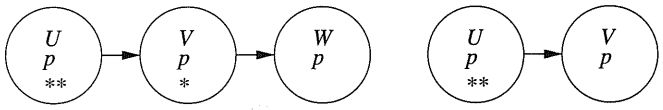


(a) After step (2) (b) After step (3)

Figure 10.5: Revoking a grant option leaves the underlying privilege

### 10.1.7 Exercises for Section 10.1

**Exercise 10.1.1:** Indicate what privileges are needed to execute the following queries. In each case, mention the most specific privileges as well as general privileges that are sufficient.

a) The tuple-based check of Fig. 7.3.

b) The assertion of Example 7.11.

c) The query of Fig. 6.5.

d) The query of Fig. 6.7.

e) The insertion of Fig. 6.15.

f) The deletion of Example 6.37.

g) The update of Example 6.39.

**Exercise 10.1.2:** Show the grant diagrams after steps (4) through (6) of the sequence of actions listed in Fig. 10.6. Assume $A$ is the owner of the relation to which privilege $p$ refers.

| Step | By | Action |
|---|---|---|
| 1 | $A$ | `GRANT` $p$ `TO` $C$ `WITH GRANT OPTION` |
| 2 | $A$ | `GRANT` $p$ `TO` $D$ |
| 3 | $C$ | `GRANT` $p$ `TO` $E$ `WITH GRANT OPTION` |
| 4 | $E$ | `GRANT` $p$ `TO` $B$, $C$, $D$ `WITH GRANT OPTION` |
| 5 | $C$ | `REVOKE` $p$ `FROM` $E$ `CASCADE` |
| 6 | $A$ | `REVOKE` $p$ `FROM` $D$ `CASCADE` |

Figure 10.6: Sequence of actions for Exercise 10.1.2

**Exercise 10.1.3:** Show the grant diagrams after steps (5) and (6) of the sequence of actions listed in Fig. 10.7. Assume $A$ is the owner of the relation to which privilege $p$ refers.

| Step | By | Action |
|---|---|---|
| 1 | $A$ | GRANT $p$ TO $D$, $E$ WITH GRANT OPTION |
| 2 | $E$ | GRANT $p$ TO $B$ WITH GRANT OPTION |
| 3 | $B$ | GRANT $p$ TO $C$ WITH GRANT OPTION |
| 4 | $D$ | GRANT $p$ TO $B$ |
| 5 | $D$ | GRANT $p$ TO $C$ WITH GRANT OPTION |
| 6 | $A$ | REVOKE GRANT OPTION FOR $p$ FROM $E$ CASCADE |

Figure 10.7: Sequence of actions for Exercise 10.1.3

! **Exercise 10.1.4:** Show the final grant diagram after the following steps, assuming $A$ is the owner of the relation to which privilege $p$ refers.

| Step | By | Action |
|---|---|---|
| 1 | $A$ | GRANT $p$ TO $B$ WITH GRANT OPTION |
| 2 | $B$ | GRANT $p$ TO $B$ WITH GRANT OPTION |
| 3 | $A$ | REVOKE $p$ FROM $B$ CASCADE |

## 10.2 Recursion in SQL

The SQL-99 standard includes provision for recursive definitions of queries. Although this feature is not part of the "core" SQL-99 standard that every DBMS is expected to implement, at least one major system — IBM's DB2 — does implement the SQL-99 proposal, which we describe in this section.

### 10.2.1 Defining Recursive Relations in SQL

The WITH statement in SQL allows us to define temporary relations, recursive or not. To define a recursive relation, the relation can be used within the WITH statement itself. A simple form of the WITH statement is:

WITH $R$ AS <definition of $R$> <query involving $R$>

That is, one defines a temporary relation named $R$, and then uses $R$ in some query. The temporary relation is not available outside the query that is part of the WITH statement.

More generally, one can define several relations after the WITH, separating their definitions by commas. Any of these definitions may be recursive. Several defined relations may be mutually recursive; that is, each may be defined in terms of some of the other relations, optionally including itself. However, any relation that is involved in a recursion must be preceded by the keyword RECURSIVE. Thus, a more general form of WITH statement is shown in Fig. 10.8.

**Example 10.8:** Many examples of the use of recursion can be found in a study of paths in a graph. Figure 10.9 shows a graph representing some flights of two

| Round | $P$ | $Q$ |
|---|---|---|
| 1) | {(12), (34)} | {NULL} |
| 2) | {(12), (34), NULL} | {(46)} |
| 3) | {(12), (34), (46)} | {(46)} |
| 4) | {(12), (34), (46)} | {(92)} |
| 5) | {(12), (34), (92)} | {(92)} |
| 6) | {(12), (34), (92)} | {(138)} |

Figure 10.14: Iterative calculation for a nonmonotone aggregation

to be the same as $R$, and $Q$ is {NULL}, since the old, empty value of $P$ is used in line (7).

At the second round, the union of lines (3) through (5) is the set

$$R \cup \{\texttt{NULL}\} = \{(12),\ (34),\ \texttt{NULL}\}$$

so that set becomes the new value of $P$. The old value of $P$ was {(12), (34)}, so on the second round $Q = \{(46)\}$. That is, 46 is the sum of 12 and 34.

At the third round, we get $P = \{(12), (34), (46)\}$ at lines (2) through (5). Using the old value of $P$, {(12), (34), NULL}, $Q$ is defined by lines (6) and (7) to be {(46)} again. Remember that NULL is ignored in a sum.

At the fourth round, $P$ has the same value, $\{(12), (34), (46)\}$, but $Q$ gets the value {(92)}, since 12+34+46=92. Notice that $Q$ has lost the tuple (46), although it gained the tuple (92). That is, adding the tuple (46) to $P$ has caused a tuple (by coincidence the same tuple) to be deleted from $Q$. That behavior is the nonmonotonicity that SQL prohibits in recursive definitions, confirming that the query of Fig. 10.13 is illegal. In general, at the $2i$th round, $P$ will consist of the tuples (12), (34), and $(46i - 46)$, while $Q$ consists only of the tuple $(46i)$. □

### 10.2.3 Exercises for Section 10.2

**Exercise 10.2.1:** The relation

```
Flights(airline, frm, to, departs, arrives)
```

from Example 10.8 has arrival- and departure-time information that we did not consider. Suppose we are interested not only in whether it is possible to reach one city from another, but whether the journey has reasonable connections. That is, when using more than one flight, each flight must arrive at least an hour before the next flight departs. You may assume that no journey takes place over more than one day, so it is not necessary to worry about arrival close to midnight followed by a departure early in the morning.

a) Write the recursion in SQL.

b) Write this recursion in Datalog.

! **Exercise 10.2.2:** In Example 10.8 we used `frm` as an attribute name. Why did we not use the more obvious name `from`?

**Exercise 10.2.3:** Suppose we have a relation

```
SequelOf(movie, sequel)
```

that gives the immediate sequels of a movie, of which there can be more than one. We want to define a recursive relation `FollowOn` whose pairs $(x, y)$ are movies such that $y$ was either a sequel of $x$, a sequel of a sequel, or so on.

a) Write the definition of `FollowOn` as a SQL recursion.

b) Write the definition of `FollowOn` as recursive Datalog rules.

c) Write a recursive SQL query that returns the set of pairs $(x, y)$ such that movie $y$ is a follow-on to movie $x$, but is not a sequel of $x$.

d) Write a recursive SQL query that returns the set of pairs $(x, y)$ meaning that $y$ is a follow-on of $x$, but is neither a sequel nor a sequel of a sequel.

! e) Write a recursive SQL query that returns the set of pairs $(x, y)$ such that movie $y$ is a follow-on of $x$ but $y$ has at most one follow-on.

! f) Write a recursive SQL query that returns the set of movies $x$ that have at least two follow-ons. Note that both could be sequels, rather than one being a sequel and the other a sequel of a sequel.

**Exercise 10.2.4:** Suppose we have a relation

```
Rel(class, rclass, mult)
```

that describes how one ODL class is related to other classes. Specifically, this relation has tuple $(c, d, m)$ if there is a relation from class $c$ to class $d$. This relation is multivalued if $m =$ `'multi'` and it is single-valued if $m =$ `'single'`. It is possible to view `Rel` as defining a graph whose nodes are classes and in which there is an arc from $c$ to $d$ labeled $m$ if and only if $(c, d, m)$ is a tuple of `Rel`. Write a recursive SQL query that produces the set of pairs $(c, d)$ such that:

a) There is a path from class $c$ to class $d$ in the graph described above.

b) There is a path from $c$ to $d$ along which every arc is labeled `multi`.

! c) There is a path from $c$ to $d$ with at least one arc labeled `single`.

d) There is a path from $c$ to $d$ but no path along which all arcs are labeled `multi`.

! e) There is a path from $c$ to $d$ along which arc labels alternate `single` and `multi`.

f) There are paths from $c$ to $d$ and from $d$ to $c$ along which every arc is labeled `multi`.

## 10.3 The Object-Relational Model

The relational model and the object-oriented model typified by ODL are two important points in a spectrum of options that could underlie a DBMS. For an extended period, the relational model was dominant in the commercial DBMS world. Object-oriented DBMS's made limited inroads during the 1990's, but never succeeded in winning significant market share from the vendors of relational DBMS's. Rather, the vendors of relational systems have moved to incorporate many of the ideas found in ODL or other object-oriented-database proposals. As a result, many DBMS products that used to be called "relational" are now called "object-relational."

This section extends the abstract relational model to incorporate several important object-relational ideas. It is followed by sections that cover object-relational extensions of SQL. We introduce the concept of object-relations in Section 10.3.1, then discuss one of its earliest embodiments — nested relations — in Section 10.3.2. ODL-like references for object-relations are discussed in Section 10.3.3, and in Section 10.3.4 we compare the object-relational model with the pure object-oriented approach.

### 10.3.1 From Relations to Object-Relations

While the relation remains the fundamental concept, the relational model has been extended to the *object-relational model* by incorporation of features such as:

1. *Structured types for attributes.* Instead of allowing only atomic types for attributes, object-relational systems support a type system like ODL's: types built from atomic types and type constructors for structs, sets, and bags, for instance. Especially important is a type that is a bag of structs, which is essentially a relation. That is, a value of one component of a tuple can be an entire relation, called a "nested relation."

2. *Methods.* These are similar to methods in ODL or any object-oriented programming system.

3. *Identifiers for tuples.* In object-relational systems, tuples play the role of objects. It therefore becomes useful in some situations for each tuple to have a unique ID that distinguishes it from other tuples, even from tuples that have the same values in all components. This ID, like the object-identifier assumed in ODL, is generally invisible to the user, although

### Backwards Compatibility

With little difference in essential features of the two models, it is interesting to consider why object-relational systems have dominated the pure object-oriented systems in the marketplace. The reason, we believe, is as follows. As relational DBMS's evolved into object-relational DBMS's, the vendors were careful to maintain backwards compatibility. That is, newer versions of the system would still run the old code and accept the same schemas, should the user not care to adopt any of the object-oriented features. On the other hand, migration to a pure object-oriented DBMS would require the installations to rewrite and reorganize extensively. Thus, whatever competitive advantage could be argued for object-oriented database systems was insufficient to motivate many to make the switch.

## 10.3.5 Exercises for Section 10.3

**Exercise 10.3.1:** Using the notation developed for nested relations and relations with references, give one or more relation schemas that represent the following information. In each case, you may exercise some discretion regarding what attributes of a relation are included, but try to keep close to the attributes found in our running movie example. Also, indicate whether your schemas exhibit redundancy, and if so, what could be done to avoid it.

a) Movies with their studio, their stars, and all the usual attributes of these.

b) Movies, with the usual attributes plus all their stars and the usual information about the stars.

! c) Studios, all the movies made by that studio, and all the stars of each movie, including all the usual attributes of studios, movies, and stars.

! **Exercise 10.3.2:** Render the players, teams, and fans of Exercise 4.1.3 in the object-relational model.

! **Exercise 10.3.3:** Render the genealogy of Exercise 4.1.6 in the object-relational model.

**Exercise 10.3.4:** Represent the banking information of Exercise 4.1.1 in the object-relational model developed in this section. Make sure that it is easy, given the tuple for a customer, to find their account(s) and *also* easy, given the tuple for an account to find the customer(s) that hold that account. Also, try to avoid redundancy.

! **Exercise 10.3.5:** If the data of Exercise 10.3.4 were modified so that an account could be held by only one customer [as in Exercise 4.1.2(a)], how could your answer to Exercise 10.3.4 be simplified?

### 10.4.7 Exercises for Section 10.4

**Exercise 10.4.1:** For our running movies example, choose type names for the attributes of each of the relations. Give attributes the same UDT if their values can reasonably be compared or exchanged, and give them different UDT's if they should not have their values compared or exchanged.

**Exercise 10.4.2:** Write type declarations for the following types:

a) `NameType`, with components for first, middle, and last names and a title.

b) `PersonType`, with a name of the person and references to the persons that are their mother and father. You must use the type from part (a) in your declaration.

c) `MarriageType`, with the date of the marriage and references to the husband and wife.

**Exercise 10.4.3:** Redesign our running battleships database schema of Exercise 2.4.3 to use type declarations and reference attributes where appropriate. Look for many-one relationships and try to represent them using an attribute with a reference type.

**Exercise 10.4.4:** Redesign our running products database schema of Exercise 2.4.1 to use type declarations and reference attributes where appropriate. In particular, in the relations `PC`, `Laptop`, and `Printer` make the `model` attribute be a reference to the `Product` tuple for that model.

! **Exercise 10.4.5:** In Exercise 10.4.4 we suggested that model numbers in the tables `PC`, `Laptop`, and `Printer` could be references to tuples of the `Product` table. Is it also possible to make the `model` attribute in `Product` a reference to the tuple in the relation for that type of product? Why or why not?

## 10.5 Operations on Object-Relational Data

All appropriate SQL operations from previous chapters apply to tables that are declared with a UDT or that have attributes whose type is a UDT. There are also some entirely new operations we can use, such as reference-following. However, some familiar operations, especially those that access or modify columns whose type is a UDT, involve new syntax.

### 10.5.1 Following References

Suppose $x$ is a value of type `REF(`$T$`)`. Then $x$ refers to some tuple $t$ of type $T$. We can obtain tuple $t$ itself, or components of $t$, by two means:

```
CREATE ORDERING FOR AddressType
ORDERING FULL BY RELATIVE WITH AddrLEG;
```

The function AddrLEG is shown in Fig. 10.23. Notice that if we reach line (7), it must be that the two city components are the same, so we compare the street components. Likewise, if we reach line (9), the only remaining possibility is that the cities are the same and the first street precedes the second alphabetically. □

```
1)  CREATE FUNCTION AddrLEG(
2)      x1 AddressType,
3)      x2 AddressType
4)  ) RETURNS INTEGER

5)  IF x1.city() < x2.city() THEN RETURN(-1)
6)  ELSEIF x1.city() > x2.city() THEN RETURN(1)
7)  ELSEIF x1.street() < x2.street() THEN RETURN(-1)
8)  ELSEIF x1.street() = x2.street() THEN RETURN(0)
9)  ELSE RETURN(1)
    END IF;
```

Figure 10.23: A comparison function for address objects

In practice, commercial DBMS's each have their own way of allowing the user to define comparisons for a UDT. In addition to the two approaches mentioned above, some of the capabilities offered are:

a) *Strict Object Equality.* Two objects are equal if and only if they are the same object.

b) *Method-Defined Equality.* A function is applied to two objects and returns true or false, depending on whether or not the two objects should be considered equal.

c) *Method-Defined Mapping.* A function is applied to one object and returns a real number. Objects are compared by comparing the real numbers returned.

### 10.5.5 Exercises for Section 10.5

**Exercise 10.5.1:** Use the StarsIn relation of Example 10.20 and the Movies and MovieStar relations accessible through StarsIn to write the following queries:

a) Find the names of the stars of *Bride and Prejudice.*

b) Find all the movies (objects of type `MovieType`) that starred Priety Zinta.

! c) Find the movies (title and year) with at least six stars.

! d) Find the titles and years of all movies in which at least one star lives in Hyderabad.

**Exercise 10.5.2:** Using your schema from Exercise 10.4.4, write the following queries. Don't forget to use references whenever appropriate.

a) Find the manufacturers of PC's with a hard disk larger than 80 gigabytes.

b) Find the manufacturers of laser printers.

! c) Produce a table giving for each model of laptop, the model of the laptop having the largest amount of RAM of any laptop made by the same manufacturer.

**Exercise 10.5.3:** Using your schema from Exercise 10.4.3, write the following queries. Don't forget to use references whenever appropriate and avoid joins (i.e., subqueries or more than one tuple variable in the `FROM` clause).

a) Find the battles in which at least one ship was damaged.

b) Find the ships with a displacement of more than 40,000 tons.

! c) Find the classes that had ships launched after 1926.

!! d) Find the battles in which at least one British ship was damaged.

! **Exercise 10.5.4:** Write a procedure to take a star name as argument and delete from `StarsIn` and `MovieStar` all tuples involving that star.

**Exercise 10.5.5:** Assuming the function `AddrLEG` of Fig. 10.23 is available, write a suitable function to compare objects of type `StarType`, and declare your function to be the basis of the ordering of `StarType` objects.

## 10.6 On-Line Analytic Processing

An important application of databases is examination of data for patterns or trends. This activity, called *OLAP* (standing for *On-Line Analytic Processing* and pronounced "oh-lap"), generally involves highly complex queries that use one or more aggregations. These queries are often termed *OLAP queries* or *decision-support queries*. Some examples will be given in Section 10.6.2. A typical example is for a company to search for those of its products that have markedly increasing or decreasing overall sales.

Decision-support queries typically examine very large amounts of data, even if the query results are small. In contrast, common database operations, such

only some dealers have had low sales of red Gobis. Thus, we further focus the query by looking at only red Gobis, and we partition along the dealer dimension as well. This query is:

```
SELECT dealer, month, SUM(price)
FROM (Sales NATURAL JOIN Autos) JOIN Days ON date = day
WHERE model = 'Gobi' AND color = 'red'
GROUP BY month, dealer;
```

At this point, we find that the sales per month for red Gobis are so small that we cannot observe any trends easily. Thus, we decide that it was a mistake to partition by month. A better idea would be to partition only by years, and look at only the last two years (2006 and 2007, in this hypothetical example). The final query is shown in Fig. 10.29. □

```
SELECT dealer, year, SUM(price)
FROM (Sales NATURAL JOIN Autos) JOIN Days ON date = day
WHERE model = 'Gobi' AND
      color = 'red' AND
      (year = 2006 OR year = 2007)
GROUP BY year, dealer;
```

Figure 10.29: Final slicing-and-dicing query about red Gobi sales

### 10.6.6 Exercises for Section 10.6

**Exercise 10.6.1:** An on-line seller of computers wishes to maintain data about orders. Customers can order their PC with any of several processors, a selected amount of main memory, any of several disk units, and any of several CD or DVD readers. The fact table for such a database might be:

```
Orders(cust, date, proc, memory, hd, od, quant, price)
```

We should understand attribute `cust` to be an ID that is the foreign key for a dimension table about customers, and understand attributes `proc`, `hd` (hard disk), and `od` (optical disk: CD or DVD, typically) analogously. For example, an `hd` ID might be elaborated in a dimension table giving the manufacturer of the disk and several disk characteristics. The `memory` attribute is simply an integer: the number of megabytes of memory ordered. The `quant` attribute is the number of machines of this type ordered by this customer, and the `price` attribute is the total cost of each machine ordered.

a) Which are dimension attributes, and which are dependent attributes?

b) For some of the dimension attributes, a dimension table is likely to be needed. Suggest appropriate schemas for these dimension tables.

! **Exercise 10.6.2:** Suppose that we want to examine the data of Exercise 10.6.1 to find trends and thus predict which components the company should order more of. Describe a series of drill-down and roll-up queries that could lead to the conclusion that customers are beginning to prefer a DVD drive to a CD drive.

## 10.7 Data Cubes

In this section, we shall consider the "formal" data cube and special operations on data presented in this form. Recall from Section 10.6.3 that the formal data cube (just "data cube" in this section) precomputes all possible aggregates in a systematic way. Surprisingly, the amount of extra storage needed is often tolerable, and as long as the warehoused data does not change, there is no penalty incurred trying to keep all the aggregates up-to-date.

In the data cube, it is normal for there to be some aggregation of the raw data of the fact table before it is entered into the data-cube and its further aggregates computed. For instance, in our cars example, the dimension we thought of as a serial number in the star schema might be replaced by the model of the car. Then, each point of the data cube becomes a description of a model, a dealer and a date, together with the sum of the sales for that model, on that date, by that dealer. We shall continue to call the points of the (formal) data cube a "fact table," even though the interpretation of the points may be slightly different from fact tables in a star schema built from a raw-data cube.

### 10.7.1 The Cube Operator

Given a fact table $F$, we can define an augmented table CUBE($F$) that adds an additional value, denoted $*$, to each dimension. The $*$ has the intuitive meaning "any," and it represents aggregation along the dimension in which it appears. Figure 10.30 suggests the process of adding a border to the cube in each dimension, to represent the $*$ value and the aggregated values that it implies. In this figure we see three dimensions, with the lightest shading representing aggregates in one dimension, darker shading for aggregates over two dimensions, and the darkest cube in the corner for aggregation over all three dimensions. Notice that if the number of values along each dimension is reasonably large, then the "border" represents only a small addition to the volume of the cube (i.e., the number of tuples in the fact table). In that case, the size of the stored data CUBE($F$) is not much greater than the size of $F$ itself.

A tuple of the table CUBE($F$) that has $*$ in one or more dimensions will have for each dependent attribute the sum (or another aggregate function) of the values of that attribute in all the tuples that we can obtain by replacing

```
('Gobi', NULL, '2001-05-21', 'Friendly Fred', 152000, 7)
('Gobi', NULL, '2001-05-21', NULL, 2348000, 100)
```

These each have NULL in a dimension (color in both cases) but do not have NULL in one or more of the following dimension attributes. □

### 10.7.3 Exercises for Section 10.7

**Exercise 10.7.1:** What is the ratio of the size of CUBE($F$) to the size of $F$ if fact table $F$ has the following characteristics?

a) $F$ has twelve dimension attributes, each with eight different values.

b) $F$ has nine dimension attributes, each with two different values.

**Exercise 10.7.2:** Use the materialized view SalesCube from Example 10.32 to answer the following queries:

a) Find the total sales of Gobis for each dealer.

b) Find the total number of blue Gobis sold by dealer "Smilin' Sally."

c) Find the average number of green Gobis sold on each day of June, 2008 by each dealer.

! **Exercise 10.7.3:** What help, if any, would the rollup SalesRollup of Example 10.33 be for each of the queries of Exercise 10.7.2?

**Exercise 10.7.4:** In Exercise 10.6.1 we spoke of PC-order data organized as a fact table with dimension tables for attributes cust, proc, memory, hd, and od. That is, each tuple of the fact table Orders has an ID for each of these attributes, leading to information about the PC involved in the order. Write a SQL query that will produce the data cube for this fact table.

**Exercise 10.7.5:** Answer the following queries using the data cube from Exercise 10.7.4. If necessary, use dimension tables as well. You may invent suitable names and attributes for the dimension tables.

a) Find the average price of computers with hard disks of size 100 gigabytes, for each month from June, 2005.

b) Find, for each amount of memory, the total number of computers ordered in each month of the year 2007.

c) List for each type of hard disk (e.g., SCSI or IDE) and each processor type the number of computers ordered.

! **Exercise 10.7.6:** The cube tuples mentioned in Example 10.32 are not in the rollup of Example 10.33. Are there other rollups that would contain these tuples?

!! **Exercise 10.7.7:** If the fact table $F$ to which we apply the CUBE operator is sparse (i.e., there are many fewer tuples in $F$ than the product of the number of possible values along each dimension), then the ratio of the sizes of CUBE($F$) and $F$ can be very large. How large can it be?

## 10.8 Summary of Chapter 10

- *Privileges*: For security purposes, SQL systems allow many different kinds of privileges to be managed for database elements. These privileges include the right to select (read), insert, delete, or update relations, the right to reference relations (refer to them in a constraint), and the right to create triggers.

- *Grant Diagrams*: Privileges may be granted by owners to other users or to the general user `PUBLIC`. If granted with the grant option, then these privileges may be passed on to others. Privileges may also be revoked. The grant diagram is a useful way to remember enough about the history of grants and revocations to keep track of who has what privilege and from whom they obtained those privileges.

- *SQL Recursive Queries*: In SQL, one can define a relation recursively — that is, in terms of itself. Or, several relations can be defined to be mutually recursive.

- *Monotonicity*: Negations and aggregations involved in a SQL recursion must be monotone — inserting tuples in one relation does not cause tuples to be deleted from any relation, including itself. Intuitively, a relation may not be defined, directly or indirectly, in terms of a negation or aggregation of itself.

- *The Object-Relational Model*: An alternative to pure object-oriented database models like ODL is to extend the relational model to include the major features of object-orientation. These extensions include nested relations, i.e., complex types for attributes of a relation, including relations as types. Other extensions include methods defined for these types, and the ability of one tuple to refer to another through a reference type.

- *User-Defined Types in SQL*: Object-relational capabilities of SQL are centered around the UDT, or user-defined type. These types may be declared by listing their attributes and other information, as in table declarations. In addition, methods may be declared for UDT's.

- *Relations With a UDT as Type*: Instead of declaring the attributes of a relation, we may declare that relation to have a UDT. If we do so, then its tuples have one component, and this component is an object of the UDT.