**Sub-query examples:**

**1.**

**CREATE**TABLE employee  (emp\_no    INTEGER NOT NULL,  
                        emp\_fname CHAR(20) NOT NULL,  
                        emp\_lname CHAR(20) NOT NULL,  
                        dept\_no   CHAR(4) NULL)

The table has data as follows:

emp\_no      emp\_fname            emp\_lname            dept\_no  
----------- -------------------- -------------------- -------  
          1 Matthew              Smith                d3  
          2 Ann                  Jones                d3  
          3 John                 Barrimore            d1  
          4 James                James                d2  
          5 Elsa                 Bertoni              d2  
          6 Elke                 Hansel               d2  
          7 Sybill               Moser                d1

**CREATE**TABLE department (dept\_no   CHAR(4) NOT NULL,  
                        dept\_name CHAR(25) NOT NULL,  
                        location  CHAR(30) NULL)

The table has data as follows:

dept\_no dept\_name                 location  
------- ------------------------- ------------------------------  
d1      developer                 Dallas  
d2      tester                    Seattle  
d3      marketing                 Dallas

**CREATE**TABLE works\_on   (emp\_no       INTEGER NOT NULL,  
                         project\_no    CHAR(4) NOT NULL,  
                         job CHAR (15) NULL,  
                         enter\_date    DATETIME NULL)

emp\_no  project\_no     job      enter\_date

------- ------------------------- ------------------------------

1  p1  analyst  1997.10.1  
1  p3  manager  1999.1.  
2  p2  clerk    1998.2.15  
2  p2  NULL      1998.6.1  
3  p2  NULL      1997.12.15  
4  p3  analyst  1998.10.15  
5  p1  manager  1998.4.15  
6  p1  NULL      1998.8.1  
7  p2  clerk    1999.2.1  
8  p3  clerk    1997.11.15  
7  p1 clerk    1998.1.4

**CREATE**TABLE project   (project\_no   CHAR(4) NOT NULL,  
                        project\_name CHAR(15) NOT NULL,  
                        budget FLOAT NULL)

Project\_no project\_name    budget

------- ------------------------- ------------------------------

p1  Search Engine        120000.00  
p2  Programming          95000.00  
p3  SQL                 186500.00

***Examples:***

-- Sub-queries and IN Operator

**SELECT**\* **FROM**employee **WHERE**dept\_no IN  
       (**SELECT**dept\_no **FROM**department **WHERE**location = 'Dallas')

*Output*  
emp\_no      emp\_fname            emp\_lname            dept\_no  
----------- -------------------- -------------------- -------  
          1 Matthew              Smith                d3  
          2 Ann                  Jones                d3  
          3 John                 Barrimore            d1  
          7 Sybill               Moser                d1

-- Nested three level sub-query with in operator  
  
**SELECT**emp\_lname **FROM**employee **WHERE**emp\_no IN  
        (**SELECT**emp\_no **FROM**works\_on **WHERE**project\_no IN  
          (**SELECT**project\_no **FROM**project **WHERE**project\_name = 'SQL'))

*Output*  
emp\_lname  
--------------------  
Smith  
James

-- Use IN operator for a single value

**SELECT**emp\_lname  
        **FROM**employee  
        **WHERE**'p3' IN  
       (**SELECT**project\_no  
          **FROM**works\_on  
          **WHERE**works\_on.emp\_no=employee.emp\_no)

GO  
emp\_lname  
--------------------  
Smith  
James

-- A subquery can be used with other comparison operators  
**SELECT**DISTINCT project\_no **FROM**works\_on **WHERE**emp\_no <  
        (**SELECT**emp\_no **FROM**employee **WHERE**emp\_lname = 'Moser')

project\_no  
----------  
p1  
p2  
p3

-- Subqueries and Comparison Operators  
**SELECT**emp\_fname, emp\_lname **FROM**employee  
        **WHERE**dept\_no = (**SELECT**dept\_no **FROM**department **WHERE**dept\_name = 'marketing')  
  
emp\_fname            emp\_lname  
-------------------- --------------------  
Matthew              Smith  
Ann                  Jones

--Get the number and the last name of the employee with the smallest employee number.  
  
**SELECT**emp\_no, emp\_lname **FROM**employee  
        **WHERE**emp\_no = (**SELECT**MIN(emp\_no) **FROM**employee)  
  
emp\_no      emp\_lname  
----------- --------------------  
          1 Smith

-- Use aggregate **function**MAX.  
**SELECT**emp\_no **FROM**works\_on **WHERE**enter\_date =  
        (**SELECT**MAX(enter\_date)  
          **FROM**works\_on **WHERE**job = 'manager')  
  
emp\_no  
-----------  
          1

-- Write a query as part of the **FROM**clause  
**SELECT**emp\_fname, emp\_lname  
        **FROM**(**SELECT**\* **FROM**employee **WHERE**emp\_no >= 1) AS empno\_10000  
  
emp\_fname            emp\_lname  
-------------------- --------------------  
Matthew              Smith  
Ann                  Jones  
John                 Barrimore  
James                James  
Elsa                 Bertoni  
Elke                 Hansel  
Sybill               Moser

-- Subqueries and ALL operator

**SELECT**job **FROM**works\_on **WHERE**emp\_no <= ALL  
        (**SELECT**emp\_no **FROM**employee)  
  
job  
---------------  
analyst  
manager

--ANY evaluates to **true if**the result of an inner query contains at least one row that satisfies the comparison  
**SELECT**DISTINCT emp\_no, project\_no, job **FROM**works\_on **WHERE**enter\_date > ANY  
        (**SELECT**enter\_date **FROM**works\_on)  
  
emp\_no      project\_no job  
----------- ---------- ---------------  
          1 p3         manager  
          2 p2         NULL  
          2 p2         clerk  
          3 p2         NULL  
          4 p3         analyst  
          5 p1         manager  
          6 p1         NULL  
          7 p1         clerk  
          7 p2         clerk  
          8 p3         clerk

-- Subqueries and ANY operator

**SELECT**emp\_fname, emp\_lname **FROM**employee **WHERE**emp\_no = ANY  
        (**SELECT**emp\_no **FROM**works\_on **WHERE**project\_no = 'p1')  
  
emp\_fname            emp\_lname  
-------------------- --------------------  
Matthew              Smith  
Elsa                 Bertoni  
Elke                 Hansel  
Sybill               Moser

-- EXISTS evaluates to **true if**its result contains at least one row  
**SELECT**job **FROM**works\_on **WHERE**exists (**SELECT**emp\_no **FROM**employee)  
  
job  
---------------  
analyst  
manager  
clerk  
NULL  
NULL  
analyst  
manager  
NULL  
clerk  
clerk  
clerk

-- EXISTS Function and Subqueries  
-- EXISTS **function**takes a subquery as an argument and returns **true if**the subquery returns one or more rows, and it returns **false if**the subquery returns zero rows.  
**SELECT**emp\_lname **FROM**employee **WHERE**EXISTS  
        (**SELECT**\* **FROM**works\_on **WHERE**employee.emp\_no = works\_on.emp\_no AND project\_no='p1')  
  
emp\_lname  
--------------------  
Smith  
Bertoni  
Hansel  
Moser

-- Correlated subquery **using**the department table in both inner and outer queries  
**SELECT**t1.\*  
       **FROM**department t1  
        **WHERE**t1.location IN  
        (**SELECT**t2.location  
          **FROM**department t2  
          **WHERE**t1.dept\_no <> t2.dept\_no)  
  
dept\_no dept\_name                 location  
------- ------------------------- ------------------------------  
d1      developer                 Dallas  
d3      marketing                 Dallas

-- Correlated Subqueries  
-- A subquery is said to be a correlated subquery **if**the inner query depends on the outer query **for**any of its values.  
**SELECT**emp\_lname  
        **FROM**employee  
        **WHERE**emp\_no IN  
        (**SELECT**emp\_no  
          **FROM**works\_on  
          **WHERE**project\_no='p3')  
  
emp\_lname  
--------------------  
Smith  
James

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**create**table employee(  
2>     ID          int,  
3>     name        nvarchar (10),  
4>     salary      int,  
5>     start\_date  datetime,  
6>     city        nvarchar (10),  
7>     region      **char**(1))

ID          name       salary      start\_date              city       region  
----------- ---------- ----------- ----------------------- ---------- ------  
          1 Jason            40420 1994-02-01 00:00:00.000 New York   W  
          2 Robert           14420 1995-01-02 00:00:00.000 Vancouver  N  
          3 Celia            24020 1996-12-03 00:00:00.000 Toronto    W  
          4 Linda            40620 1997-11-04 00:00:00.000 New York   N  
          5 David            80026 1998-10-05 00:00:00.000 Vancouver  W  
          6 James            70060 1999-09-06 00:00:00.000 Toronto    N  
          7 Alison           90620 2000-08-07 00:00:00.000 New York   W  
          8 Chris            26020 2001-07-08 00:00:00.000 Vancouver  N  
          9 Mary             60020 2002-06-09 00:00:00.000 Toronto    W

**create**table job(  
3>     ID              int,  
4>     title nvarchar  (10),  
5>     averageSalary   **int**)

ID          title      averageSalary  
----------- ---------- -------------  
          1 Developer           3000  
          2 Tester              4000  
          3 Designer            5000  
          4 Programmer          6000

a)

-- A subquery is simply a **SELECT**query within a **SELECT**query.   
Subqueries scalar expressions.  
  
-- Scalar Expressions  
**SELECT**City, Salary  
        , (**SELECT**AVG(Salary) **FROM**Employee)  
     AS AvgSalary  
 **FROM**Employee  
  
City       Salary      AvgSalary  
---------- ----------- -----------  
New York         40420       49580  
Vancouver        14420       49580  
Toronto          24020       49580  
New York         40620       49580  
Vancouver        80026       49580  
Toronto          70060       49580  
New York         90620       49580  
Vancouver        26020       49580  
Toronto          60020       49580

b)

**SELECT**ID, Name, Salary  
        , Salary - (**SELECT**Avg(Salary) **FROM**Employee)  
     **As**AvgSalaryDifference  
**FROM**Employee

ID          Name       Salary      AvgSalaryDifference  
----------- ---------- ----------- -------------------  
          1 Jason            40420               -9160  
          2 Robert           14420              -35160  
          3 Celia            24020              -25560  
          4 Linda            40620               -8960  
          5 David            80026               30446  
          6 James            70060               20480  
          7 Alison           90620               41040  
          8 Chris            26020              -23560  
          9 Mary             60020               10440

c)

-- Subqueries can be nested.  
 **SELECT**ID, Name  
 **FROM**Employee  
 **WHERE**ID IN  
       ( **SELECT**ID  
         **FROM**Job  
         **WHERE**title IN  
             (  **SELECT**title  
                **FROM**job )  
       )

ID          Name  
----------- ----------  
          1 Jason  
          2 Robert  
          3 Celia  
          4 Linda

d)

**SELECT**  
   MIN (start\_date),  
   (**SELECT**MAX (start\_date) **FROM**employee)  
**FROM**  
   Employee

----------------------- -----------------------  
2001-04-16 00:00:00.000 2008-03-18 00:00:00.000

e)

-- Using the EXISTS() Function  
-- It doesn't really matter what column or columns are returned in the   
subquery because you don't actually use these values.  
  
**SELECT**ID, Name  
**FROM**Employee  
**WHERE**EXISTS  
        (**SELECT**\* **FROM**Job  
         **WHERE**ID = Employee.ID)  
  
ID          Name  
----------- ----------  
          1 Jason  
          2 Robert  
          3 Celia  
          4 Linda

f)

-- NOT EXISTS() Add the NOT operator before the EXISTS statement:  
**SELECT**ID, Name  
**FROM**Employee  
**WHERE**NOT EXISTS  
        (**SELECT**\* **FROM**Job  
         **WHERE**ID = Employee.ID)  
  
ID          Name  
----------- ----------  
          5 David  
          6 James  
          7 Alison  
          8 Chris  
          9 Mary

g)

--The **WHERE**clause in the subquery's **SELECT**statement links the inner query to the outer query. This makes the inner query a correlated subquery  
  
**SELECT**TOP 29 e.ID, e.name, (**SELECT**j.title **FROM**job j **WHERE**j.ID = e.ID) 'title'  
**FROM**Employee e  
  
ID          name       title  
----------- ---------- ----------  
          1 Jason      Developer  
          2 Robert     Tester  
          3 Celia      Designer  
          4 Linda      Programmer  
          5 David      NULL  
          6 James      NULL  
          7 Alison     NULL  
          8 Chris      NULL  
          9 Mary       NULL

h)

-- Correlated subquery **using**Distinct:  
**SELECT**ID , Name  
**FROM**Employee AS e **WHERE**1 =  
     (**SELECT**DISTINCT ID  
      **FROM**Job **As**j  
      **WHERE**j.ID = e.ID)

ID          Name  
----------- ----------  
          1 Jason

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**CREATE**TABLE Employees  
 (  
   empid   **int**NOT NULL,  
   mgrid   **int**NULL,  
   empname varchar(25) NOT NULL,  
   salary money        NOT NULL)

1, NULL, 'Nancy', $10000.00)  
2, 1, 'Andrew', $5000.00)  
3, 1, 'Janet', $5000.00)  
4, 1, 'Margaret', $5000.00)  
5, 2, 'Steven', $2500.00)  
6> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(6, 2, 'Michael', $2500.00)  
7> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(7, 3, 'Robert', $2500.00)  
8> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(8, 3, 'Laura', $2500.00)  
9> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(9, 3, 'Ann', $2500.00)  
10> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(10, 4, 'Ina', $2500.00)  
11> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(11, 7, 'David', $2000.00)  
12> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(12, 7, 'Ron', $2000.00)  
13> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(13, 7, 'Dan', $2000.00)  
14> **INSERT**INTO employees(empid, mgrid, empname, salary) VALUES(14, 11, 'James', $1500.00)

a)Find employees who are not managers.(correlated subquery)

**SELECT**\*  
 **FROM**  
   Employees AS M  
 **WHERE**  
   NOT EXISTS  
              (  
              **SELECT**  
                empid  
               **FROM**  
                Employees AS E  
               **WHERE**  
                E.mgrid = M.empid  
              )

empid       mgrid       empname                   salary  
----------- ----------- ------------------------- ---------------------  
          5           2 Steven                                2500.0000  
          6           2 Michael                               2500.0000  
          8           3 Laura                                 2500.0000  
          9           3 Ann                                   2500.0000  
         10           4 Ina                                   2500.0000  
         12           7 Ron                                   2000.0000  
         13           7 Dan                                   2000.0000  
         14          11 James                                 1500.0000

**The *pubs2* Database**

*pubs2* database contains the tables *publishers, authors, titles, titleauthor, au\_pix, salesdetail, sales, stores, discounts, roysched,* and *blurbs.*

The *pubs2* database also contains primary and foreign keys, rules, defaults, views, triggers, and stored procedures.

## Diagram of the *pubs2* database

## 

## Tables in the *pubs2* database

The following sections describe each *pubs2* table. In the tables, each column header specifies the column name, its datatype (including any user-defined datatypes), and its null or not null status. The column header also specifies any defaults, rules, triggers, and indexes that affect the column.

### *publishers* table

*publishers* is defined as follows:

**create table publishers**

**(pub\_id char(4) not null,**

**pub\_name varchar(40) not null,**

**city varchar(20) null,**

**state char(2) null)**

Its primary key is *pub\_id*:

**sp\_primarykey publishers, pub\_id**

Its *pub\_idrule* rule is defined as:

**create rule pub\_idrule**

**as @pub\_id in**

**("1389", "0736", "0877", "1622", "1756")**

**or @pub\_id like "99[0-9][0-9]"**

Table A-1 lists the contents of *publishers*:

| publishers table |  |  |  |
| --- | --- | --- | --- |
| pub\_id | pub\_name | city | state |
| 0736 | New Age Books | Boston | MA |
| 0877 | Binnet & Hardley | Washington | DC |
| 1389 | Algodata Infosystems | Berkeley | CA |

### *authors* table

*authors* is defined as follows:

**create table authors**

**(au\_id id not null,**

**au\_lname varchar(40) not null,**

**au\_fname varchar(20) not null,**

**phone char(12) not null,**

**address varchar(40) null,**

**city varchar(20) null,**

**state char(2) null,**

**country varchar(12) null,**

**postalcode char(10) null)**

Its primary key is *au\_id*:

**sp\_primarykey authors, au\_id**

Its nonclustered index for the *au\_lname* and *au\_fname* columns is defined as:

**create nonclustered index aunmind**

**on authors (au\_lname, au\_fname)**

The *phone* column has the following default:

**create default phonedflt as "UNKNOWN"**

**sp\_bindefault phonedft, "authors.phone"**

The following view uses *authors*:

**create view titleview**

**as**

**select title, au\_ord, au\_lname,**

**price, total\_sales, pub\_id**

**from authors, titles, titleauthor**

**where authors.au\_id = titleauthor.au\_id**

**and titles.title\_id = titleauthor.title\_id**

Table A-2 lists the contents of *authors*:

| authors table |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| au\_id | au\_- lname | au\_ fname | phone | address | city | state | coun- try | post- alcode |
| 172-32-1176 | White | Johnson | 408 496-7223 | 10932 Bigge Rd. | Menlo Park | CA | USA | 94025 |
| 213-46-8915 | Green | Marjorie | 510 986-7020 | 309 63rd St. #411 | Oakland | CA | USA | 94618 |
| 238-95-7766 | Carson | Cheryl | 510 548-7723 | 589 Darwin Ln. | Berkeley | CA | USA | 94705 |
| 267-41-2394 | O'Leary | Michael | 408 286-2428 | 22 Cleveland Av. #14 | San Jose | CA | USA | 95128 |
| 274-80-9391 | Straight | Dick | 510 834-2919 | 5420 College Av. | Oakland | CA | USA | 94609 |
| 341-22-1782 | Smith | Meander | 913 843-0462 | 10 Mississippi Dr. | Lawrence | KS | USA | 66044 |
| 409-56-7008 | Bennet | Abraham | 510 658-9932 | 6223 Bateman St. | Berkeley | CA | USA | 94705 |
| 427-17-2319 | Dull | Ann | 415 836-7128 | 3410 Blonde St. | Palo Alto | CA | USA | 94301 |
| 472-27-2349 | Gringlesby | Burt | 707 938-6445 | PO Box 792 | Covelo | CA | USA | 95428 |
| 486-29-1786 | Locksley | Chastity | 415 585-4620 | 18 Broadway Av. | San Francisco | CA | USA | 94130 |
| 527-72-3246 | Greene | Morningstar | 615 297-2723 | 22 Graybar House Rd. | Nashville | TN | USA | 37215 |
| 648-92-1872 | Blotchet-Halls | Reginald | 503 745-6402 | 55 Hillsdale Bl. | Corvallis | OR | USA | 97330 |
| 672-71-3249 | Yokomoto | Akiko | 415 935-4228 | 3 Silver Ct. | Walnut Creek | CA | USA | 94595 |
| 712-45-1867 | del Castillo | Innes | 615 996-8275 | 2286 Cram Pl. #86 | Ann Arbor | MI | USA | 48105 |
| 722-51-5454 | DeFrance | Michel | 219 547-9982 | 3 Balding Pl. | Gary | IN | USA | 46403 |
| 724-08-9931 | Stringer | Dirk | 510 843-2991 | 5420 Telegraph Av. | Oakland | CA | USA | 94609 |
| 724-80-9391 | MacFeather | Stearns | 510 354-7128 | 44 Upland Hts. | Oakland | CA | USA | 94612 |
| 756-30-7391 | Karsen | Livia | 510 534-9219 | 5720 McAuley St. | Oakland | CA | USA | 94609 |
| 807-91-6654 | Panteley | Sylvia | 301 946-8853 | 1956 Arlington Pl. | Rockville | MD | USA | 20853 |
| 846-92-7186 | Hunter | Sheryl | 415 836-7128 | 3410 Blonde St. | Palo Alto | CA | USA | 94301 |
| 893-72-1158 | McBadden | Heather | 707 448-4982 | 301 Putnam | Vacaville | CA | USA | 95688 |
| 899-46-2035 | Ringer | Anne | 801 826-0752 | 67 Seventh Av. | Salt Lake City | UT | USA | 84152 |
| 998-72-3567 | Ringer | Albert | 801 826-0752 | 67 Seventh Av. | Salt Lake City | UT | USA | 84152 |

### *titles* table

*titles* is defined as follows:

**create table titles**

**(title\_id tid not null,**

**title varchar(80) not null,**

**type char(12) not null,**

**pub\_id char(4) null,**

**price money null,**

**advance money null,**

**total\_sales int null,**

**notes varchar(200) null,**

**pubdate datetime not null,**

**contract bit not null)**

Its primary key is *title\_id*:

**sp\_primarykey titles, title\_id**

Its *pub\_id* column is a foreign key to the *publishers* table:

**sp\_foreignkey titles, publishers, pub\_id**

Its nonclustered index for the *title* column is defined as:

**create nonclustered index titleind**

**on titles (title)**

Its *title\_idrule* is defined as:

**create rule title\_idrule**

**as**

**@title\_id like "BU[0-9][0-9][0-9][0-9]" or**

**@title\_id like "[MT]C[0-9][0-9][0-9][0-9]" or**

**@title\_id like "P[SC][0-9][0-9][0-9][0-9]" or**

**@title\_id like "[A-Z][A-Z]xxxx" or**

**@title\_id like "[A-Z][A-Z]yyyy"**

The *type* column has the following default:

**create default typedflt as "UNDECIDED"**

**sp\_bindefault typedflt, "titles.type"**

The *pubdate* column has this default:

**create default datedflt as getdate()**

**sp\_bindefault datedflt, "titles.pubdate"**

*titles* uses the following trigger:

**create trigger deltitle**

**on titles**

**for delete**

**as**

**if (select count(\*) from deleted, salesdetail**

**where salesdetail.title\_id = deleted.title\_id) >0**

**begin**

**rollback transaction**

**print "You can't delete a title with sales."**

**end**

The following view uses *titles*:

**create view titleview**

**as**

**select title, au\_ord, au\_lname,**

**price, total\_sales, pub\_id**

**from authors, titles, titleauthor**

**where authors.au\_id = titleauthor.au\_id**

**and titles.title\_id = titleauthor.title\_id**

Table A-3 lists the contents of *titles:*

| titles table |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| title- \_id | title | type | pub- \_id | price | ad- vance | total\_- sales | notes | pub- date | con- tract |
| BU1032 | The Busy Executive's Database Guide | business | 1389 | 19.99 | 5000.00 | 4095 | An overview of available database systems with emphasis on common business appli- cations. Illustrated. | Jun 12, 1986 | 1 |
| BU1111 | Cooking with Computers:Surrepti-tious Balance Sheets | business | 1389 | 11.95 | 5000.00 | 3876 | Helpful hints on how to use your electronic resources to the best advantage. | Jun 9, 1988 | 1 |
| BU2075 | You Can Combat Computer Stress! | business | 0736 | 2.99 | 10125.00 | 18722 | The latest medical and psycho-logical techniques for living with the electronic office. Easy-to-understand explana-tions. | Jun 30, 1985 | 1 |
| BU7832 | Straight Talk About Computers | business | 1389 | 19.99 | 5000.00 | 4095 | Annotated analysis of what computers can do for you: a no-hype guide for the critical user. | Jun 22, 1987 | 1 |
| MC2222 | Silicon Valley Gastro-nomic Treats | mod\_- cook | 0877 | 19.99 | 0.00 | 2032 | Favorite recipes for quick, easy, and elegant meals; tried and tested by people who never have time to eat, let alone cook. | Jun 9, 1989 | 1 |
| MC3021 | The Gourmet Microwave | mod\_- cook | 0877 | 2.99 | 15000.00 | 22246 | TraditionalFrench gourmet recipes adapted for modern microwave cooking. | Jun 18, 1985 | 1 |
| PC1035 | But Is It User Friendly? | popular\_comp | 1389 | 22.95 | 7000.00 | 8780 | A survey of software for the naive user, focusing on the "friendli-ness" of each. | Jun 30, 1986 | 1 |
| MC3026 | The Psychology of Computer Cooking | UNDE- CIDED | 0877 | NULL | NULL | NULL | NULL | Jul 24, 1991 | 0 |
| PC8888 | Secrets of Silicon Valley | popular\_comp | 1389 | 20.00 | 8000.00 | 4095 | Muckrak-ing reporting by two courag-eous women on the world's largest computer hardware and software manufact-urers. | Jun 12, 1987 | 1 |
| PC9999 | Net Etiquette | popular\_comp | 1389 | NULL | NULL | NULL | A must-read for computer conferenc-ing debu-tantes! | Jul 24, 1996 | 0 |
| PS1372 | Computer Phobic and Non-Phobic Individ-uals: Behavior Variations | psycho- logy | 0877 | 21.59 | 7000.00 | 375 | A must for the specialist, this book examines the difference between those who hate and fear computers and those who think they are swell. | Oct 21,1990 | 1 |
| PS2091 | Is Anger the Enemy? | psycho- logy | 0736 | 10.95 | 2275.00 | 2045 | Carefully researched study of the effects of strong emotions on the body. Metabolic charts included. | Jun 15, 1989 | 1 |
| PS2106 | Life Without Fear | psycho- logy | 0736 | 7.00 | 6000.00 | 111 | New exercise, meditation, and nutritional techniques that can reduce the shock of daily inter-actions. Popular audience. Sample menus included. Exercise video available separately. | Oct 5, 1990 | 1 |
| PS3333 | Prolonged Data Depriva-tion: Four Case Studies | psycho- logy | 0736 | 19.99 | 2000.00 | 4072 | What happens when the data runs dry? Searching evaluations of informationshortage effects on heavy users. | Jun 12, 1988 | 1 |
| PS7777 | Emotional Security: A New Algorithm | psycho- logy | 0736 | 7.99 | 4000.00 | 3336 | Protecting yourself and your loved ones from undue emotional stress in the modern world. Use of computer and nutritional aids empha-sized. | Jun 12, 1988 | 1 |
| TC3218 | Onions, Leeks, and Garlic: Cooking Secrets of the Mediterra- nean | trad\_- cook | 0877 | 20.95 | 7000.00 | 375 | Profusely illustrated in color, this makes a wonderful gift book for a cuisine-oriented friend. | Oct 21, 1990 | 1 |
| TC4203 | Fifty Years in Bucking-ham Palace Kitchens | trad\_cook | 0877 | 11.95 | 4000.00 | 15096 | More anecdotes from the Queen's favorite cook, describing life among English royalty. Recipes, techniques, tender vignettes. | Jun 12, 1985 | 1 |
| TC7777 | Sushi, Anyone? | trad\_cook | 0877 | 14.99 | 8000.00 | 4095 | Detailed instructions on improving your position in life by learning how to make authentic Japanese sushi in your spare time. 5-10% increase in number of friends per recipe reported from beta test. | Jun 12, 1987 | 1 |

### *titleauthor* table

*titleauthor* is defined as follows:

**create table titleauthor**

**(au\_id id not null,**

**title\_id tid not null,**

**au\_ord tinyint null,**

**royaltyper int null)**

Its primary keys are *au\_id* and *title\_id*:

**sp\_primarykey titleauthor, au\_id, title\_id**

Its *title\_id* and *au\_id* columns are foreign keys to *titles* and *authors*:

**sp\_foreignkey titleauthor, titles, title\_id**

**sp\_foreignkey titleauthor, authors, au\_id**

Its nonclustered index for the *au\_id* column is defined as:

**create nonclustered index auidind**

**on titleauthor(au\_id)**

Its nonclustered index for the *title\_id* column is defined as:

**create nonclustered index titleidind**

**on titleauthor(title\_id)**

The following view uses *titleauthor*:

**create view titleview**

**as**

**select title, au\_ord, au\_lname,**

**price, total\_sales, pub\_id**

**from authors, titles, titleauthor**

**where authors.au\_id = titleauthor.au\_id**

**and titles.title\_id = titleauthor.title\_id**

The following procedure uses *titleauthor:*

**create procedure byroyalty @percentage int**

**as**

**select au\_id from titleauthor**

**where titleauthor.royaltyper = @percentage**

Table A-4 lists the contents of *titleauthor*:

| titleauthor table |  |  |  |
| --- | --- | --- | --- |
| au\_id | title\_id | au\_ord | royaltyper |
| 172-32-1176 | PS3333 | 1 | 100 |
| 213-46-8915 | BU1032 | 2 | 40 |
| 213-46-8915 | BU2075 | 1 | 100 |
| 238-95-7766 | PC1035 | 1 | 100 |
| 267-41-2394 | BU1111 | 2 | 40 |
| 267-41-2394 | TC7777 | 2 | 30 |
| 274-80-9391 | BU7832 | 1 | 100 |
| 409-56-7008 | BU1032 | 1 | 60 |
| 427-17-2319 | PC8888 | 1 | 50 |
| 472-27-2349 | TC7777 | 3 | 30 |
| 486-29-1786 | PC9999 | 1 | 100 |
| 486-29-1786 | PS7777 | 1 | 100 |
| 648-92-1872 | TC4203 | 1 | 100 |
| 672-71-3249 | TC7777 | 1 | 40 |
| 712-45-1867 | MC2222 | 1 | 100 |
| 722-51-5454 | MC3021 | 1 | 75 |
| 724-80-9391 | BU1111 | 1 | 60 |
| 724-80-9391 | PS1372 | 2 | 25 |
| 756-30-7391 | PS1372 | 1 | 75 |
| 807-91-6654 | TC3218 | 1 | 100 |
| 846-92-7186 | PC8888 | 2 | 50 |
| 899-46-2035 | MC3021 | 2 | 25 |
| 899-46-2035 | PS2091 | 2 | 50 |
| 998-72-3567 | PS2091 | 1 | 50 |
| 998-72-3567 | PS2106 | 1 | 100 |

### *salesdetail* table

*salesdetail* is defined as follows:

**create table salesdetail**

**(stor\_id char(4) not null,**

**ord\_num numeric(6,0),**

**title\_id tid not null,**

**qty smallint not null,**

**discount float not null)**

Its primary keys are *stor\_id* and *ord\_num*:

**sp\_primarykey salesdetail, stor\_id, ord\_num**

Its *title\_id*, *stor\_id*, and *ord\_num* columns are foreign keys to *titles* and *sales*:

**sp\_foreignkey salesdetail, titles, title\_id**

**sp\_foreignkey salesdetail, sales, stor\_id, ord\_num**

Its nonclustered index for the *title\_id* column is defined as:

**create nonclustered index titleidind**

**on salesdetail (title\_id)**

Its nonclustered index for the *stor\_id* column is defined as:

**create nonclustered index salesdetailind**

**on salesdetail (stor\_id)**

Its *title\_idrule* rule is defined as:

**create rule title\_idrule**

**as**

**@title\_id like "BU[0-9][0-9][0-9][0-9]" or**

**@title\_id like "[MT]C[0-9][0-9][0-9][0-9]" or**

**@title\_id like "P[SC][0-9][0-9][0-9][0-9]" or**

**@title\_id like "[A-Z][A-Z]xxxx" or**

**@title\_id like "[A-Z][A-Z]yyyy"**

*salesdetail* uses the following trigger:

**create trigger totalsales\_trig on salesdetail**

**for insert, update, delete**

**as**

**/\* Save processing: return if there are no rows affected \*/**

**if @@rowcount = 0**

**begin**

**return**

**end**

**/\* add all the new values \*/**

**/\* use isnull: a null value in the titles table means**

**\*\* "no sales yet" not "sales unknown"**

**\*/**

**update titles**

**set total\_sales = isnull(total\_sales, 0) + (select sum(qty)**

**from inserted**

**where titles.title\_id = inserted.title\_id)**

**where title\_id in (select title\_id from inserted)**

**/\* remove all values being deleted or updated \*/**

**update titles**

**set total\_sales = isnull(total\_sales, 0) - (select sum(qty)**

**from deleted**

**where titles.title\_id = deleted.title\_id)**

**where title\_id in (select title\_id from deleted)**

Table A-5 lists the contents of *salesdetail*:

| salesdetail table |  |  |  |  |
| --- | --- | --- | --- | --- |
| stor\_id | ord\_num | title\_id | qty | discount |
| 7896 | 234518 | TC3218 | 75 | 40.000000 |
| 7896 | 234518 | TC7777 | 75 | 40.000000 |
| 7131 | Asoap432 | TC3218 | 50 | 40.000000 |
| 7131 | Asoap432 | TC7777 | 80 | 40.000000 |
| 5023 | XS-135-DER-432-8J2 | TC3218 | 85 | 40.000000 |
| 8042 | 91-A-7 | PS3333 | 90 | 45.000000 |
| 8042 | 91-A-7 | TC3218 | 40 | 45.000000 |
| 8042 | 91-A-7 | PS2106 | 30 | 45.000000 |
| 8042 | 91-V-7 | PS2106 | 50 | 45.000000 |
| 8042 | 55-V-7 | PS2106 | 31 | 45.000000 |
| 8042 | 91-A-7 | MC3021 | 69 | 45.000000 |
| 5023 | BS-345-DSE-860-1F2 | PC1035 | 1000 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | BU2075 | 500 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | BU1032 | 200 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | BU7832 | 150 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | PS7777 | 125 | 46.700000 |
| 5023 | NF-123-ADS-642-9G3 | TC7777 | 1000 | 46.700000 |
| 5023 | NF-123-ADS-642-9G3 | BU1032 | 1000 | 46.700000 |
| 5023 | NF-123-ADS-642-9G3 | PC1035 | 750 | 46.700000 |
| 7131 | Fsoap867 | BU1032 | 200 | 46.700000 |
| 7066 | BA52498 | BU7832 | 100 | 46.700000 |
| 7066 | BA71224 | PS7777 | 200 | 46.700000 |
| 7066 | BA71224 | PC1035 | 300 | 46.700000 |
| 7066 | BA71224 | TC7777 | 350 | 46.700000 |
| 5023 | ZD-123-DFG-752-9G8 | PS2091 | 1000 | 46.700000 |
| 7067 | NB-3.142 | PS2091 | 200 | 46.700000 |
| 7067 | NB-3.142 | PS7777 | 250 | 46.700000 |
| 7067 | NB-3.142 | PS3333 | 345 | 46.700000 |
| 7067 | NB-3.142 | BU7832 | 360 | 46.700000 |
| 5023 | XS-135-DER-432-8J2 | PS2091 | 845 | 46.700000 |
| 5023 | XS-135-DER-432-8J2 | PS7777 | 581 | 46.700000 |
| 5023 | ZZ-999-ZZZ-999-0A0 | PS1372 | 375 | 46.700000 |
| 7067 | NB-3.142 | BU1111 | 175 | 46.700000 |
| 5023 | XS-135-DER-432-8J2 | BU7832 | 885 | 46.700000 |
| 5023 | ZD-123-DFG-752-9G8 | BU7832 | 900 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | TC4203 | 550 | 46.700000 |
| 7131 | Fsoap867 | TC4203 | 350 | 46.700000 |
| 7896 | 234518 | TC4203 | 275 | 46.700000 |
| 7066 | BA71224 | TC4203 | 500 | 46.700000 |
| 7067 | NB-3.142 | TC4203 | 512 | 46.700000 |
| 7131 | Fsoap867 | MC3021 | 400 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | PC8888 | 105 | 46.700000 |
| 5023 | NF-123-ADS-642-9G3 | PC8888 | 300 | 46.700000 |
| 7066 | BA71224 | PC8888 | 350 | 46.700000 |
| 7067 | NB-3.142 | PC8888 | 335 | 46.700000 |
| 7131 | Asoap432 | BU1111 | 500 | 46.700000 |
| 7896 | 234518 | BU1111 | 340 | 46.700000 |
| 5023 | AX-532-FED-452-2Z7 | BU1111 | 370 | 46.700000 |
| 5023 | ZD-123-DFG-752-9G8 | PS3333 | 750 | 46.700000 |
| 8042 | 13-J-9 | BU7832 | 300 | 51.700000 |
| 8042 | 13-E-7 | BU2075 | 150 | 51.700000 |
| 8042 | 13-E-7 | BU1032 | 300 | 51.700000 |
| 8042 | 13-E-7 | PC1035 | 400 | 51.700000 |
| 8042 | 91-A-7 | PS7777 | 180 | 51.700000 |
| 8042 | 13-J-9 | TC4203 | 250 | 51.700000 |
| 8042 | 13-E-7 | TC4203 | 226 | 51.700000 |
| 8042 | 13-E-7 | MC3021 | 400 | 51.700000 |
| 8042 | 91-V-7 | BU1111 | 390 | 51.700000 |
| 5023 | AB-872-DEF-732-2Z1 | MC3021 | 5000 | 50.000000 |
| 5023 | NF-123-ADS-642-9G3 | PC8888 | 2000 | 50.000000 |
| 5023 | NF-123-ADS-642-9G3 | BU2075 | 2000 | 50.000000 |
| 5023 | GH-542-NAD-713-9F9 | PC1035 | 2000 | 50.000000 |
| 5023 | ZA-000-ASD-324-4D1 | PC1035 | 2000 | 50.000000 |
| 5023 | ZA-000-ASD-324-4D1 | PS7777 | 1500 | 50.000000 |
| 5023 | ZD-123-DFG-752-9G8 | BU2075 | 3000 | 50.000000 |
| 5023 | ZD-123-DFG-752-9G8 | TC7777 | 1500 | 50.000000 |
| 5023 | ZS-645-CAT-415-1B2 | BU2075 | 3000 | 50.000000 |
| 5023 | ZS-645-CAT-415-1B2 | BU2075 | 3000 | 50.000000 |
| 5023 | XS-135-DER-432-8J2 | PS3333 | 2687 | 50.000000 |
| 5023 | XS-135-DER-432-8J2 | TC7777 | 1090 | 50.000000 |
| 5023 | XS-135-DER-432-8J2 | PC1035 | 2138 | 50.000000 |
| 5023 | ZZ-999-ZZZ-999-0A0 | MC2222 | 2032 | 50.000000 |
| 5023 | ZZ-999-ZZZ-999-0A0 | BU1111 | 1001 | 50.000000 |
| 5023 | ZA-000-ASD-324-4D1 | BU1111 | 1100 | 50.000000 |
| 5023 | NF-123-ADS-642-9G3 | BU7832 | 1400 | 50.000000 |
| 5023 | BS-345-DSE-860-1F2 | TC4203 | 2700 | 50.000000 |
| 5023 | GH-542-NAD-713-9F9 | TC4203 | 2500 | 50.000000 |
| 5023 | NF-123-ADS-642-9G3 | TC4203 | 3500 | 50.000000 |
| 5023 | BS-345-DSE-860-1F2 | MC3021 | 4500 | 50.000000 |
| 5023 | AX-532-FED-452-2Z7 | MC3021 | 1600 | 50.000000 |
| 5023 | NF-123-ADS-642-9G3 | MC3021 | 2550 | 50.000000 |
| 5023 | ZA-000-ASD-324-4D1 | MC3021 | 3000 | 50.000000 |
| 5023 | ZS-645-CAT-415-1B2 | MC3021 | 3200 | 50.000000 |
| 5023 | BS-345-DSE-860-1F2 | BU2075 | 2200 | 50.000000 |
| 5023 | GH-542-NAD-713-9F9 | BU1032 | 1500 | 50.000000 |
| 5023 | ZZ-999-ZZZ-999-0A0 | PC8888 | 1005 | 50.000000 |
| 7896 | 124152 | BU2075 | 42 | 50.500000 |
| 7896 | 124152 | PC1035 | 25 | 50.500000 |
| 7131 | Asoap132 | BU2075 | 35 | 50.500000 |
| 7067 | NB-1.142 | PC1035 | 34 | 50.500000 |
| 7067 | NB-1.142 | TC4203 | 53 | 50.500000 |
| 8042 | 12-F-9 | BU2075 | 30 | 55.500000 |
| 8042 | 12-F-9 | BU1032 | 94 | 55.500000 |
| 7066 | BA27618 | BU2075 | 200 | 57.200000 |
| 7896 | 124152 | TC4203 | 350 | 57.200000 |
| 7066 | BA27618 | TC4203 | 230 | 57.200000 |
| 7066 | BA27618 | MC3021 | 200 | 57.200000 |
| 7131 | Asoap132 | MC3021 | 137 | 57.200000 |
| 7067 | NB-1.142 | MC3021 | 270 | 57.200000 |
| 7067 | NB-1.142 | BU2075 | 230 | 57.200000 |
| 7131 | Asoap132 | BU1032 | 345 | 57.200000 |
| 7067 | NB-1.142 | BU1032 | 136 | 57.200000 |
| 8042 | 12-F-9 | TC4203 | 300 | 62.200000 |
| 8042 | 12-F-9 | MC3021 | 270 | 62.200000 |
| 8042 | 12-F-9 | PC1035 | 133 | 62.200000 |
| 5023 | AB-123-DEF-425-1Z3 | TC4203 | 2500 | 60.500000 |
| 5023 | AB-123-DEF-425-1Z3 | BU2075 | 4000 | 60.500000 |
| 6380 | 342157 | BU2075 | 200 | 57.200000 |
| 6380 | 342157 | MC3021 | 250 | 57.200000 |
| 6380 | 356921 | PS3333 | 200 | 46.700000 |
| 6380 | 356921 | PS7777 | 500 | 46.700000 |
| 6380 | 356921 | TC3218 | 125 | 46.700000 |
| 6380 | 234518 | BU2075 | 135 | 46.700000 |
| 6380 | 234518 | BU1032 | 320 | 46.700000 |
| 6380 | 234518 | TC4203 | 300 | 46.700000 |
| 6380 | 234518 | MC3021 | 400 | 46.700000 |

### *sales* table

*sales* is defined as follows:

**create table sales**

**(stor\_id char(4) not null,**

**ord\_num varchar(20) not null,**

**date datetime not null)**

Its primary keys are *stor\_id* and *ord\_num*:

**sp\_primarykey sales, stor\_id, ord\_num**

Its *stor\_id* column is a foreign key to *stores*:

**sp\_foreignkey sales, stores, stor\_id**

Table A-6 lists the contents of *sales*:

| sales table |  |  |
| --- | --- | --- |
| stor\_id | ord\_num | date |
| 5023 | AB-123-DEF-425-1Z3 | Oct 31 1995 |
| 5023 | AB-872-DEF-732-2Z1 | Nov 6 1995 |
| 5023 | AX-532-FED-452-2Z7 | Dec 1 1996 |
| 5023 | BS-345-DSE-860-1F2 | Dec 12 1996 |
| 5023 | GH-542-NAD-713-9F9 | Feb 15 1997 |
| 5023 | NF-123-ADS-642-9G3 | Mar 18 1997 |
| 5023 | XS-135-DER-432-8J2 | Mar 21 1997 |
| 5023 | ZA-000-ASD-324-4D1 | Jul 27 1994 |
| 5023 | ZD-123-DFG-752-9G8 | Mar 21 1997 |
| 5023 | ZS-645-CAT-415-1B2 | Mar 21 1997 |
| 5023 | ZZ-999-ZZZ-999-0A0 | Mar 21 1997 |
| 6380 | 342157 | Dec 13 1994 |
| 6380 | 356921 | Feb 17 1995 |
| 7066 | BA27618 | Oct 12 1996 |
| 7066 | BA52498 | Oct 27 1995 |
| 7066 | BA71224 | Aug 5 1996 |
| 7067 | NB-1.142 | Jan 2 1997 |
| 7067 | NB-3.142 | Jun 13 1995 |
| 7131 | Asoap132 | Nov 16 1996 |
| 7131 | Asoap432 | Dec 20 1995 |
| 7131 | Fsoap867 | Sep 8 1996 |
| 7896 | 124152 | Aug 14 1996 |
| 7896 | 234518 | Feb 14 1997 |
| 8042 | 12-F-9 | Jul 13 1996 |
| 8042 | 13-E-7 | May 23 1995 |
| 8042 | 13-J-9 | Jan 13 1997 |
| 8042 | 55-V-7 | Feb 20 1997 |
| 8042 | 91-A-7 | Mar 20 1997 |
| 8042 | 91-V-7 | Mar 20 1997 |

### *stores* table

*stores* is defined as follows:

**create table stores**

**(stor\_id char(4) not null,**

**stor\_name varchar(40) not null,**

**stor\_address varchar(40) null,**

**city varchar(20) null,**

**state char(2) null,**

**country varchar(12) null,**

**postalcode char(10) null,**

**payterms varchar(12) null)**

Its primary key is *stor\_id*:

**sp\_primarykey stores, stor\_id**

Table A-7 lists the contents of *stores*:

| stores table |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| stor\_id | stor\_name | stor\_address | city | state | country | postal- code | pay- terms |
| 7066 | Barnum's | 567 Pasadena Ave. | Tustin | CA | USA | 92789 | Net 30 |
| 7067 | News & Brews | 577 First St. | Los Gatos | CA | USA | 96745 | Net 30 |
| 7131 | Doc-U-Mat: Quality Laundry and Books | 24-A Avrogado Way | Remulade | WA | USA | 98014 | Net 60 |
| 8042 | Bookbeat | 679 Carson St. | Portland | OR | USA | 89076 | Net 30 |
| 6380 | Eric the Read Books | 788 Catamaugus Ave. | Seattle | WA | USA | 98056 | Net 60 |
| 7896 | Fricative Bookshop | 89 Madison St. | Fremont | CA | USA | 90019 | Net 60 |
| 5023 | Thoreau Reading Discount Chain | 20435 Walden Expressway | Concord | MA | USA | 01776 | Net 60 |

### *roysched* table

*roysched* is defined as follows:

**create table roysched**

**title\_id tid not null,**

**lorange int null,**

**hirange int null,**

**royalty int null)**

Its primary key is *title\_id*:

**sp\_primarykey roysched, title\_id**

Its *title\_id* column is a foreign key to *titles*:

**sp\_foreignkey roysched, titles, title\_id**

Its nonclustered index for the *title\_id* column is defined as:

**create nonclustered index titleidind**

**on roysched (title\_id)**

Table A-8 lists the contents of *roysched*:

| roysched table |  |  |  |
| --- | --- | --- | --- |
| title\_id | lorange | hirange | royalty |
| BU1032 | 0 | 5000 | 10 |
| BU1032 | 5001 | 50000 | 12 |
| PC1035 | 0 | 2000 | 10 |
| PC1035 | 2001 | 3000 | 12 |
| PC1035 | 3001 | 4000 | 14 |
| PC1035 | 4001 | 10000 | 16 |
| PC1035 | 10001 | 50000 | 18 |
| BU2075 | 0 | 1000 | 10 |
| BU2075 | 1001 | 3000 | 12 |
| BU2075 | 3001 | 5000 | 14 |
| BU2075 | 5001 | 7000 | 16 |
| BU2075 | 7001 | 10000 | 18 |
| BU2075 | 10001 | 12000 | 20 |
| BU2075 | 12001 | 14000 | 22 |
| BU2075 | 14001 | 50000 | 24 |
| PS2091 | 0 | 1000 | 10 |
| PS2091 | 1001 | 5000 | 12 |
| PS2091 | 5001 | 10000 | 14 |
| PS2091 | 10001 | 50000 | 16 |
| PS2106 | 0 | 2000 | 10 |
| PS2106 | 2001 | 5000 | 12 |
| PS2106 | 5001 | 10000 | 14 |
| PS2106 | 10001 | 50000 | 16 |
| MC3021 | 0 | 1000 | 10 |
| MC3021 | 1001 | 2000 | 12 |
| MC3021 | 2001 | 4000 | 14 |
| MC3021 | 4001 | 6000 | 16 |
| MC3021 | 6001 | 8000 | 18 |
| MC3021 | 8001 | 10000 | 20 |
| MC3021 | 10001 | 12000 | 22 |
| MC3021 | 12001 | 50000 | 24 |
| TC3218 | 0 | 2000 | 10 |
| TC3218 | 2001 | 4000 | 12 |
| TC3218 | 4001 | 6000 | 14 |
| TC3218 | 6001 | 8000 | 16 |
| TC3218 | 8001 | 10000 | 18 |
| TC3218 | 10001 | 12000 | 20 |
| TC3218 | 12001 | 14000 | 22 |
| TC3218 | 14001 | 50000 | 24 |
| PC8888 | 0 | 5000 | 10 |
| PC8888 | 5001 | 10000 | 12 |
| PC8888 | 10001 | 15000 | 14 |
| PC8888 | 15001 | 50000 | 16 |
| PS7777 | 0 | 5000 | 10 |
| PS7777 | 5001 | 50000 | 12 |
| PS3333 | 0 | 5000 | 10 |
| PS3333 | 5001 | 10000 | 12 |
| PS3333 | 10001 | 15000 | 14 |
| PS3333 | 15001 | 50000 | 16 |
| BU1111 | 0 | 4000 | 10 |
| BU1111 | 4001 | 8000 | 12 |
| BU1111 | 8001 | 10000 | 14 |
| BU1111 | 12001 | 16000 | 16 |
| BU1111 | 16001 | 20000 | 18 |
| BU1111 | 20001 | 24000 | 20 |
| BU1111 | 24001 | 28000 | 22 |
| BU1111 | 28001 | 50000 | 24 |
| MC2222 | 0 | 2000 | 10 |
| MC2222 | 2001 | 4000 | 12 |
| MC2222 | 4001 | 8000 | 14 |
| MC2222 | 8001 | 12000 | 16 |
| MC2222 | 8001 | 12000 | 16 |
| MC2222 | 12001 | 20000 | 18 |
| MC2222 | 20001 | 50000 | 20 |
| TC7777 | 0 | 5000 | 10 |
| TC7777 | 5001 | 15000 | 12 |
| TC7777 | 15001 | 50000 | 14 |
| TC4203 | 0 | 2000 | 10 |
| TC4203 | 2001 | 8000 | 12 |
| TC4203 | 8001 | 16000 | 14 |
| TC4203 | 16001 | 24000 | 16 |
| TC4203 | 24001 | 32000 | 18 |
| TC4203 | 32001 | 40000 | 20 |
| TC4203 | 40001 | 50000 | 22 |
| BU7832 | 0 | 5000 | 10 |
| BU7832 | 5001 | 10000 | 12 |
| BU7832 | 10001 | 15000 | 14 |
| BU7832 | 15001 | 20000 | 16 |
| BU7832 | 20001 | 25000 | 18 |
| BU7832 | 25001 | 30000 | 20 |
| BU7832 | 30001 | 35000 | 22 |
| BU7832 | 35001 | 50000 | 24 |
| PS1372 | 0 | 10000 | 10 |
| PS1372 | 10001 | 20000 | 12 |
| PS1372 | 20001 | 30000 | 14 |
| PS1372 | 30001 | 40000 | 16 |
| PS1372 | 40001 | 50000 | 18 |

### *discounts* table

*discounts* is defined as follows:

**create table discounts**

**(discounttype varchar(40) not null,**

**stor\_id char(4) null,**

**lowqty smallint null,**

**highqty smallint null,**

**discount float not null)**

Its primary keys are *discounttype* and *stor\_id*:

**sp\_primarykey discounts, discounttype, stor\_id**

Its *stor\_id* is a foreign key to *stores*:

**sp\_foreignkey discounts, stores, stor\_id**

Table A-9 lists the contents of *discounts*:

| discounts table |  |  |  |  |
| --- | --- | --- | --- | --- |
| discounttype | stor\_id | lowqty | highqty | discount |
| Initial Customer | 8042 | NULL | NULL | 10.5 |
| Volume Discount | NULL | 100 | 1001 | 6.7 |
| Huge Volume Discount | NULL | 1001 | NULL | 10 |
| Customer Discount | 8042 | NULL | NULL | 5 |

### *blurbs* table

*blurbs* is defined as follows:

**create table blurbs**

**(au\_id id not null,**

**copy text null)**

Its primary key is *au\_id*:

**sp\_primarykey blurbs, au\_id**

Its *au\_id* column is a foreign key to *authors*:

**sp\_foreignkey blurbs, authors, au\_id**

Table A-10 lists the contents of *blurbs*:

| blurbs table |  |
| --- | --- |
| au\_id | copy |
| 486-29-1786 | If Chastity Locksley didn't exist, this troubled world would have created her! Not only did she master the mystic secrets of inner strength to conquer adversity when she encountered it in life, but also, after "reinventing herself," as she says, by writing "Emotional Security: A New Algorithm" following the devastating loss of her cat, Old Algorithm, she founded Publish or Perish, the page-by-page, day-by-day, write-yourself-to-wellness encounter workshops franchise empire, the better to share her inspiring discoveries with us all. Her "Net Etiquette," a brilliant social treatise in its own right and a fabulous pun, is the only civilized alternative to the gross etiquette often practiced on the public networks. |
| 648-92-1872 | A chef's chef and a raconteur's raconteur, Reginald Blotchet-Halls calls London his second home. "Th' palace kitchen's me first 'ome, act'lly!" Blotchet-Halls' astounding ability to delight our palates with palace delights is matched only by his equal skill in satisfying our perpetual hunger for delicious back-stairs gossip by serving up tidbits and entrees literally fit for a king! |
| 998-72-3567 | Albert Ringer was born in a trunk to circus parents, but another kind of circus trunk played a more important role in his life years later. He grew up as an itinerant wrestler and roustabout in the renowned Ringer Brothers and Betty and Bernie's Circus. Once known in the literary world only as Anne Ringer's wrestling brother, he became a writer while recuperating from a near-fatal injury received during a charity benefit bout with a gorilla. "Slingshotting" himself from the ring ropes, Albert flew over the gorilla's head and would have landed head first on the concrete. He was saved from certain death by Nana, an elephant he had befriended as a child, who caught him in her trunk. Nana held him so tightly that three ribs cracked and he turned blue from lack of oxygen. "I was delirious. I had an out-of-body experience! My whole life passed before me eyes. I promised myself 'If I get through this, I'll use my remaining time to share what I learned out there.' I owe it all to Nana!" |
| 899-46-2035 | Anne Ringer ran away from the circus as a child. A university creative writing professor and her family took Anne in and raised her as one of their own. In this warm and television-less setting she learned to appreciate the great classics of literature. The stream of aspiring and accomplished writers that flowed constantly through the house confirmed her repudiation of the circus family she'd been born into: "Barbarians!" The steadily growing recognition of her literary work was, to her, vindication. When her brother's brush with death brought them together after many years, she took advantage of life's crazy chance thing and broke the wall of anger that she had constructed to separate them. Together they wrote, "Is Anger the Enemy?" an even greater blockbuster than her other collaborative work, with Michel DeFrance, "The Gourmet Microwave." |
| 672-71-3249 | They asked me to write about myself and my book, so here goes: I started a restaurant called "de Gustibus" with two of my friends. We named it that because you really can't discuss taste. We're very popular with young business types because we're young business types ourselves. Whenever we tried to go out to eat in a group we always got into these long tiresome negotiations: "I just ate Italian," or "I ate Greek yesterday," or "I NEVER eat anything that's not organic!" Inefficient. Not what business needs today. So, it came to us that we needed a restaurant we could all go to every day and not eat the same thing twice in a row maybe for a year! We thought, "Hey, why make people choose one kind of restaurant over another, when what they really want is a different kind of food?" At de Gustibus you can eat Italian, Chinese, Japanese, Greek, Russian, Tasmanian, Iranian, and on and on all at the same time. You never have to choose. You can even mix and match! We just pooled our recipes, opened the doors, and never looked back. We're a big hit, what can I say? My recipes in "Sushi, Anyone?" are used at de Gustibus. They satisfy crowds for us every day. They will work for you, too. Period! |
| 409-56-7008 | Bennet was the classic too-busy executive. After discovering computer databases, he now has the time to run several successful businesses and sit on three major corporate boards. Bennet also donates time to community service organizations. Miraculously, he also finds time to write and market executive-oriented, in-depth computer hardware and software reviews. "I'm hyperkinetic, so being dynamic and fast-moving is a piece of cake. But being organized isn't easy for me or for anyone I know. There's just one word for that: 'databases!' Databases can cure you or kill you. If you get the right one, you can be like me. If you get the wrong one, watch out. Read my book!" |

### *au\_pix* table

*au\_pix* is defined as follows:

**create table au\_pix**

**(au\_id char(11) not null,**

**pic image null,**

**format\_type char(11) null,**

**bytesize int null,**

**pixwidth\_hor char(14) null,**

**pixwidth\_vert char(14) null)**

Its primary key is *au\_id*:

**sp\_primarykey au\_pix, au\_id**

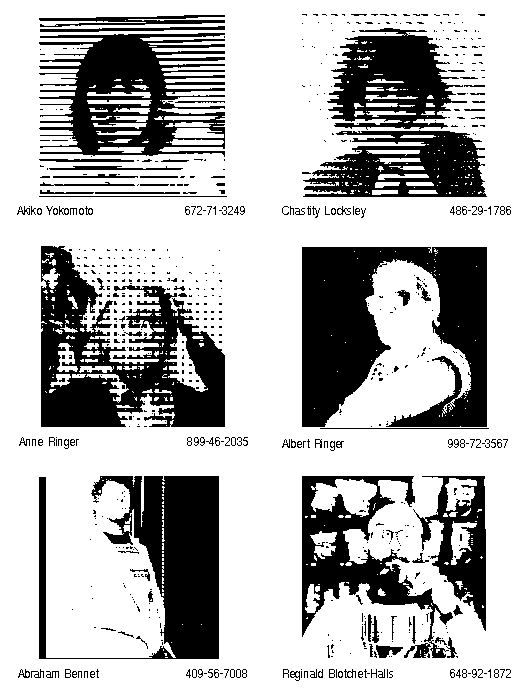
Its *au\_id* column is a foreign key to authors:

**sp\_foreignkey au\_pix, authors, au\_id**

Table A-11 lists the contents of *au\_pix*:

| au\_pix table |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| au\_id | pic | format\_type | bytesize | pixwidth\_hor | pixwidth\_vert |
| 409-56-7008 | 0x0000... | PICT | 30220 | 626 | 635 |
| 486-29-1786 | 0x59a6... | Sunraster | 27931 | 647 | 640 |
| 648-92-1872 | 0x59a6... | Sunraster | 36974 | 647 | 640 |
| 672-71-3249 | 0x000a... | PICT | 13487 | 654 | 639 |
| 899-46-2035 | 0x4949... | TIF | 52023 | 648 | 641 |
| 998-72-3567 | 0x4949... | TIF | 52336 | 653 | 637 |

The *pic* column contains binary data, which is not reproduced in its entirety in this table. The pictures represented by this data are shown on the next page. Since the image data (six pictures, two each in PICT, TIF, and Sunraster file formats) is quite large, you should run the **installpix2** script *only* if you want to use or test the image datatype. The image data is supplied to show how Sybase stores image data. Sybase does not supply any tools for displaying image data: you must use the appropriate screen graphics tools in order to display the images once you have extracted them from the database.

**Figure 20-1: Author portraits from the au\_pix table**

**Nested/Inner subquery**

**List** **the names of all authors whose royalty split is more than $75.**

**select au\_fname, au\_lname**

**from authors**

**where au\_id in**

**(select au\_id**

**from titleauthor**

**where royaltyper > 75)**

**Find the books that have the same price as *Straight Talk About Computers.***

**select title, price**

**from titles**

**where price =**

**(select price**

**from titles**

**where title = "Straight Talk About Computers")**

title price

--------------------------------------- -----

The Busy Executive's Database Guide 19.99

Straight Talk About Computers 19.99

Silicon Valley Gastronomic Treats 19.99

Prolonged Data Deprivation: Four Case Studies 19.99

**Find authors who live in the same city as Livia Karsen.**

**select au1.au\_lname, au1.au\_fname, au1.city**

**from authors au1**

**where au1.city in**

**(select au2.city**

**from authors au2**

**where au2.au\_fname = "Livia"**

**and au2.au\_lname = "Karsen")**

au\_lname au\_fname city

----------- --------- -------

Green Marjorie Oakland

Straight Dick Oakland

Stringer Dirk Oakland

MacFeather Stearns Oakland

Karsen Livia Oakland

Explicit correlation names make it clear that the reference to *authors* in the subquery is not the same as the reference to *authors* in the outer query.

Without explicit correlation, the subquery is:

**select au\_lname, au\_fname, city**

**from authors**

**where city in**

**(select city**

**from authors**

**where au\_fname = "Livia"**

**and au\_lname = "Karsen")**

Alternatively, you can state the above query, as well as other statements in which the subquery and the outer query refer to the same table, as self-joins:

**select au1.au\_lname, au1.au\_fname, au1.city**

**from authors au1, authors au2**

**where au1.city = au2.city**

**and au2.au\_lname = "Karsen"**

**and au2.au\_fname = "Livia"**

A subquery restated as a join may not return the results in the same order; additionally, the join may require the **distinct** keyword to eliminate duplicates.

**Find the names of authors who have participated in writing at least one popular computing book.**

**select au\_lname, au\_fname**

**from authors**

**where au\_id in**

**(select au\_id**

**from titleauthor**

**where title\_id in**

**(select title\_id**

**from titles**

**where type = "popular\_comp") )**

au\_lname au\_fname

---------------------- ------------

Carson Cheryl

Dull Ann

Locksley Chastity

Hunter Sheryl

(4 rows affected)

The outermost query selects all author names. The next query finds the authors' IDs, and the innermost query returns the title ID numbers PC1035, PC8888, and PC9999.

You can also express this query as a join:

**select au\_lname, au\_fname**

**from authors, titles, titleauthor**

**where authors.au\_id = titleauthor.au\_id**

**and titles.title\_id = titleauthor.title\_id**

**and type = "popular\_comp"**

**Update by doubling the price of all books published by New Age Books.**

**update titles**

**set price = price \* 2**

**where pub\_id in**

**(select pub\_id**

**from publishers**

**where pub\_name = "New Age Books")**

An equivalent **update** statement using a join is:

**update titles**

**set price = price \* 2**

**from titles, publishers**

**where titles.pub\_id = publishers.pub\_id**

**and pub\_name = "New Age Books"**

**Remove all records of sales of business books.**

**delete salesdetail**

**where title\_id in**

**(select title\_id**

**from titles**

**where type = "business")**

An equivalent **delete** statement using a join is:

**delete salesdetail**

**from salesdetail, titles**

**where salesdetail.title\_id = titles.title\_id**

**and type = "business"**

**Remove all records of sales of business books after checking for the records using subquery.**

**if exists (select title\_id**

**from titles**

**where type = "business")**

**begin**

**delete salesdetail**

**where title\_id in**

**(select title\_id**

**from titles**

**where type = "business")**

**end**

**Find the titles and types of books that have been written by authors living in California and that are also published there.**

**select title, type**

**from titles**

**where title in**

**(select title**

**from titles, titleauthor, authors**

**where titles.title\_id = titleauthor.title\_id**

**and titleauthor.au\_id = authors.au\_id**

**and authors.state = "CA")**

**and title in**

**(select title**

**from titles, publishers**

**where titles.pub\_id = publishers.pub\_id**

**and publishers.state = "CA")**

title type

----------------------------------- ----------

The Busy Executive's Database Guide business

Cooking with Computers:

Surreptitious Balance Sheets business

Straight Talk About Computers business

But Is It User Friendly? popular\_comp

Secrets of Silicon Valley popular\_comp

Net Etiquette popular\_comp

**Select the book titles that have had more than 5000 copies sold, lists their prices, and the price of the most expensive book.**

**select title, price,**

**(select max(price) from titles)**

**from titles**

**where total\_sales > 5000**

title price

----------------------------------- ----- ------

You Can Combat Computer Stress! 2.99 22.95

The Gourmet Microwave 2.99 22.95

But Is It User Friendly? 22.95 22.95

Fifty Years in Buckingham Palace

Kitchens 11.95 22.95

**Find the names of authors who live in the city where Algodata Infosystems.**

**select au\_lname, au\_fname**

**from authors**

**where city =**

**(select city**

**from publishers**

**where pub\_name = "Algodata Infosystems")**

au\_lname au\_fname

-------------- --------------

Carson Cheryl

Bennet Abraham

**Find the names of books that are priced higher than the current minimum price.**

**select title**

**from titles**

**where price >**

**(select min(price)**

**from titles)**

title

---------------------------------------------------

The Busy Executive's Database Guide

Cooking with Computers: Surreptitious Balance

Sheets

Straight Talk About Computers

Silicon Valley Gastronomic Treats

But Is It User Friendly?

Secrets of Silicon Valley

Computer Phobic and Non-Phobic Individuals:

Behavior Variations

Is Anger the Enemy?

Life Without Fear

Prolonged Data Deprivation: Four Case Studies

Emotional Security: A New Algorithm

Onions, Leeks, and Garlic: Cooking Secrets of the

Mediterranean

Fifty Years in Buckingham Palace Kitchens

Sushi, Anyone?

**Find the books that are priced higher than the lowest priced book in the *trad\_cook* category.**

**select title**

**from titles**

**where price >**

**(select min(price)**

**from titles**

**group by type**

**having type = "trad\_cook")**

**Which books commanded an advance greater than the largest advance paid by New Age Books?**

**select title**

**from titles**

**where advance > all**

**(select advance**

**from publishers, titles**

**where titles.pub\_id = publishers.pub\_id**

**and pub\_name = "New Age Books")**

title

----------------------------------------

The Gourmet Microwave

For each title, the outer query gets the titles and advances from the *titles* table, and it compares these to the advance amounts paid by New Age Books returned from the subquery. The outer query looks at the largest value in the list and determines whether the title being considered has commanded an even greater advance.

**Find the books that are priced higher than the highest-priced book in the *mod\_cook* category.**

**select title from titles where price > all**

**(select price from titles**

**where type = "mod\_cook")**

title

---------------------------------------------------

But Is It User Friendly?

Secrets of Silicon Valley

Computer Phobic and Non-Phobic Individuals:

Behavior Variations

Onions, Leeks, and Garlic: Cooking Secrets of

the Mediterranean

**Find the books that are priced higher than the highest-priced book in the *popular\_comp* category.**

**select title from titles where price > all**

**(select price from titles**

**where type = "popular\_comp")**

title

---------------------------------------------------

(0 rows affected)

No rows were returned because the subquery found that one of the books, *Net Etiquette*, has a null price.

**Find out which authors live in the same city by looking at the postal code.**

**select au\_fname, au\_lname, city**

**from authors**

**where city = all**

**(select city**

**from authors**

**where postalcode like "946%")**

**Find each title that has an advance larger than any advance amount paid by New Age Books.**

**select title**

**from titles**

**where advance > any**

**(select advance**

**from titles, publishers**

**where titles.pub\_id = publishers.pub\_id**

**and pub\_name = "New Age Books")**

title

---------------------------------------------------

The Busy Executive's Database Guide

Cooking with Computers: Surreptitious Balance

Sheets

You Can Combat Computer Stress!

Straight Talk About Computers

The Gourmet Microwave

But Is It User Friendly?

Secrets of Silicon Valley

Computer Phobic and Non-Phobic Individuals:

Behavior Variations

Is Anger the Enemy?

Life Without Fear

Emotional Security: A New Algorithm

Onions, Leeks, and Garlic: Cooking Secrets of

the Mediterranean

Fifty Years in Buckingham Palace Kitchens

Sushi, Anyone?

(14 rows affected)

For each title selected by the outer query, the inner query finds a list of advance amounts paid by New Age Books. The outer query looks at all the values in the list and determines whether the title being considered has commanded an advance that is larger than any of those values. In other words, this example finds titles with advances as large as or larger than the *lowest* value paid by New Age Books.

If the subquery does not return any values, the entire query fails.

**Find authors that live in the same city as any publisher.**

**select au\_lname, au\_fname**

**from authors**

**where city = any**

**(select city**

**from publishers)**

***Alternatively***

**select au\_lname, au\_fname**

**from authors**

**where city in**

**(select city**

**from publishers)**

au\_lname au\_fname

-------------- --------------

Carson Cheryl

Bennet Abraham

**Find the authors who live in a city where no publisher is located.**

**select au\_lname, au\_fname**

**from authors**

**where city != any**

**(select city**

**from publishers)**

The results include all 23 authors. This is because every author lives in *some* city where no publisher is located, and each author lives in only one city.

The inner query finds all the cities in which publishers are located, and then, for *each* city, the outer query finds the authors who do not live there.

Here is what happens when you substitute **not** **in** in the same query:

**select au\_lname, au\_fname**

**from authors**

**where city not in**

**(select city**

**from publishers)**

au\_lname au\_fname

-------------- ------------

White Johnson

Green Marjorie

O'Leary Michael

Straight Dick

Smith Meander

Dull Ann

Gringlesby Burt

Locksley Chastity

Greene Morningstar

Blotchet-Halls Reginald

Yokomoto Akiko

del Castillo Innes

DeFrance Michel

Stringer Dirk

MacFeather Stearns

Karsen Livia

Panteley Sylvia

Hunter Sheryl

McBadden Heather

Ringer Anne

Ringer Albert

(21 rows affected)

These are the results you want. They include all the authors except Cheryl Carson and Abraham Bennet, who live in Berkeley, where Algodata Infosystems is located.

You get the same results if you use **!=all**, which is equivalent to **not** **in**:

**select au\_lname, au\_fname**

**from authors**

**where city != all**

**(select city**

**from publishers)**

**Finds the names of the publishers who have published business books.**

**select pub\_name**

**from publishers**

**where pub\_id in**

**(select pub\_id**

**from titles**

**where type = "business")**

pub\_name

----------------------------------------

New Age Books

Algodata Infosystems

(2 rows affected)

This statement is evaluated in two steps. The inner query returns the identification numbers of the publishers who have published business books, 1389 and 0736. These values are then substituted in the outer query, which finds the names that go with the identification numbers in the *publishers* table. The query looks like this:

**select pub\_name**

**from publishers**

**where pub\_id in ("1389", "0736")**

Another way to formulate this query using a subquery is:

**select pub\_name**

**from publishers**

**where "business" in**

**(select type**

**from titles**

**where pub\_id = publishers.pub\_id)**

Note that the expression following the **where** keyword in the outer query can be a constant as well as a column name. You can use other types of expressions, such as combinations of constants and column names.

The preceding queries, like many other subqueries, can be alternatively formulated as a join query:

**select distinct pub\_name**

**from publishers, titles**

**where publishers.pub\_id = titles.pub\_id**

**and type = "business"**

Both this query and the subquery versions find publishers who have published business books. All are equally correct and produce the same results, though you may need to use the **distinct** keyword to eliminate duplicates.

**Find the names of all second authors who live in California and receive less than 30 percent of the royalties on a book.**

Using a subquery, the statement is:

**select au\_lname, au\_fname**

**from authors**

**where state = "CA"**

**and au\_id in**

**(select au\_id**

**from titleauthor**

**where royaltyper < 30**

**and au\_ord = 2)**

au\_lname au\_fname

------------------------ ------------

MacFeather Stearns

(1 row affected)

The outer query produces a list of the 15 authors who live in California. The inner query is then evaluated, producing a list of the IDs of the authors who meet the qualifications.

More than one condition can be included in the **where** clause of both the inner and the outer query.

Using a join, the query is expressed like this:

**select au\_lname, au\_fname**

**from authors, titleauthor**

**where state = "CA"**

**and authors.au\_id = titleauthor.au\_id**

**and royaltyper < 30**

**and au\_ord = 2**

A join can always be expressed as a subquery. A subquery can often be expressed as a join.

**Find the names of the publishers who have *not* published business books.**

**select pub\_name from publishers**

**where pub\_id not in**

**(select pub\_id**

**from titles**

**where type = "business")**

pub\_name

----------------------------------------

Binnet & Hardley

**Find the names of all the publishers who publish business books.**

**select pub\_name**

**from publishers**

**where exists**

**(select \***

**from titles**

**where pub\_id = publishers.pub\_id**

**and type = "business")**

pub\_name

----------------------------------------

New Age Books

Algodata Infosystems

(2 rows affected)

To conceptualize the resolution of this query, consider each publisher's name in turn. Does this value cause the subquery to return at least one row? In other words, does it cause the existence test to evaluate to TRUE?

In the results of the preceding query, the second publisher's name is Algodata Infosystems, which has an identification number of 1389. Are there any rows in the *titles* table in which *pub\_id* is 1389 and *type* is business? If so, "Algodata Infosystems" should be one of the values selected. The same process is repeated for each of the other publisher's names.

**Find authors that live in the same city as a publisher.**

**select au\_lname, au\_fname**

**from authors**

**where city = any**

**(select city**

**from publishers)**

**select au\_lname, au\_fname**

**from authors**

**where exists**

**(select \***

**from publishers**

**where authors.city = publishers.city)**

au\_lname au\_fname

-------------- --------------

Carson Cheryl

Bennet Abraham

**Find titles of books published by any publisher located in a city that begins with the letter "B".**

**select title**

**from titles**

**where exists**

**(select \***

**from publishers**

**where pub\_id = titles.pub\_id**

**and city like "B%")**

**select title**

**from titles**

**where pub\_id in**

**(select pub\_id**

**from publishers**

**where city like "B%")**

title

---------------------------------------------------

You Can Combat Computer Stress!

Is Anger the Enemy?

Life Without Fear

Prolonged Data Deprivation: Four Case Studies

Emotional Security: A New Algorithm

The Busy Executive's Database Guide

Cooking with Computers: Surreptitious Balance

Sheets

Straight Talk About Computers

But Is It User Friendly?

Secrets of Silicon Valley

Net Etiquette

**Find the names of publishers who do *not* publish business books.**

**select pub\_name**

**from publishers**

**where not exists**

**(select \***

**from titles**

**where pub\_id = publishers.pub\_id**

**and type = "business")**

pub\_name

----------------------------------------

Binnet & Hardley

(1 row affected)

**Find the titles for which there have been no sales.**

**select title**

**from titles**

**where not exists**

**(select title\_id**

**from salesdetail**

**where title\_id = titles.title\_id)**

title

-----------------------------------------

The Psychology of Computer Cooking

Net Etiquette

**Find the cities in which both an author and a publisher are located.**

**select distinct city**

**from authors**

**where exists**

**(select \***

**from publishers**

**where authors.city = publishers.city)**

city

--------------------

Berkeley

**Find the cities where an author lives but no publisher is located.**

**select distinct city**

**from authors**

**where not exists**

**(select \***

**from publishers**

**where authors.city = publishers.city)**

city

--------------------

Gary

Covelo

Oakland

Lawrence

San Jose

Ann Arbor

Corvallis

Nashville

Palo Alto

Rockville

Vacaville

Menlo Park

Walnut Creek

San Francisco

Salt Lake City

**Correlated subqueries**

**Find the names of all authors who earn 100 percent royalty on a book.**

**select au\_lname, au\_fname**

**from authors**

**where 100 in**

**(select royaltyper**

**from titleauthor**

**where au\_id = authors.au\_id)**

au\_lname au\_fname

-------------- ----------

White Johnson

Green Marjorie

Carson Cheryl

Straight Dick

Locksley Chastity

Blotchet-Hall Reginald

del Castillo Innes

Panteley Sylvia

Ringer Albert

This is how the preceding query is evaluated: Transact-SQL considers each row of the *authors* table for inclusion in the results, by substituting the value in each row in the inner query. For example, suppose Transact-SQL first examines the row for Johnson White. Then, *authors.au\_id* takes the value "172-32-1176," which Transact-SQL substitutes for the inner query:

**select royaltyper**

**from titleauthor**

**where au\_id = "172-32-1176"**

The result is 100, so the outer query evaluates to:

**select au\_lname, au\_fname**

**from authors**

**where 100 in (100)**

Since the **where** condition is true, the row for Johnson White is included in the results. If you go through the same procedure with the row for Abraham Bennet, you can see how that row is not included in the results.

**Find the types of books that are published by more than one publisher.**

**select distinct t1.type**

**from titles t1**

**where t1.type in**

**(select t2.type**

**from titles t2**

**where t1.pub\_id != t2.pub\_id)**

type

--------------------

business

psychology

(2 rows affected)

Correlation names are required in the following query to distinguish between the two roles in which the *titles* table appears. This nested query is equivalent to the self-join query:

**select distinct t1.type**

**from titles t1, titles t2**

**where t1.type = t2.type**

**and t1.pub\_id != t2.pub\_id**

**Find the sales of psychology books where the quantity is less than average for sales of that title.**

**select s1.ord\_num, s1.title\_id, s1.qty**

**from salesdetail s1**

**where title\_id like "PS%"**

**and s1.qty <**

**(select avg(s2.qty)**

**from salesdetail s2**

**where s2.title\_id = s1.title\_id)**

ord\_num title\_id qty

------------------ -------- ---

91-A-7 PS3333 90

91-A-7 PS2106 30

55-V-7 PS2106 31

AX-532-FED-452-2Z7 PS7777 125

BA71224 PS7777 200

NB-3.142 PS2091 200

NB-3.142 PS7777 250

NB-3.142 PS3333 345

ZD-123-DFG-752-9G8 PS3333 750

91-A-7 PS7777 180

356921 PS3333 200

(11 rows affected)

The outer query selects the rows of the *sales* table (or "s1"*)* one by one. The subquery calculates the average quantity for each sale being considered for selection in the outer query. For each possible value of *s1*, Transact-SQL evaluates the subquery and puts the record being considered in the results, if the quantity is less than the calculated average.

**Find the titles of books that have prices higher than average for books of the same type.**

**select t1.type, t1.title**

**from titles t1**

**where t1.price >**

**(select avg(t2.price)**

**from titles t2**

**where t1.type = t2.type)**

type title

--------- --------------------------------------

business The Busy Executive's Database Guide

business Straight Talk About Computers

mod\_cook Silicon Valley Gastronomic Treats

popular\_comp But Is It User Friendly?

psychology Computer Phobic and Non-Phobic

Individuals: Behavior Variations

psychology Prolonged Data Deprivation: Four Case

Studies

trad\_cook Onions, Leeks, and Garlic: Cooking

Secrets of the Mediterranean

(7 rows affected)

For each possible value of *t1*, Transact-SQL evaluates the subquery and includes the row in the results if the price value of that row is greater than the calculated average. It is not necessary to group by type explicitly, because the rows for which the average price is calculated are restricted by the **where** clause in the subquery.

**Find the types of books in which the maximum advance is more than twice the average within a given group.**

**select t1.type**

**from titles t1**

**group by t1.type**

**having max(t1.advance) >= any**

**(select 2 \* avg(t2.advance)**

**from titles t2**

**where t1.type = t2.type)**

type

----------

mod\_cook

(1 row affected)

The subquery above is evaluated once for each group that is defined in the outer query, that is, once for each type of book.