14.7 Give an example of an object that belongs to the domain of the type `[Name: STRING, Children: {Person}, Cars: {[Make: STRING, Model: STRING, Year: STRING]}]`. Consider a supertype `[Name: STRING, Cars: {[Make: STRING, Model: STRING]}]` of that type. Show how one can obtain an object of the second type from the object of the first type, which you constructed earlier.

Solution: An object of the first type can be

```
(#23, [Name: John,
       Children: {#1, #2},
       Cars: {[Make: Dodge, Model: Spirit, Year: 2000],
              [Make: Honda, Model: Accord, Year: 2004]}])
```

The corresponding object in the supertype is

```
(#23, [Name: John,
       Cars: {[Make: Dodge, Model: Spirit],
              [Make: Honda, Model: Accord]}])
```

14.8 Consider the following type, which describes projects: `[Name: STRING, Members: {Person}, Address: [Building: INTEGER, Room: INTEGER]]`. Use SQL:1999/2003 to specify the UDT corresponding to this type.

Solution:
We assume that Person is a pre-defined type. Let us assume that the type is called ProjectType.

```
CREATE TYPE ProjectType AS (
        Name CHAR(20),
        Members REF(Person) MULTISET,
        Address ROW(Building INTEGER, Room INTEGER) );
```

14.9 Use the UDT constructed in Exercise 16.8 to answer the following query: List the names of all projects that have more than five members.

Solution:
Assume that we have a table, Project, of type ProjectType.

```
SELECT P.Name
FROM Project P
WHERE count(P→Id) > 5
```

14.11 Use SQL:2003 (with the MULTISET construct, if necessary) to complete the schema partially defined in Section 14.3. Include UDTs for the following tables: `Person`, `Student`, `Course`, `Professor`, `Teaching`, and `Transcript`. Your solution should follow the object-oriented design methodology. Repeating the statements from Chapters 3 and 4, which use SQL-92, is not acceptable.

Solution:
The objective of this exercise is to have the students use an object-oriented design and not merely repeat the SQL-92 statements from Chapters 3 and 4.

```
CREATE TYPE PersonType AS (
        Name CHAR(20),
        Address ROW(Number INTEGER, Street CHAR(20), ZIP CHAR(5)) );

-- We make transcripts accessible from Student, only
-- because this is how transcript records are usually accessed
-- Note: we are not using references for TranscriptRecordType,
-- because we are not going to gain much by sharing these objects

CREATE TYPE StudentType UNDER PersonType AS (
        Id INTEGER,
        Status CHAR(2),
        Transcript TranscriptRecordType MULTISET );

CREATE TABLE Student OF StudentType;
CREATE TYPE ProfessorType UNDER PersonType AS (
        Id INTEGER,
        DeptId CHAR(4) );

-- Note: we de_ne an extra oid attribute, prof_oid, using REF IS
-- because Teaching references professor objects. Without the
-- explicit oid attribute we will not be able to populate Teaching
-- with tuples that have correct references to Professor objects

CREATE TABLE Professor OF ProfessorType REF IS prof_oid;
CREATE TYPE CourseType AS (
        CrsCode CHAR(6),
        DeptId CHAR(4),
        CrsName CHAR(20),
        Description CHAR(50) );

-- Note: we de_ne an extra oid attribute, course_oid, using REF IS
-- because Teaching and TranscriptRecordType references course
-- objects. Without the explicit oid attribute we will not be able to populate
-- Teaching and Student (which uses TranscriptRecordType)
-- with tuples that have correct references to Course objects

CREATE TABLE Course OF CourseType REF IS course_oid;
CREATE TYPE TranscriptRecordType AS (
        Course REF(CourseType) SCOPE Course,
        Semester CHAR(6),
        Grade CHAR(2) );
```

```
-- We could make teaching records accessible from Professor
-- but they are more often used to print catalogs, which justi_es
-- having them as a separate table
CREATE TABLE Teaching (
        Professor REF(ProfessorType) SCOPE Professor,
        Course REF(CourseType) SCOPE Course,
        Semester CHAR(6),
        Grade CHAR(1) );
```