APPENDIX

Solutions

Chapter 1

S.1.1

Data independence means that database application programs are not dependent on the physical structure of the stored data in a database (physical data independence) and that database application programs are independent of the logical structure of the database (logical data independence).

S.1.2

The main concept of the relational model is a relation, i.e., a table. The whole relational model consists of only tables with one or more columns and zero or more rows. At every position in the table there is only one data value.

S.1.3

The table **employee** represents an entity, while the data for Ann Jones specifies an object, i.e., an instance of the entity.

S.1.4

The **works_on** table represents the relationship between employees and projects. The difference between the **works_on** table and the other tables of the sample database is that the **works_on** table shows the relationship between two entities, while each of the other tables represents an entity.

S.1.5

A) Yes, because it is unique.
B) Yes, because each title uniquely determines the corresponding **isbn**.
C) Yes (there are no dependencies between the non-key attributes, because there is just one attribute).

S.1.6

A) Yes, because all other columns are dependent on this one.
B) No, because of the functional dependency on the column **order_no**.

S.1.7

The **company** table is not in any normal form, because the column **Location** is multivalued.

S.1.8

A) 1NF, because the column **city** is functionally dependent on the column **supplier_no**, which is the partial key of the table.
B) Using the following two tables:
supplier1 (supplier_no, article)
supplier2 (supplier_no, city)

S.1.9

1NF, because the column **C** is dependent on the column **B**, which is the partial key of the table.

S.1.10

This table is in 3NF, but its design is not well done. (The primary key should be the combination of the columns **A** and **C**.)

Chapter 2

S.2.1

Right-click the **Databases** folder and select **New Database**. Enter **test** in the field **Name**, enter the value 2 in the field **File growth – In megabytes** and 20 in the field **Maximum file size – Restrict filegrowth (MB**). In the **Location** column change the path to **C:\tmp**. Do the same for the **Initial size** column to change it to10MB. Then click OK.

S.2.2

Select the **test** database inside the folder **Databases** of your SQL Server. Right-click it, select **Properties**, and choose the **Transaction Log** tab. Proceed with changing the values similar to exercise 2.1.

S.2.3

In the **Properties** dialog box of the **test** database choose the tab **Options** and select **DBO use only**. Several DBOs can use the database if the option **Single User** is not activated.

S.2.4

Right-click the folder **Tables** inside the **test** database. Select **New Table** and enter the name of the first table (**department**). Then enter the column names with the corresponding data types. Proceed in the same way with all four tables.

S.2.5

Click the database **AdventureWorks** in the folder **Databases** and select **Tables**. All user tables of the database are now shown in the detail pane. Double-click the **Person.Address** table to display its properties.

S.2.6

It's not possible to create another database with the same name.

S.2.7

Choose the function **File** in the main menu and select **Save as**. Then change the destination directory, enter the new filename (**createdb**), and click OK.

S.2.8

Either enter USE test and execute it inside the Query Analyzer's window or select the database from the **Database** select box in the tool bar.

S.2.9

Open the Query Editor, select the **AdventureWorks** database from the drop-down menu, and enter SELECT * FROM orders. After starting the execution of the statement, press the red stop button to interrupt the current execution.

Chapter 3

S.3.1

The difference is in storage length and range of possible values.

TINYINT is stored in one byte with a value range from 0 to 255. SMALLINT is stored in two bytes with a value range from –32768 to 32767. INT is stored in four bytes with a value range from –2147483648 to 2147483647.

S.3.2

CHAR is a string which can store up to 8,000 characters. It stores the amount of characters given by declaration. VARCHAR also can store up to 8,000 characters. However, the strings are stored with their actual length. Use CHAR when the data values in a column are expected to be approximately of the same size. Use VARCHAR when the data values in a column are expected to vary considerably in size.

S.3.3

```
SET DATEFORMAT ymd
```

S.3.4

```
SELECT DB_ID('test')
```

S.3.5

```
SELECT @@VERSION, @@LANGUAGE
```

S.3.6

(01000101)
(11011111)
(00000110)

S.3.7

A + NULL : result is NULL, independent of A
NULL = NULL : result is NULL
B OR NULL : true, if B is true, otherwise NULL
B AND NULL : false, if B is false, otherwise NULL

S.3.8

By setting QUOTED_IDENTIFIER to OFF

S.3.9

Delimited identifiers specify a special kind of identifiers that allow the use of reserved keywords as identifiers or table names that include blanks.

Chapter 4

S.4.1

```
USE master
GO
CREATE DATABASE test_db
        ON (NAME = test_db_dat,
        FILENAME='C:\tmp\test_db.mdf',
        SIZE = 5, MAXSIZE = UNLIMITED, FILEGROWTH = 8%)
        LOG ON
         (NAME=test_db_log,
        FILENAME = 'C:\tmp\test_db_log.ldf',
        SIZE = 2, MAXSIZE = 10, FILEGROWTH = 500KB)
```

S.4.2

```
USE master
ALTER DATABASE test_db
    ADD LOG FILE (NAME=emp_log1,
                  FILENAME='C:\tmp\test_db.ldf',
                   SIZE=2, MAXSIZE=UNLIMITED, FILEGROWTH=2)
```

S.4.3

```
USE master
ALTER DATABASE test_db
       MODIFY FILE
       (NAME = test_db_dat, SIZE = 10MB)
```

S.4.4

The NOT NULL specification is necessary for all columns that are part of the primary key.

S.4.5

dept_no and **project_no** are defined as CHAR-values, because they may contain characters other than numerical values.

S.4.6

```
CREATE TABLE customers (customerid CHAR(5) NOT NULL,
                        companyName VARCHAR(40) NOT NULL,
                        contactName CHAR(30) NULL,
                         address VARCHAR(60) NULL,
                         city CHAR(15) NULL,
                         phone CHAR(24) NULL,
                         fax CHAR(24) NULL)

CREATE TABLE orders (orderid INTEGER NOT NULL,
                     customerid CHAR(5) NOT NULL,
                     orderdate DATETIME NULL,
                     shippeddate DATETIME NULL,
                     freight MONEY NULL,
                     shipname VARCHAR(40) NULL,
                     shipaddress VARCHAR(60) NULL,
                      quantity INTEGER NULL)
```

S.4.7

```
ALTER TABLE orders
        ADD shipregion INTEGER NULL
```

S.4.8

```
ALTER TABLE orders
         ALTER COLUMN shipregion CHAR(8) NULL
```

S.4.9

```
ALTER TABLE orders
           DROP COLUMN shipregion
```

S.4.10

After deleting a table with DROP TABLE all data, indices and triggers belonging to the removed table are also dropped. In contrast to that, all views that are defined using the table are not removed.

S.4.11

```
DROP TABLE orders

DROP TABLE customers

CREATE TABLE customers (customerid CHAR(5) NOT NULL
                        CONSTRAINT prim_cust PRIMARY KEY,

                        companyName VARCHAR(40) NOT NULL,
                        contactName CHAR(30) NULL,
                        address VARCHAR(60) NULL,
                        city CHAR(15) NULL,
                        phone CHAR(24) NULL,
                        fax CHAR(24) NULL)

CREATE TABLE orders (orderid INTEGER NOT NULL,
                     customerid CHAR(5) NOT NULL,
                     orderdate DATETIME NULL,
                      shippeddate DATETIME NULL,
                       freight MONEY NULL,
```

```
                    shipname VARCHAR(40) NULL,
                     shipaddress VARCHAR(60) NULL,
                    quantity INTEGER NULL,
                   CONSTRAINT prim_ord PRIMARY KEY(orderid),
                    CONSTRAINT foreign_orders FOREIGN
KEY(customerid) REFERENCES customers(customerid))
```

S.4.12

It is not possible to insert the row, because of the referential constraint enforced in the CREATE TABLE statement (see S.4.11). SQL Server will print the following message:

"INSERT statement conflicted with COLUMN FOREIGN KEY constraint"

S.4.13

```
ALTER TABLE orders
      ALTER COLUMN orderdate DATETIME NULL

ALTER TABLE orders
       ADD CONSTRAINT AddDateDflt DEFAULT getdate() FOR orderdate
```

S.4.14

```
ALTER TABLE orders
      ADD CONSTRAINT limit_qu
      CHECK (quantity BETWEEN 1 AND 30)
```

S.4.15

Creating the new data type:

```
sp_addtype western_countries, 'char(2)', 'NOT NULL'
GO
```

Creating a new default value for our data type:

```
CREATE DEFAULT western_countries_default AS 'CA'
GO
```

Binding the new default value to our data type:

```
sp_bindefault 'western_countries_default', 'western_countries'
GO
```

Creating a rule for the allowed values for our data type:

```
CREATE RULE western_countries_rule
        AS @selection IN ('CA','WA','OR','NM')
GO
```

Binding the new rule to our data type:

```
sp_bindrule western_countries_rule, western_countries
      GO
```

Now creating the new table with the user-defined data type:

```
CREATE TABLE regions
      (city CHAR(25) NOT NULL,
       country western_countries)
```

S.4.16

```
sp_helpconstraint orders
```

S.4.17

```
ALTER TABLE customers
        DROP CONSTRAINT prim_cust
```

That statement will not work, because the primary key constraint **prim_cust** is referenced by the foreign key constraint defined in the table **orders**.

S.4.18

```
ALTER TABLE orders
      DROP CONSTRAINT limit_qu
```

S.4.19

```
sp_rename 'customers.city', town
```

Chapter 5

S.5.1

```
SELECT *
    FROM works_on
```

S.5.2

```
SELECT emp_no
    FROM works_on
    WHERE Job = 'Clerk'
```

S.5.3

```
SELECT emp_no
    FROM works_on
    WHERE project_no = 'p2'
    AND emp_no < 10000
```

or:

```
SELECT emp_no
    FROM works_on
    WHERE project_no = 'p2'
    AND emp_no BETWEEN 0 AND 9999
```

S.5.4

```
SELECT emp_no
  FROM works_on
  WHERE enter_date NOT BETWEEN
  '01.01.1998' AND '12.31.1998'
```

S.5.5

```
SELECT emp_no
    FROM works_on
    WHERE project_no = 'p1'
    AND (job = 'Manager' OR job = 'Analyst')
```

S.5.6

```
SELECT enter_date
       FROM works_on
       WHERE project_no = 'p2'
      AND Job IS NULL
```

S.5.7

```
SELECT emp_no, emp_lname
      FROM employee
      WHERE emp_fname LIKE '%t%t%'
```

S.5.8

```
SELECT emp_no, emp_fname
      FROM employee
     WHERE emp_lname LIKE '_[ao]%es'
```

S.5.9

```
SELECT emp_no
     FROM employee
    WHERE dept_no =
        (SELECT dept_no FROM department
              WHERE location = 'Seattle')
```

S.5.10

```
SELECT emp_lname, emp_fname
       FROM employee
       WHERE emp_no IN
          (SELECT emp_no
               FROM works_on
              WHERE enter_date = '04/01/1998')
```

S.5.11

```
SELECT location
      FROM department
      GROUP BY location
```

S.5.12

If you use GROUP BY without any additional specifications (aggregates, HAVING clause), it is exactly like DISTINCT. (It divides a table into groups and returns one row for each group.)

S.5.13

All NULL values belong to one group. (This is not exactly in accordance with the fact that each NULL value is a value per se, and it cannot be compared with other NULL values.)

S.5.14

COUNT(expression) takes an argument (i.e., a column or expression) and displays all non NULL occurrences of that argument. COUNT(*) counts all rows, whether or not any particular column contains a NULL value.

S.5.15

```
SELECT MAX(emp_no)
      FROM employee
```

S.5.16

```
SELECT job
      FROM works_on
     GROUP BY job
     HAVING COUNT(*) > 2
```

S.5.17

```
SELECT DISTINCT emp_no
      FROM works_on
      WHERE (Job = 'Clerk' OR  emp_no IN
           (SELECT emp_no
              FROM employee
             WHERE dept_no='d3'))
```

S.5.18

The inner SELECT statement can be used in conjunction with a comparison operator, such as =, if its result set has a maximum of one row (in this case = may be used). The result of the SELECT statement in E.5.18 has more than one row. Therefore, the comparison operator = has to be replaced with the IN operator.

The correct syntax form is:

```
SELECT project_name
   FROM project
   WHERE project_no IN
      (SELECT project_no FROM works_on WHERE Job = 'Clerk')
```

S.5.19

Temporary tables can be used to store the intermediate result of a complex query.

S.5.20

Local temporary tables are removed at the end of the current session, while the global temporary tables are removed at the end of the session that created the table.

Chapter 6

S.6.1

```
SELECT *
       FROM project, works_on
       WHERE project.project_no = works_on.project_no

SELECT project.*, emp_no, job, enter_date
      FROM project, works_on
      WHERE project.project_no = works_on.project_no

SELECT *
     FROM project, works_on
```

SQL-92:

```
SELECT *
       FROM project
       JOIN works_on ON project.project_no = works_on.project_no

SELECT project.*, emp_no, Job, enter_date
      FROM project JOIN works_on
     ON project.project_no = works_on.project_no

SELECT *
   FROM project CROSS JOIN works_on
```

S.6.2

You need at least N-1 join conditions.

S.6.3

```
SELECT emp_no, job
    FROM works_on, project
   WHERE works_on.project_no = project.project_no
   AND project_name = 'Gemini'
```

SQL-92:

```
SELECT emp_no, job
      FROM works_on JOIN project
      ON works_on.project_no = project.project_no
      WHERE project_name = 'Gemini'
```

S.6.4

```
SELECT emp_fname, emp_lname
       FROM employee, department
      WHERE employee.dept_no = department.dept_no
      AND (dept_name = 'Research' OR dept_name = 'Accounting')
```

SQL-92:

```
SELECT emp_fname, emp_lname
       FROM employee JOIN department
       ON employee.dept_no = department.dept_no
       WHERE (dept_name = 'Research' OR dept_name = 'Accounting')
```

S.6.5

```
SELECT enter_date
       FROM works_on, employee
      WHERE works_on.emp_no = employee.emp_no
       AND job = 'Clerk'
       AND dept_no = 'd1'
```

SQL-92:

```
SELECT enter_date
      FROM works_on JOIN employee
     ON works_on.emp_no = employee.emp_no
     WHERE job = 'Clerk'
     AND dept_no = 'd1'
```

S.6.6

```
SELECT project_name
      FROM project
      WHERE project_no IN
      (SELECT project_no
           FROM works_on
          WHERE Job = 'Clerk'
          GROUP BY project_no
          HAVING COUNT(*) > 1)
```

S.6.7

```
SELECT emp_fname, emp_lname
      FROM employee, works_on, project
      WHERE employee.emp_no = works_on.emp_no
      AND works_on.project_no = project.project_no
      AND project_name = 'Mercury'
     AND job = 'Manager'
```

SQL-92:

```
SELECT emp_fname, emp_lname
      FROM employee
      JOIN works_on ON employee.emp_no = works_on.emp_no
      JOIN project ON works_on.project_no = project.project_no
      WHERE project_name = 'Mercury'
      AND job = 'Manager'
```

S.6.8

```
SELECT emp_fname, emp_lname
     FROM employee
     WHERE emp_no IN
```

```
(SELECT a.emp_no
        FROM works_on a, works_on b
        WHERE  b.enter_date=a.enter_date
        AND a.emp_no != b.emp_no)
```

S.6.9

```
SELECT a.emp_no
        FROM employee_enh a, employee_enh b
       WHERE a.domicile = b.domicili
        AND a.dept_no = b.dept_no
       AND a.emp_no != b.emp_no
```

S.6.10

```
SELECT emp_no
        FROM employee, department
       WHERE employee.dept_no = department.dept_no
        AND dept_name = 'Marketing'

SELECT emp_no
       FROM employee
       WHERE dept_no =
        (SELECT dept_no
            FROM department
            WHERE dept_name = 'Marketing')
```

SQL-92:

```
SELECT emp_no
       FROM employee
       JOIN department
       ON employee.dept_no = department.dept_no
       WHERE dept_name = 'Marketing'
```

Chapter 7

S.7.1

```
INSERT INTO employee values(11111,'Julia','Long',NULL)
```

S.7.2

```
CREATE TABLE emp_d1_d2  (emp_no INTEGER NOT NULL,
                  emp_fname CHAR(20) NOT NULL,
                 emp_lname CHAR(20) NOT NULL,
                 dept_no CHAR(4) NULL)

INSERT INTO emp_d1_d2
SELECT emp_no, emp_fname, emp_lname, dept_no
       FROM employee
       WHERE dept_no IN ('d1', 'd2')
```

or:

```
SELECT emp_no, emp_fname, emp_lname, dept_no
   INTO emp_d1_d2
   FROM employee
  WHERE dept_no IN ('d1', 'd2')
```

S.7.3

```
CREATE TABLE employee_three  (emp_no INTEGER NOT NULL,
          emp_fname CHAR(20) NOT NULL,
          emp_lname CHAR(20) NOT NULL,
          dept_no CHAR(4) NULL)
INSERT INTO employee_three(emp_no, emp_fname, emp_lname, dept_no)
   SELECT emp_no, emp_fname, emp_lname, dept_no
       FROM employee
       WHERE emp_no IN
       (SELECT emp_no FROM works_on
            WHERE enter_date BETWEEN '01.01.1998' AND '12.31.1998')
```

S.7.4

```
UPDATE works_on
     SET job = 'Clerk'
      WHERE job = 'Manager' AND project_no = 'p1'
```

S.7.5

```
UPDATE project
      SET budget = NULL
```

S.7.6

```
UPDATE works_on
      SET job = 'Manager'
       WHERE emp_no = 28559
```

S.7.7

```
UPDATE project
      SET budget = budget/10+budget
       WHERE project_no IN(
         SELECT project_no FROM works_on
         WHERE job='Manager' AND emp_no=10102)
```

S.7.8

```
UPDATE department
SET dept_name='Sales'
WHERE dept_no =
(SELECT dept_no FROM employee
      WHERE emp_lname='James')
```

S.7.9

```
UPDATE works_on
      SET enter_date='12/12/1998'
      WHERE project_no='p1' AND emp_no IN (
               SELECT emp_no FROM employee
              JOIN department ON employee.dept_no = department.dept_no
              WHERE dept_name='Sales')
```

Chapter 8

S.8.1

```
-- This procedure inserts 3,000 rows in the employee table
USE sample
declare @i integer
declare @first_name char(20)
declare @last_name char(20)
declare @department char(4)
```

```
set @i = 1
set @first_name = 'Jane'
set @last_name = 'Smith'
set @department = 'd1'
while @i < 3001
begin
insert into employee
  values (@i, @first_name, @last_name, @department)
set @i = @i+1
end
```

S.8.2

```
declare @i int , @order_id integer
declare @customer_id char(5)
declare @shipped_date datetime
declare @freight money

set @i = 1
set @customer_id = 'ALKHE'
set @shipped_date = getdate()
set @freight = 100.00
while @i < 10001
begin

set @order_id =ceiling (rand ((datepart(mm,getdate())*100000)
+(datepart(ss, GETDATE())*1000)
          + DATEPART(ms,getdate())) * 100000000)
insert into orders (orderid, customerid, shippeddate, freight)
  values( @order_id, @customer_id, @shipped_date, @freight)
set @i = @i+1
end
```

Chapter 9

S.9.1

```
CREATE INDEX i_enterdate
   ON works_on(enter_date)
   WITH FILLFACTOR = 60
```

S.9.2

```
CREATE UNIQUE INDEX i_lfname
    ON employee (emp_lname, emp_fname)
```

A composite index can be used for index access for the leading part of the index. Therefore, there is a significant difference if you change the order of the columns in a composite index.

S.9.3

An index that is implicitly created for the primary key of a table cannot be dropped using the DROP INDEX statement. It can be dropped only if you drop the constraint (using the ALTER TABLE statement with the DROP CONSTRAINT clause).

S.9.4

During index access, only the rows that satisfy the search criteria of the query are accessed. This is in most cases an obvious advantage in relation to a table scan, where the system does not use an index. But besides this significant benefit, index scan can have two disadvantages: In contrast to a table scan, SQL Server uses smaller I/O units to read rows for index access; therefore, a number of read operations will be comparatively higher. The second disadvantage of the index access method (using a nonclustered index) is that data pages must be read repeatedly, because the rows to be selected are scattered on data pages.

S.9.5

```
CREATE INDEX i_employee_lname ON employee (emp_lname)
```

S.9.6

```
CREATE INDEX i_emp_name ON employee (emp_lname, emp_fname)
```

S.9.7

```
CREATE INDEX i_workson_empno ON works_on (emp_no)
CREATE INDEX i_employee_empno ON employee (emp_no)
```

S.9.8

```
CREATE INDEX i_department_deptno ON department (dept_no)
CREATE INDEX i_employee_deptno ON employee (dept_no)
CREATE INDEX i_department_deptname ON department (dept_name)
```

Chapter 10

S.10.1

```
CREATE VIEW v_10_1
 AS SELECT *
 FROM employee WHERE dept_no = 'd1'
```

S.10.2

```
CREATE VIEW v_10_2
 AS SELECT project_no, project_name
   FROM project
```

S.10.3

```
CREATE VIEW v_10_3
 AS SELECT emp_lname, emp_fname
     FROM employee, works_on
     WHERE works_on.emp_no = employee.emp_no
     AND enter_date BETWEEN '06/01/1998' AND '12/31/1998'
```

S.10.4

```
CREATE VIEW v_10_4 (first, last)
 AS SELECT emp_fname, emp_lname
     FROM v_10_3
```

S.10.5

```
SELECT *
 FROM v_10_1 WHERE emp_lname LIKE 'M%'
```

S.10.6

```
CREATE VIEW v_10_6
  AS SELECT project.*
      FROM project, employee, works_on
      WHERE project.project_no = works_on.project_no
      AND employee.emp_no = works_on.emp_no
     AND emp_lname = 'Smith'
```

S.10.7

```
ALTER VIEW v_10_1
 AS SELECT *
 FROM employee WHERE dept_no IN( 'd1','d2')
```

S.10.8

```
DROP VIEW v_10_3.
```

The DROP VIEW statement also removes the view v_10_4.

S.10.9

```
INSERT INTO v_10_2 VALUES('p4','Moon')
```

S.10.10

```
CREATE VIEW v_10_10
 AS SELECT emp_no, emp_fname, emp_lname, dept_no
       FROM employee
      WHERE emp_no  < 10000
      WITH CHECK OPTION
INSERT INTO v_10_10 VALUES(22123, 'Michael' , 'Kohn', 'd3')
       doesn't work, because the employee number is greater than 10,000
```

S.10.11

```
CREATE VIEW v_10_11
 AS SELECT emp_no, emp_fname, emp_lname, dept_no
       FROM employee
      WHERE emp_no  < 10000
INSERT INTO v_10_11 VALUES(22123, 'Michael' , 'Kohn', 'd3')
--        works, because the employee number won't be checked
```

S.10.12

```
CREATE VIEW v_10_12
 AS SELECT emp_no, project_no, enter_date, job
    FROM works_on
    where enter_date between '01.01.1998' and '12.31.1999'
    with check option
```

```
UPDATE v_10_12 SET enter_date = '06/01/1997'
  where emp_no = 29346 and project_no='p1';
--    doesn't work, because the date does not belong to the years 1998 or 1999.
```

S.10.13

```
CREATE VIEW v_10_13
 AS SELECT emp_no, project_no, enter_date, job
    FROM works_on
    where enter_date between '01.01.1998' and '12.31.1999'

UPDATE v_10_12 SET enter_date = '06/01/1997'
  where emp_no = 29346 and project_no='p1';
--         this UPDATE statement works
```

Chapter 11

S.11.1

```
USE master
SELECT filename from sysdatabases
  WHERE name = 'sample'
```

S.11.2

```
USE sample
SELECT sysindexes.name
 FROM sysobjects, sysindexes
 WHERE sysobjects.id = sysindexes.id
 AND sysobjects.name = 'employee'
 AND sysindexes.indid = 1
```

S.11.3

```
USE sample
SELECT COUNT(*)
 FROM sysconstraints, sysobjects
 WHERE sysconstraints.id = sysobjects.id
  AND sysobjects.name = 'employee'
```

S.11.4

```
USE sample
SELECT sysconstraints.status
 FROM syscolumns, sysobjects, sysconstraints
 WHERE syscolumns.id = sysconstraints.colid
 AND sysobjects.id = sysconstraints.id
 AND sysobjects.name = 'employee'
  AND syscolumns.name = 'dept_no'
```

S.11.5

The system procedure **sp_depends** uses the information that is contained in the system table **sysdepends**.

S.11.6

```
SELECT syscolumns.name
      FROM systypes, syscolumns
      WHERE systypes.xtype = syscolumns.xtype
      AND systypes.usertype = syscolumns.usertype
     AND systypes.type = syscolumns.type
    AND systypes.name = 'western_countries'
```

S.11.7

```
USE northwind
SELECT table_name from information_schema.tables
WHERE table_type = 'BASE TABLE'
```

S.11.8

```
USE sample
SELECT column_name, data_type, ordinal_position
    FROM information_schema.columns
   WHERE table_name = 'employee'
```

Chapter 12

S.12.1

In Windows security mode SQL Server exclusively uses Windows user accounts, assuming that they already have been validated at the operating system level (trusted

connection). In Mixed mode, there are two security options: SQL Server security and Windows security.

S.12.2

The login is used to allow a certain user to log in to the SQL Server system, whereas the account is used to grant access to a particular database for a certain user or a role.

S.12.3

```
sp_addlogin 'peter','abc','sample'
GO
sp_addlogin 'paul','def', 'sample'
GO
sp_addlogin 'mary','fgh', 'sample'
GO
use master
SELECT name FROM syslogins
```

S.12.4

```
use sample
GO
sp_grantdbaccess 'peter','s_peter'
GO
sp_grantdbaccess 'paul','s_paul'
GO
sp_grantdbaccess 'mary','s_mary'
```

S.12.5

```
use sample
GO
sp_addrole 'managers'
GO
sp_addrolemember 'managers','s_peter'
GO
sp_addrolemember 'managers','s_mary'
GO
sp_addrolemember 'managers','s_paul'
sp_helpuser 'managers'
```

S.12.6

```
GRANT CREATE TABLE
        TO s_peter

GRANT CREATE PROCEDURE
      TO s_mary
```

S.12.7

```
GRANT UPDATE ON employee(emp_lname,emp_fname)
        TO s_peter
```

S.12.8

```
CREATE VIEW readnames
       AS SELECT emp_lname,emp_fname FROM employee
GO
GRANT SELECT ON readnames
      TO s_peter,s_mary
```

S.12.9

```
GRANT INSERT ON project
      TO managers
```

S.12.10

```
REVOKE SELECT ON readnames
       FROM s_peter
```

S.12.11

```
DENY INSERT ON project
   TO s_mary
```

S.12.12

The functionality of views in relation to the T-SQL statements GRANT, REVOKE, and DENY is limited, because with the former you can restrict only the access to one or more columns and one or more rows. (Using T-SQL statements you can restrict operations on data, such as reading and writing.)

S.12.13

```
USE sample
GO
sp_helpuser s_mary
```

Chapter 13

S.13.1

```
CREATE TRIGGER tr_refint_dept
        ON department
        FOR DELETE, UPDATE
        AS
        IF UPDATE(dept_no)
                BEGIN
                IF (SELECT COUNT(*)
                     FROM employee, deleted
                     WHERE employee.dept_no = deleted.dept_no) >0
                    BEGIN
                    ROLLBACK TRANSACTION
                    PRINT "Transaction failed!"
                    END
         ELSE PRINT "Transaction succeeded"
         END

CREATE TRIGGER tr_refint_dept2
         ON employee
         FOR INSERT, UPDATE
        AS
         IF UPDATE(dept_no)
                BEGIN
                IF (SELECT department.dept_no
                     FROM department, inserted
                     WHERE department.dept_no = inserted.dept_no) IS NULL
                     BEGIN
                     ROLLBACK TRANSACTION
                     PRINT "Transaction failed!"
                    END
               ELSE PRINT "Transaction succeeded"
               END
```

S.13.2

```
CREATE TRIGGER tr_refint_project
ON project
FOR DELETE, UPDATE
  AS
  IF UPDATE(project_no)
  BEGIN
      IF (SELECT COUNT(*)
            FROM works_on, deleted
            WHERE works_on.project_no = deleted.project_no) >0
      BEGIN
ROLLBACK TRANSACTION
PRINT "Transaction failed!"
END
ELSE PRINT "Transaction succeeded"
END
CREATE TRIGGER tr_ref_project2
ON works_on
FOR INSERT, UPDATE
 AS
   IF UPDATE(project_no)
 BEGIN
   IF (SELECT project.project_no
         FROM project, inserted
        WHERE project.project_no = inserted.project_no) IS NULL
BEGIN
ROLLBACK TRANSACTION
PRINT "Transaction failed!"
END
ELSE PRINT "Transaction succeeded"
END
```

S.13.3

First step: Implement the program

```
using System;
using System.Data;
using System.Data.SqlClient;
```

```
using Microsoft.SqlServer.Server;
public class StoredProcedures
{
   public static void WorksOn_Integrity()
   {
      SqlTriggerContext context = SqlContext.TriggerContext;
      if(context.IsUpdatedColumn(0)) //Emp_No
      {
         SqlConnection conn = new SqlConnection("context connection=true");
         conn.Open();
         SqlCommand cmd = conn.CreateCommand();
         cmd.CommandText = @"SELECT employee.emp_no
                     FROM employee, inserted
                     WHERE employee.emp_no = inserted.emp_no";
         SqlPipe pipe = SqlContext.Pipe;
         if(cmd.ExecuteScalar() == null)
         {
            System.Transactions.Transaction.Current.Rollback();
            pipe.Send("No insertion/modification of the row");
         }
         else
            pipe.Send("The row inserted/modified");
      }
   }
}
```

Second step: Compile the program

```
csc /target:library Example13_3.cs
```

Third step: Create the corresponding assembly and create the trigger

```
CREATE ASSEMBLY Example13_3 FROM 'C:\Program\Microsoft SQL
Server\assemblies\Example13_3.dll'
WITH PERMISSION_SET=EXTERNAL_ACCESS
GO
CREATE TRIGGER workson_integrity ON works_on
AFTER INSERT, UPDATE AS
EXTERNAL NAME Example13_3.StoredProcedures.WorksOn_Integrity
```

S.13.4

First step: Implement the program

```
using System;
using System.Data;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;
public class StoredProcedures
 {
  public static void Refint_WorksOn2()
  {
     SqlTriggerContext context = SqlContext.TriggerContext;
     if(context.IsUpdatedColumn(2)) //Emp_No
     {
        SqlConnection conn = new SqlConnection("context
connection=true");
        conn.Open();
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = @"SELECT COUNT(*)
                    FROM WORKS_ON, deleted
                    WHERE works_on.emp_no = deleted.emp_no";
        SqlPipe pipe = SqlContext.Pipe;
        if(Convert.ToInt32(cmd.ExecuteScalar()) > 0)
        {
           System.Transactions.Transaction.Current.Rollback();
           pipe.Send("No deletion/modification of the row");
        }
        else
           pipe.Send("The row deleted/modified");
     }
  }
}
```

Second step: Compile the program

```
csc /target:library Example13_4.cs
```

Third step: Create the corresponding assembly and create the trigger

```
CREATE ASSEMBLY Example13_4 FROM 'C:\Program\Microsoft SQL
Server\assemblies\Example13_4.dll'
      WITH PERMISSION_SET=EXTERNAL_ACCESS
GO
CREATE TRIGGER refint_workson2 ON employee
       AFTER DELETE, UPDATE AS
      EXTERNAL NAME Example13_4.StoredProcedures.Refint_WorksOn2
```

Chapter 14

S.14.1

Transactions are used to keep the data consistent using its "all or nothing" property: Either all statements are (successfully) executed or no one of them is executed.

S.14.2

A distributed transaction needs a coordinator that coordinates the execution of all transaction parts on different servers. Also, you use the BEGIN TRANSACTION statement to start a local transaction and BEGIN DISTRIBUTED TRANSACTION to start a distributed transaction.

S.14.3

Each Transact-SQL statement always belongs either implicitly or explicitly to a transaction. When a session operates in implicit transaction mode, selected statements implicitly issue the BEGIN TRANSACTION statement. This means that you do nothing to start such a transaction. However, the end of each implicit transaction must be explicitly committed or rolled back using the COMMIT (i.e., ROLLBACK) statement. An explicit transaction is specified with the pair of statements BEGIN TRANSACTION and COMMIT TRANSACTION (or ROLLBACK TRANSACTION).

S.14.4

None.

S.14.5

Using the global variable @@error

S.14.6

The SAVE TRANSACTION statement is used to execute parts of an entire transaction.

S.14.7

The advantage of the row-level locking is that it maximizes concurrency, because all other rows of the table that are stored on the same page can be used by other processes. On the other hand, it increases system overhead, because each locked row requires one lock (and you need many more locks if you use row-level locking instead of page-level locking). The advantage of row-level locking is the disadvantage of page-level locking and vice versa.

S.14.8

Using the SET LOCK_TIMEOUT statement, a user can specify whether a transaction should wait or not for a lock to be released. Also, there are several options in the FROM clause of the SELECT statement that can be used by a user to influence locking behavior of SQL Server (such as UPDALOCK, TABLOCK; ROWLOCK, and PAGLOCK).

S.14.9

An intent lock is always placed at a next level in a hierarchy of database objects above the process intents to lock. An X lock, i.e., S lock, is always used for the object that actually should be locked.

S.4.10

The process of converting many page-level locks into one table lock (or many row-level locks into one page lock).

S.14.11

READ UNCOMMITTED is the simplest isolation level and therefore allows the maximum of data inconsistency of all isolation levels. On the other hand, the advantage of this isolation level is that it allows the highest concurrency. The advantage of SERIALIZABLE is that there will be no data inconsistency at all when you apply this isolation level for a process. On the other hand, it decreases concurrency of the processes at most.

S.14.12

A deadlock is a special situation in which two transactions block the progress of each other. (It is possible that several transactions cause a deadlock, if the first transaction blocks the second, the second the third, and so on, and if the last transaction blocks the first one.)

S.14.13

SQL Server always chooses the process that closed the loop in a deadlock. Users can use the SET DEADLOCK_PRIORITY statement to choose the "victim" process.

S.14.14

Pessimistic concurrency control locks resources as they are required, for the duration of a transaction. If a conflict occurs, the database system guaranties data consistency. Optimistic concurrency control allows transactions to execute without using locking mechanism. If a conflict occurs, the application must read the data and attempt the change again.

Chapter 15

S.15.1

An index page is identical to a data page except that it has only two parts:

- Page header
- Data space

(The row offset table does not appear at the end of an index page.)

S.15.2

In the TEMP directory.

S.15.3

Multiple instances have the benefits in the following two areas:

- *Dividing different database types.* (The main purpose of multiple instances is to divide production databases, test databases, and sample databases into different groups to run on different instances.)
- *Server consolidation.* (Instead of having N machines to run N SQL Server systems, you can use one [big] computer and run N instances.)

The disadvantage of multiple instances is that you need a very powerful computer to manage multiple instances.

S.15.4

Multithreading specifies that several threads from different clients are scheduled and execute using one system process.

S.15.5

Data load, backup, and recovery as well as query execution can be executed in parallel using SQL Server.

Chapter 18

S.18.1

The primary filegroup is by default also the default filegroup. The database administrator can set another filegroup to be the default filegroup using the ALTER DATABASE statement.

S.18.2

The **sp_dboption** system procedure is the deprecated feature. For this reason you should use the DATABASEPROPERTYEX function to view database options.

S.18.3

To modify database options use either SQL Server Management Studio or the DBCC command. The use of the **sp_dboption** system procedure is not recommended (see S.18.2).

S.18.4

When the **autoshrink** database option is set, SQL Server implicitly shrinks the size of database files that belong to the database.

S.18.5

No.

Chapter 19

S.19.1

Expand the server, expand the **Security folder**, right-click **Logins**, and select **New Login**. On the **General** tab of the **Login** dialog box, select **SQL Server Authentication**, type the new login name (for example peter1), and type the corresponding password. In the **Defaults** database dialog box, select sample. The new login appears in the detail pane.

S.19.2

In the **Databases** folder expand the database. Expand **Security**, right-click **Users**, and select **New User**. In the **Database User** dialog box select the login name and type the new user name under it. In the **Database role membership** frame, select **public**.

S.19.3

In the **Databases** folder expand the database. Expand **Security**, right-click **Roles**, and select **New Database Role**. In the **Database Role** dialog box type the new role name (managers). In the **Members of the role** frame, click **Add** to add user names. Select the user names that belong to the new role.

S.19.4

Right-click the database and select **Properties**. On the **Permissions** page of the **Database Properties** dialog box in the row with the user name (**s_peter**), check the boxes for the **Create table** and **Create procedure** permissions.

S.19.5

To manage permissions for a user, expand the server and expand **Databases**. Right-click the database and then click **Properties**. In **the Database Properties** dialog box choose the **Permissions** page. Select the user and check the corresponding box in the **Grant** column.

S.19.6

In the **Databases** folder expand the database, expand **Roles**, and expand **Database Roles**. Double-click the role (**managers**) that you want to modify. Select the **Securables** page, mark the table (**project**), and check the box at the intersection of **Grant** and **Insert**.

Chapter 20

S.20.1

The benefit of differential backups is that you save time in the restore process, because to recover a database completely, you need a full database backup and only the *latest* differential backup. If you use transaction logs for the same scenario, you have to apply the full database backup and *all* existing transaction logs to bring the database to a consistent state.

A disadvantage of differential backups is that you cannot use them to recover data to a specific point in time because they do not store intermediate changes to the database.

S.20.2

It depends upon several factors, such as the size of the database, the number of modification operations, and so on.

S.20.3

There is no way to make the differential backup of the master database. You can make only the full database backup of this system database.

S.20.4

One common technique is to configure database data files on a RAID 0 drive and place the transaction log and backups on a mirrored drive (RAID 1). If the data must be quickly recoverable, use RAID 5 for a database and RAID 1 for the corresponding transaction log(s).

S.20.5

Automatic recovery is done by the system, while manual recovery must be initiated by the system administrator.

S.20.6

The RESTORE VERIFYONLY statement.

S.20.7

Using the full recovery model, no work is lost due to a lost or damaged data file, and you can recover to any point in time. On the other hand, the corresponding transaction log may be very voluminous. Using bulk-logged recovery model, you cannot recover to any point in time, but the corresponding transaction log is smaller, because minimal log space is used by bulk operations. The simple recovery model provides the simplest backup strategy, but all changes since the most recent database or differential backup must be manually redone.

Chapter 21

S.21.1

You can automate, among others, the following tasks: data transfer, backing up the database and transaction log, as well as maintaining indices and data integrity.

S.21.2

Create one job to back up your transaction log and specify two schedules.

S.21.3

Create an alert on the SQL Server counter called Locks object counter.

S.21.4

A SQL Server error message contains the following parts: A unique error message number, a severity level number, a line number, which identifies the line where the error occurred, and the error text.

S.21.5

The three most important columns of the **sys.messages** catalog view are **message_id, severity**, and **text**.

Chapter 24

S.24.1

Primary key is required to uniquely identify the rows of the published table. All tables using transactional replication must explicitly contain a primary key.

S.24.2

Partition tables and/or filter data, which you want to replicate.

S.24.3

You can minimize update conflicts by limiting Subscriber update capabilities to an appropriate subset of data.

S.24.4

Log Reader Agent searches for marked transactions and copies them from the transaction log on the publisher to the **distribution** database. It is used for transactional replications. The synchronization job between all sites is done by Merge Agent. It is used for merge replications. Finally, Snapshot Agent generates the schema and data of the published tables during snapshot replication.

Chapter 25

S.25.1

OLTP systems have short transactions, many (possibly hundreds or thousands of) users, continuous read and write operations, and medium-size data. Data warehousing systems

have a small number of users, large size of data stored in a database, and, after the load process, only read operations.

S.25.2

The ER model is highly normalized, while the dimensional model is usually denormalized, because it uses nonredundant data. Also the ER model produces very complex database design for large databases.

S.25.3

Extracting specifies the process of loading source data from multiple, heterogeneous operational systems in a temporary staging area. Data transformation is the process of formatting and modifying data that is extracted from various sources to make the information more useful. Finally, during the load process, the cleaned data is loaded into the data warehouse.

S.25.4

In a dimensional model there is one fact table and many dimension tables. A fact table contains a very large amount of the data stored in a data warehouse (about 70%). Also, columns of a fact table are numeric and additive.

S.25.5

The MOLAP structure offers the best query performance for data analysis, because the aggregations and a copy of the base data are stored in a multidimensional structure allowing the high-speed query processor to retrieve data quickly. However, data in MOLAP is duplicated in the cube and consumes the most storage space.

The ROLAP structure allows you to use standard Transact-SQL statements to query against the relational tables. It also eliminates data duplication and does not require extra storage space. However, the query performance is not as fast as with the MOLAP and HOLAP structure.

The HOLAP structure retrieves data in the cube quickly and consumes less storage space than MOLAP.

S.25.6

Because of the large amount of the data that is stored in a data warehouse. In that case, doing aggregations on the fly requires a significant amount of time.

Chapter 28

S.28.1

The query for this report is:

```
SELECT w.emp_no, emp_lname
    FROM employee e, works_on w
   WHERE e.emp_no = w.emp_no
    AND w.job = 'Clerk'
```

S.28.2

The query for this report is:

```
USE sample
 SELECT budget
     FROM employee e, project p, department d, works_on w
    WHERE w.project_no = p.project_no
   AND d.dept_no = e.dept_no
   AND e.emp_no = w.emp_no
   AND d.dept_name = 'Research'
   AND e.emp_no < 25000
```