_Ref<job> job;
};
```

8.12 Answer: If an object is created without any references to it, it can neither be accessed nor deleted via a program. The only way is for the database system to locate and delete such objects by itself. This is called *garbage collection*. One way to do garbage collection is by the method of *mark and sweep*. First, the objects referred to directly by programs are marked. Then references from these objects to other objects are followed, and those referred objects are marked. This procedure is followed repeatedly until no more unmarked objects can be reached by following reference chains from the marked objects. At this point, all these remaining unmarked objects are deleted. This method is correct; we can prove that if no new objects are marked after a round of mark and sweep, the remaining unmarked objects are indeed unreferenced.

8.13 Answer: A database system must provide for such features as transactions, queries (associative retrieval of objects), security, and integrity. A persistent object system may not offer such features.