練習問題の解答

第1章

解答 1.1

データの独立性とは、データベースアプリケーションプログラムが、データベースに格納されているデータの物理的な構造に依存しないこと（データの物理的独立性）と、データベースの論理的な構造に依存しないこと（データの論理的独立性）を意味します。

解答 1.2

リレーショナルモデルの中心となる概念は関係（すなわち、テーブル）です。リレーショナルモデル自体は列数が 1 列以上、行数が 0 行以上のテーブルの集まりにすぎません。このテーブル内の各位置にそれぞれデータ値がただ 1 つだけ格納されます。

解答 1.3

employee テーブルは従業員の実体を表し、Ann Jones のデータはオブジェクト（すなわち、実体のインスタンス）を表します。

解答 1.4

works_on テーブルは、従業員とプロジェクトの間の関係を表します。sample データベースの他のテーブルとの違いは、このテーブルが 2 つの実体の間の関係を表すのに対し、他のテーブルはそれぞれ特定の実体を表すことです。

解答 1.5

A. はい。なぜなら、title（書名）は一意であり、テーブルの行を一意に識別できるからです。

B. はい。なぜなら、書名によってそれに対応する ISBN が一意に決まるからです。

C. はい（book テーブルでは、非キー属性の間に依存関係がありません）。

解答 1.6

A. はい。なぜなら、他のすべての列がこの列に依存しているからです。

B. いいえ。なぜなら、customer_no 列は order_no 列に関数従属しているからです。

解答 1.7

company テーブルはいずれの正規形でもありません。location 列が複数値属性（繰り返し項目）を持つからです。

解答 1.8

A. 1NF です。city 列が supplier_no 列に関数従属していて、supplier_no 列がテーブルの部分キーだからです。

B. 次の 2 つのテーブルを使います。

- supplier1(supplier_no, article)
- supplier2(supplier_no, city)

解答 1.9

1NF です。C 列が B 列に依存し、B 列がテーブルの部分キーだからです。

解答 1.10

このテーブルは 3NF ですが、良い設計とは言えません。

第 2 章

解答 2.1

手順は、次のとおりです。

1. コンソールツリーから［データベース］フォルダを右クリックし、［新しいデータベース］を選択します。
2. ［新しいデータベース］ダイアログボックスから［全般］ページを選択します。
3. ［全般］ページの［名前］フィールドに「test」と入力します。
4. ［データベースファイル］フレームの［論理名］列でデータファイルの名前を「test」から「testdata_a」に変更し、［初期サイズ（MB)］列を 10 に変更します。
5. ［自動拡張］列にある［...］ボタンをクリックし、表示されたダイアログボックスから、［自動拡張を有効にする］チェックボックスをオンにし、［ファイル拡張］内の［MB 単位］フィールドに値 2 を、［ファイルの最大サイズ］フレーム内の［ファイル拡張の制限（MB)］フィールドに 20 を入力します。
6. ［OK］ボタンをクリックします。

解答 2.2

手順は、次のとおりです。

1. ［データベース］フォルダ内で test データベースを右クリックして［プロパティ］を選択します。
2. test データベースのプロパティダイアログボックスが開いたら、［ファイル］ページを選択します。
3. ［データベースファイル］フレームからトランザクションログファイル（［ファイルの種類］列の値が「ログ」となっている行）の［初期サイズ（MB)］列の値を選択し、10 に変更します。
4. 練習 2.1 と同じようにして自動拡張の設定を行います。

解答 2.3

test データベースのプロパティダイアログボックスで［オプション］ページを選択し、［その他のオプション］セクションにある［アクセスの制限］の値を「Restricted」に設定します。この設定では、db_owner 固定データベースロール、dbcreator 固定サーバーロール、sysadmin 固定サーバーロールのメンバだけがそのデータベースにアクセスできるようになります。同時にアクセスできるメンバの数に制限はありません。

解答 2.4

test データベース内の［テーブル］フォルダを右クリックして［新しいテーブル］を選択します。［新しいテーブル］ダイアログボックスで、すべての列の名前と、その列のデータ型および NULL 値の許容を設定します。設定が終わったら、［新しいテーブル］ダイアログボックスを閉じます。ダイアログボックスを閉じると［名前の選択］ダイアログボックスが表示されるので、テーブルの名前を入力します。以降、この手順を残り 3 つのすべてのテーブルについて繰り返します。

解答 2.5

コンソールツリーから［データベース］フォルダ→［AdventureWorks］データベースの順に展開し、［テーブル］フォルダをクリックします。データベースのすべてのテーブルが詳細ペインに表示されます。Person.Address テーブル（［名前］列に「Address」、［スキーマ］列に「Person」と表示される項目）を右クリックして［プロパティ］を選択します。

解答 2.6

同じ名前で別のデータベースを作成することはできないため、以下のメッセージが表示されます。

```
メッセージ 1801、レベル 16、状態 3、行 1
データベース 'test' は既に存在します。
```

解答 2.7

［ファイル］メニューから［名前を付けて保存］を選択します。その後、保存先のディレクトリに移り、新しいファイル名（createdb）を入力し、［保存］ボタンをクリックします。

解答 2.8

クエリエディタに「USE test」と入力して実行するか、ツールバーのドロップダウンリストボックスから test データベースを選択します。

解答 2.9

ツールバーのドロップダウンリストボックスから AdventureWorks データベースを選択し、クエリエディタに「SELECT * FROM Sales.Customer」と入力します。［F5］キー（あるいはツールバーの［実行］ボタン）を押してこのステートメントを実行した後、ツールバーから赤い四角のボタンを押して現在の実行を中断します。

第 3 章

解答 3.1

記憶サイズと値の範囲が違います。TINYINT は 1 バイトで格納され、値の範囲は 0 ～ 255 です。SMALLINT は 2 バイトで格納され、値の範囲は -32768 ～ 32767 です。INT は 4 バイトで格納され、値の範囲は -2147483648 ～ 2147483647 です。

解答 3.2

CHAR は文字列で、最大 8,000 文字まで格納できます。その文字列は宣言時の長さで格納されます。VARCHAR も最大 8,000 文字まで格納できますが、文字列は実際の長さで格納されます。列のデータ値がほぼ同じサイズなら CHAR を使い、サイズがかなり変動するようであれば VARCHAR を使います。

解答 3.3

日本語版 SQL Server では既定で「yyyy/mm/dd」という形式の日付値を入力できますが、日付の入力形式を明示的に設定する場合には、次のステートメントを実行します。

```
SET DATEFORMAT ymd
```

解答 3.4

```
SELECT DB_ID('AdventureWorks')
```

解答 3.5

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

解答 3.6

```
(01000101)
(11011011)
(00000110)
```

解答 3.7

A + NULL	→	A が何であれ、結果は NULL
NULL = NULL	→	結果は NULL
B OR NULL	→	B が TRUE なら TRUE、そうでなければ NULL
B AND NULL	→	B が FALSE なら FALSE、そうでなければ NULL

解答 3.8

QUOTED_IDENTIFIER オプションが OFF のときです。

解答 3.9

区切り識別子は、予約済みキーワードを識別子として使えるようにする特別な種類の識別子です。テーブル名に空白が含まれるときにもこれを使います。

第 4 章

解答 4.1

```
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)
```

解答 4.2

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

解答 4.3

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

解答 4.4

主キーを構成する列には NOT NULL を指定する必要があります。例 4.3 の場合、employee テーブルの emp_no 列、department テーブルの dept_no 列、project テーブルの project_no 列、works_on テーブルの emp_no 列と project_no 列はいずれも主キーであるため、NOT NULL を指定します。

解答 4.5

dept_no 列と project_no 列が CHAR 型として定義されているのは、数値以外に文字列が含まれることもあるからです。

解答 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 INT 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 INT NULL)
```

解答 4.7

```
ALTER TABLE orders
   ADD shipregion INT NULL
```

解答 4.8

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

解答 4.9

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

解答 4.10

DROP TABLE でテーブルを削除すると、そのテーブルに属しているデータ、インデックス、トリガもすべて削除されます。これと対照的に、そのテーブルを使って定義されているビューは削除されません。

解答 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 INT 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 INT NULL,
    CONSTRAINT prim_ord PRIMARY KEY(orderid),
    CONSTRAINT foreign_orders FOREIGN KEY(customerid)
       REFERENCES customers(customerid))
```

解答 4.12

コンソールツリーから［データベース］フォルダ→ test_db データベース→［テーブル］フォルダの順で展開します。次に、orders テーブルを右クリックして［テーブルを開く］を選択します。すると、ドキュメントウィンドウに orders テーブルの内容が表示されます（この時点では、まだ orders テーブルにデータが入っていません）。ここから、テーブルにデータを挿入できますが、この練習問題の場合、orders テーブルに customers テーブルへの外部キー制約が定義されているため、orders テーブルにデータを挿入するには、まず customers テーブルに適切なデータを挿入しなければなりません。

解答 4.13

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

解答 4.14

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

解答 4.15

```
/* 新規のデータ型を作成する */
sp_addtype Western_Countries, 'CHAR(2)', 'NOT NULL'
GO
/* 新規のデータ型の新しい既定値（デフォルト）を作成する */
CREATE DEFAULT western_countries_default AS 'CA'
GO
/* 新規のデータ型に新しい既定値をバインドする */
sp_bindefault 'western_countries_default', 'Western_Countries'
GO
/* 新規のデータ型に許される値のルールを作成する */
CREATE RULE western_countries_rule
   AS @selection IN ('CA', 'WA', 'OR', 'NM')
GO
/* 新規のデータ型に新しいルールをバインドする */
sp_bindrule western_countries_rule, Western_Countries
GO
/* ユーザー定義のデータ型を使って新規のテーブルを作成する */
CREATE TABLE regions
   (city CHAR(25) NOT NULL,
    country Western_Countries)
```

解答 4.16

```
sp_helpconstraint orders
```

解答 4.17

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

このステートメントはうまくいきません。ordersテーブル内で定義されている外部キー制約foreign_ordersから主キー制約prim_custが参照されているからです。

解答 4.18

```
ALTER TABLE orders
   DROP CONSTRAINT foreign_orders

ALTER TABLE orders
   DROP CONSTRAINT prim_ord

ALTER TABLE customers
   DROP CONSTRAINT prim_cust
```

解答 4.19

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

第5章

解答 5.1

```
SELECT *
   FROM works_on
```

解答 5.2

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

解答 5.3

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

または

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

解答 5.4

```
SELECT emp_no
   FROM works_on
   WHERE enter_date NOT BETWEEN '1998.01.01' AND '1998.12.31'
```

解答 5.5

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

解答 5.6

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

解答 5.7

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

解答 5.8

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

解答 5.9

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

解答 5.10

```
SELECT emp_lname, emp_fname
   FROM employee
   WHERE emp_no IN
   (SELECT emp_no
      FROM works_on
      WHERE enter_date = '1998.01.04')
```

解答 5.11

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

解答 5.12

他の指定（集計やHAVING句）を伴わないGROUP BYは、DISTINCTとちょうど同じ働きをします（テーブルの行をグループに分け、グループごとに1行を返します）。

解答 5.13

すべての NULL 値が 1 つのグループに属します。しかし、これは NULL 値が本質的に値である、ということを意味するものではありません。NULL 値は他の NULL 値とは比べられません。

解答 5.14

COUNT(*colomn*) は引数（列または式）を取り、その引数の NULL でないすべての出現数（オカレンス）を表示します。COUNT(*) は、列に NULL 値が含まれるかどうかに関係なくすべての行をカウントします。

解答 5.15

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

解答 5.16

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

解答 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'))
```

解答 5.18

内側の SELECT ステートメントを比較演算子（「=」など）と組み合わせることができるのは、その結果セットが 1 行以下の場合です。今回の問題の結果セットは複数行になるので、「=」演算子を IN 演算子に置き換える必要があります。

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

解答 5.19

一時テーブルは、複雑なクエリの中間結果の格納に使うことができます。

解答 5.20

ローカル一時テーブルは、その一時テーブルを作成したユーザーだけが利用でき、そのユーザーのセッションの最後に削除されます。グローバル一時テーブルは、すべてのユーザーが利用でき、その一時テーブルを作成したユーザーのセッションが終了した後、その一時テーブルを参照する他のユーザーがいなくなると削除されます。

第 6 章

解答 6.1

SQL Server 結合構文：

```
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
```

ANSI 結合構文：

```
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
```

解答 6.2

最低 n-1 個の結合条件が必要です。

解答 6.3

SQL Server 結合構文：

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

ANSI 結合構文：

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

解答 6.4

SQL Server 結合構文：

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

ANSI 結合構文：

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

解答 6.5

SQL Server 結合構文：

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

ANSI 結合構文：

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

解答 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)
```

解答 6.7

SQL Server 結合構文：

```
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'
```

ANSI 結合構文：

```
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'
```

解答 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)
```

解答 6.9

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

解答 6.10

SQL Server 結合構文：

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

ANSI 結合構文：

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

相関サブクエリ：

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

第7章

解答 7.1

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

解答 7.2

```
CREATE TABLE emp_d1_d2
   (emp_no INT 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')
```

または

```
SELECT emp_no, emp_fname, emp_lname, dept_no
   INTO emp_d1_d2a
   FROM employee
   WHERE dept_no IN ('d1', 'd2')
```

解答 7.3

```
CREATE TABLE employee_three
   (emp_no INT 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 '1998.01.01' AND '1998.12.31')
```

解答 7.4

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

解答 7.5

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

解答 7.6

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

解答 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)
```

解答 7.8

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

解答 7.9

```
UPDATE works_on
   SET enter_date = '1998.12.12'
   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')
```

第8章

解答 8.1

```
DECLARE @emp_no INT
SET @emp_no = 1
WHILE @emp_no < 3001
BEGIN
   INSERT INTO employee3 (emp_no, emp_fname, emp_lname, dept_no)
      VALUES(@emp_no, 'Jane', 'Smith', 'd1')
   SET @emp_no = @emp_no+1
END
```

解答 8.2

```
DECLARE @i INT
DECLARE @emp_no INT
SET @i = 0
SET @emp_no = (CONVERT(INT, (RAND() * 10000)))
WHILE @i < 1000
BEGIN
   WHILE (SELECT COUNT(*) FROM employee WHERE emp_no = @emp_no) > 0
   BEGIN
      SET @emp_no = (CONVERT(INT, (RAND() * 10000)))
   END
   INSERT INTO employee VALUES(@emp_no, 'Jane', 'Smith', 'd1')
   SET @i = @i + 1
END
```

第9章

解答 9.1

```
CREATE INDEX i_enterdate
   ON works_on(enter_date)
   WITH FILLFACTOR = 16
```

解答 9.2

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

複合インデックスによるインデックスアクセスではインデックスの先頭部分が関係するので、複合インデックスの列の順序を変更すると、インデックスの使われ方に大きな変化が生じます。

解答 9.3

テーブルの主キーに対して暗黙に作られたインデックスを DROP INDEX ステートメントで削除することはできません。ALTER TABLE ステートメントの DROP CONSTRAINT 句を使って主キー制約を削除した場合にだけ削除できます。

解答 9.4

インデックスアクセスでは、クエリの検索条件を満たす行だけがアクセスされます。つまり、インデックスを使わないテーブルスキャンと比べて、たいていは確かに有利です。しかし、このように明らかに有利なインデックスアクセスにも次の２つの欠点があります。第１に、テーブルスキャンとは異なり、インデックスアクセスによる行の読み取りにはより小さな I/O 単位が使われるので、読み取り操作の回数が比較的多くなります。インデックスアクセス（特に非クラスタ化インデックスを使う場合）の第２の欠点は、選択すべき行がいくつかのデータページに散らばって存在するので、データページを何度も読まなければならないことです。

解答 9.5

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

解答 9.6

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

解答 9.7

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

解答 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)
```

第 10 章

解答 10.1

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

解答 10.2

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

解答 10.3

```
CREATE VIEW v_10_3
   AS SELECT emp_lname, emp_fname
         FROM employee JOIN works_on
         ON works_on.emp_no = employee.emp_no
         AND enter_date BETWEEN '1998.06.11' AND '1998.12.31'
```

解答 10.4

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

解答 10.5

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

解答 10.6

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

解答 10.7

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

解答 10.8

```
DROP VIEW v_10_3
```

この DROP VIEW ステートメントによって、v_10_4 ビューが無効になります。

解答 10.9

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

解答 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
```

次に示す INSERT ステートメントは、従業員番号が 10,000 より大きいので失敗します。

```
INSERT INTO v_10_10 VALUES(22123, 'David', 'Kohn', 'd3')
```

解答 10.11

```
CREATE VIEW v_10_11
   AS SELECT emp_no, emp_fname, emp_lname, dept_no
         FROM employee
         WHERE emp_no < 10000
```

次に示す INSERT ステートメントは、従業員番号がチェックされないので正常に実行されます。

```
INSERT INTO v_10_10 VALUES(22123, 'David', 'Kohn', 'd3')
```

解答 10.12

```
CREATE VIEW v_10_12
   AS SELECT emp_no, project_no, enter_date, job
         FROM works_on
         WHERE enter_date BETWEEN '1998.01.01' AND '1999.12.31'
         WITH CHECK OPTION
```

次に示す UPDATE ステートメントは、日付が 1998 年と 1999 年のどちらにも属さないので失敗します。

```
UPDATE v_10_12 SET enter_date = '1997.06.01'
   WHERE emp_no = 29346 AND project_no = 'p1';
```

解答 10.13

```
CREATE VIEW v_10_12
   AS SELECT emp_no, project_no, enter_date, job
         FROM works_on
         WHERE enter_date BETWEEN '1998.01.01' AND '1999.12.31'
```

次に示す UPDATE ステートメントは、プロジェクト参加日がチェックされないので正常に実行されます。

```
UPDATE v_10_12 SET enter_date = '1997.06.01'
   WHERE emp_no = 29346 AND project_no = 'p1'
```

第 11 章

解答 11.1

```
USE sample
SELECT physical_name
  FROM sys.database_files
```

解答 11.2

```
USE sample
SELECT name
   FROM sys.indexes
   WHERE object_id = OBJECT_ID('employee')
   AND type = 1
```

解答 11.3

```
USE sample
SELECT COUNT(*)
   FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
   WHERE TABLE_NAME = 'employee'
```

解答 11.4

sp_depends システムストアドプロシージャを使うと、テーブル、ビュー、トリガ、ストアドプロシージャの間の従属情報を表示できます。

解答 11.5

```
USE AdventureWorks
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
   WHERE TABLE_TYPE = 'BASE TABLE'
```

解答 11.6

```
USE sample
SELECT COLUMN_NAME, DATA_TYPE, ORDINAL_POSITION
   FROM INFORMATION_SCHEMA.COLUMNS
   WHERE TABLE_NAME = 'employee'
```

第 12 章

解答 12.1

Windows 認証モードでは、Windows 認証だけを利用できる認証モードです。Windows 認証では、オペレーティングシステムのレベルで認証が済んでいると仮定して Windows のユーザーアカウントだけを使います。混合モードでは、Windows 認証に加えて SQL Server 認証を使った接続も可能になります。SQL Server 認証では、既にオペレーティングシステムにログオンしているユーザーが、さらに SQL Server 内で定義したログインアカウントを使って SQL Server に接続します。

解答 12.2

ログインは、SQL Server システムにアクセスするためのアカウントです。データベースユーザーは、特定のデータベースへアクセスするためのアカウントです。

解答 12.3

```
sp_addlogin 'ann', 'abc', 'sample'
GO
sp_addlogin 'burt', 'def', 'sample'
GO
sp_addlogin 'chuck', 'fgh', 'sample'
GO

SELECT name FROM sys.sql_logins
```

解答 12.4

```
USE sample
GO
sp_grantdbaccess 'ann', 's_ann'
GO
sp_grantdbaccess 'burt', 's_burt'
GO
sp_grantdbaccess 'chuck', 's_chuck'
```

解答 12.5

```
USE sample
GO
CREATE ROLE 'managers'
GO
sp_addrolemember 'managers', 's_ann'
GO
sp_addrolemember 'managers', 's_burt'
GO
sp_addrolemember 'managers', 's_chuck'
GO
sp_helpuser 'managers'
```

解答 12.6

```
GRANT CREATE TABLE
   TO s_ann
GRANT CREATE PROCEDURE
   TO s_burt
```

解答 12.7

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

解答 12.8

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

解答 12.9

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

解答 12.10

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

解答 12.11

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

解答 12.12

ビューは、Transact-SQL ステートメントの GRANT ／ REVOKE ／ DENY に比べると機能が限られます。ビューでは、アクセス範囲を行と列で絞り込むことしかできません（Transact-SQL ステートメントでは、読み取りや書き込みといったデータ操作を制限できます）。

解答 12.13

```
USE sample
GO
sp_helpuser s_burt
```

第 13 章

解答 13.1

```
CREATE TRIGGER tr_refint_emp
   ON employee
   FOR DELETE, UPDATE
   AS
   IF (SELECT COUNT(*)
          FROM works_on JOIN deleted
          ON works_on.emp_no = deleted.emp_no) > 0
      BEGIN
         ROLLBACK TRANSACTION
         PRINT 'Transaction failed!'
      END
   ELSE PRINT 'Transaction succeeded.'
GO
CREATE TRIGGER tr_refint_emp2
   ON works_on
   FOR INSERT, UPDATE
   AS
   IF (SELECT employee.emp_no
          FROM employee JOIN inserted
          ON employee.emp_no = inserted.emp_no) IS NULL
      BEGIN
         ROLLBACK TRANSACTION
         PRINT 'Transaction failed!'
      END
   ELSE PRINT 'Transaction succeeded.'
```

解答 13.2

```
CREATE TRIGGER tr_refint_project
   ON project
   FOR DELETE, UPDATE
   AS
   IF (SELECT COUNT(*)
          FROM works_on JOIN deleted
          ON works_on.project_no = deleted.project_no) > 0
      BEGIN
         ROLLBACK TRANSACTION
         PRINT 'Transaction failed!'
      END
   ELSE PRINT 'Transaction succeeded.'
GO
CREATE TRIGGER tr_refint_project2
   ON works_on
```

```
FOR INSERT, UPDATE
AS
IF (SELECT project.project_no
       FROM project JOIN inserted
       ON project.project_no = inserted.project_no) IS NULL
   BEGIN
      ROLLBACK TRANSACTION
      PRINT 'Transaction failed!'
   END
ELSE PRINT 'Transaction succeeded.'
```

解答 13.3

ステップ 1：プログラムの実装

```
using System;
using System.Data.SqlClient;
using System.Transactions;
using Microsoft.SqlServer.Server;
public class StoredProcedures
{
  public static void WorksOn_Integrity()
  {
    SqlTriggerContext context = SqlContext.TriggerContext;
    SqlConnection conn = new SqlConnection("Context
Connection=true");
    conn.Open();
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = @"SELECT employee.emp_no
                        FROM employee JOIN inserted
                        ON employee.emp_no = inserted.emp_no";
    SqlPipe pipe = SqlContext.Pipe;
    using (TransactionScope transScope = new TransactionScope())
    {
      if(cmd.ExecuteScalar() == null)
      {
        pipe.Send("No insertion/modification of the row.");
        transScope.Dispose();
      }
      else
      {
        pipe.Send("The row inserted/modified.");
        transScope.Complete();
      }
    }
  }
}
```

ステップ 2：プログラムのコンパイル

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

ステップ 3：アセンブリとトリガの作成

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

解答 13.4

ステップ 1：プログラムの実装

```
using System;
using System.Data.SqlClient;
using System.Transactions;
using Microsoft.SqlServer.Server;
public class StoredProcedures
{
  public static void Refint_WorksOn2()
  {
    SqlTriggerContext context = SqlContext.TriggerContext;
    SqlConnection conn = new SqlConnection("Context
Connection=true");
    conn.Open();
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = @"SELECT COUNT(*)
                        FROM works_on JOIN deleted
                        ON works_on.emp_no = deleted.emp_no";
    SqlPipe pipe = SqlContext.Pipe;
    using (TransactionScope transScope = new TransactionScope())
    {
      if(Convert.ToInt32(cmd.ExecuteScalar()) > 0)
      {
        pipe.Send("No deletion/modification of the row.");
        transScope.Dispose();
      }
      else
      {
        pipe.Send("The row deleted/modified.");
        transScope.Complete();
      }
    }
  }
}
```

ステップ 2：プログラムのコンパイル

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

ステップ 3：アセンブリとトリガの作成

```
CREATE ASSEMBLY Example13_4
   FROM 'C:¥Program¥Microsoft SQL Server¥assemblies¥Example13_4.dll'
   WITH PERMISSION_SET=SAFE
GO
CREATE TRIGGER workson_integrity2 ON employee
AFTER DELETE, UPDATE
AS EXTERNAL NAME Example13_4.StoredProcedures.Refint_WorksOn2
```

第 14 章

解答 14.1

トランザクションはデータの一貫性を保つためのしくみであり、「all or nothing」の特性――つまり、すべてのステートメントが（正常に）実行されるか、どのステートメントも実行されないか、のどちらかになること――によって実現されます。

解答 14.2

分散トランザクションでは、複数のサーバーに分散したトランザクションの各部を調整するコーディネータが必要です。ローカルトランザクションは BEGIN TRANSACTION ステートメントで始まり、分散トランザクションは BEGIN DISTRIBUTED TRANSACTION ステートメントで始まります。

解答 14.3

それぞれの Transact-SQL ステートメントは必ず特定のトランザクションに暗黙的または明示的に属します。セッションが暗黙のトランザクションモードのとき、ステートメントは BEGIN TRANSACTION ステートメントを暗黙に発行しますが、そのトランザクションの最後は COMMIT（または ROLLBACK）ステートメントで明示的にコミットまたはロールバックしなければなりません。明示的トランザクションは、BEGIN TRANSACTION と COMMIT TRANSACTION（または ROLLBACK TRANSACTION）のペアで指定します。

解答 14.4

排他ロックと互換性のあるロックはありません。

解答 14.5

@@ERROR を使います。

解答 14.6

SAVE TRANSACTION ステートメントは、トランザクションを部分的に実行するときに使います。

解答 14.7

行レベルのロックでは、同じページに格納されている他のすべての行を他のプロセスが使用できるので、同時実行性が向上します。一方、行ごとにロックが必要となるため、システムのオーバーヘッドが増えます。行レベルのロックの利点はページレベルのロックの欠点であり、ページレベルのロックの利点は行レベルのロックの欠点です。

解答 14.8

SET LOCK_TIMEOUT ステートメントを使うことで、ロックの解放をトランザクションに待たせるかどうかを指定できます。また、SELECT ステートメントの FROM 句のオプションには SQL Server のロックの動作を変更するユーザーオプション（UPDLOCK ／ TABLOCK ／ ROWLOCK ／ PAGLOCK など）があります。

解答 14.9

インテントロックは、データベースオブジェクトの階層内でプロセスがロックしようとするレベルの上のレベルにかけられます。共有ロックと排他ロックは、実際にロックされるオブジェクトにかけられます。

解答 14.10

複数のページレベルのロックを 1 つのテーブルロックに、または複数の行レベルのロックを 1 つのページロックに変換することです。

解答 14.11

READ UNCOMMITTED は一番簡単な分離レベルであり、すべての分離レベルの中でデータの整合性が最も低くなる可能性があります。一方、この分離レベルの利点は、同時実行性が最大になることです。SERIALIZABLE の利点は、データの不整合がまったく生じないことです。一方、プロセスの同時実行性は最も低くなります。

解答 14.12

デッドロックとは、2 つのトランザクションが互いに相手の進行を阻止するという特別な状況を指します（1 番目のトランザクションが 2 番目を阻止し、2 番目が 3 番目、... そして最後のトランザクションが最初のトランザクションを阻止するというように 2 つ以上のトランザクションがデッドロックを引き起こす可能性もあります）。

解答 14.13

既定では、ロールバックのコストが最も低いトランザクションが犠牲者として選択されます。SET DEADLOCK_PRIORITY ステートメントを使用すれば、「犠牲者」の選択動作を設定できます。

解答 14.14

ペシミスティック同時実行制御では、トランザクションの実行中、必要なリソースにロックをかけます。これによって、リソース競合が発生した場合でもデータの一貫性が保たれます。オプティミスティック同時実行制御では、ロックを使わずにトランザクションを実行します。データを変更するときにリソース競合を検査し、競合が検知された場合は、アプリケーションが再度データを読み込んで変更をやり直します。

第 15 章

解答 15.1

インデックスページは、次の 2 つの部分だけから成る点を除けば、基本的にはデータページと同じです。

- ページヘッダー
- データ領域

インデックスページの最後に行オフセットテーブルはありません。

解答 15.2

一時テーブルは、tempdb データベース内に作成されます。

解答 15.3

複数のインスタンスの利点は次の 2 つです。

- データベースをタイプ別（実稼働データベース、テストデータベース、サンプルデータベースなど）に分けることができる
- サーバーを統合できる（n 台のコンピュータを使って n 個の SQL Server システムを実行する代わりに、1 台の大きなコンピュータで n 個のインスタンスを実行できる）

インスタンスを複数作成する際の欠点は、すべてのインスタンスを管理するために強力なコンピュータが必要となることです。

解答 15.4

マルチスレッドとは、各クライアントからの複数のスレッドが 1 つのシステムプロセスでスケジュールされ実行されることです。

解答 15.5

データのロード、バックアップ／復元、クエリの実行、インデックス操作などを並列化できます。

第 18 章

解答 18.1

既定のファイルグループを明示的に指定しなければ、プライマリファイルグループが既定のファイルグループになります。データベース管理者は ALTER DATABASE ステートメントを使って、他のファイルグループを既定のファイルグループにすることができます。

解答 18.2

sp_dboption は旧バージョンとの互換性のためにサポートされているだけで、SQL Server 2005 で積極的に使用すべきものではありません。データベースオプションを表示するときは、DATABASEPROPERTYEX 関数を使用する方がよいでしょう。

解答 18.3

データベースオプションを変更するときは、SQL Server Management Studio または DBCC コマンドを使用してください。sp_dboption システムストアドプロシージャを使用するのはお勧めできません（解答 18.2 を参照）。

解答 18.4

データベースに自動圧縮オプションが設定されている場合に、データベースファイルのサイズが自動的に縮小されます。この設定は、DATABASEPROPERTYEX 関数の IsAutoShrink プロパティで調べることができます。

解答 18.5

データファイルとトランザクションログファイルを同じファイルにすることはできません。

第 19 章

解答 19.1

コンソールツリーからサーバーの［セキュリティ］フォルダを展開します。［ログイン］フォルダを右クリックし［新しいログイン］を選択します。［ログイン－新規作成］ダイアログボックスが表示されたら、［全般］ページを選択します。次に［SQL Server 認証］ラジオボタンを選択し、［ログイン名］ボックスに新たに作成するログインの名前（peter1 など）を、［パスワード］ボックスと［パスワードの確認入力］ボックスにそのログインのパスワードを入力します。［既定のデータベース］リストボックスでは、このログインの既定のデータベースとして sample データベースを選択します。すべての設定が終わったら、［OK］ボタンをクリックします。以上の手順を peter1、paul1、mary1

について繰り返します。
作成したログインを確認するには、コンソールツリーから［セキュリティ］フォルダを展開し、［ログイン］フォルダをクリックします。詳細ペインにログインの一覧が表示されます。

解答 19.2

コンソールツリーから［データベース］フォルダ→ sample データベース→［セキュリティ］フォルダの順で展開します。［ユーザー］フォルダを右クリックし［新しいユーザー］を選択します。［データベースユーザー－新規］ダイアログボックスが表示されたら、［全般］ページを選択します。次に［ログイン名］ボックスに練習 19.1 で作成したログインの名前（peter1 など）を入力し、［ユーザー名］ボックスに新たに作成するデータベースユーザーの名前（s_peter など）を入力します。すべての設定が終わったら、［OK］ボタンをクリックします。以上の手順を peter1、paul1、mary1 について繰り返します。

解答 19.3

コンソールツリーから［データベース］フォルダ→ sample データベース→［セキュリティ］フォルダの順で展開します。［ロール］フォルダを右クリックし、［新規作成］→［新しいデータベースロール］の順で選択します。［データベースロール－新規］ダイアログボックスが表示されたら、［全般］ページを選択します。［ロール名］ボックスに新しいロールの名前（managers）を入力します。［このロールのメンバ］セクションで［追加］ボタンをクリックし、練習 19.2 で作成したユーザーを追加します。すべての設定が終わったら、［OK］ボタンをクリックします。以上の手順を s_peter、s_paul、s_mary について繰り返します。
作成した managers ロールとそのメンバに関する情報を表示するには、まずコンソールツリーから［セキュリティ］フォルダ→［ロール］フォルダ→［データベースロール］フォルダの順で展開します。次に、managers ロールをダブルクリックして表示される［データベースロールのプロパティ］ダイアログボックスから［全般］ページを選択すると、［このロールのメンバ］セクションでロールに所属するメンバを確認できます。

解答 19.4

コンソールツリーから sample データベースを右クリックして［プロパティ］を選択し、表示された［データベースのプロパティ］ダイアログボックスから［権限］ページを選択します。
s_peter にテーブルの作成権限を与えるには、［ユーザーまたはロール］セクションから s_peter の行を選択し、［s_peter の明示的な権限］セクションで［Create table］権限の［許可］列にチェックを入れ、［OK］ボタンをクリックします。
s_mary にストアドプロシージャの作成権限を与えるには、［ユーザーまたはロール］セクションから s_mary の行を選択し、［s_mary の明示的な権限］セクションで［Create procedure］権限の［許可］にチェックを入れ、［OK］ボタンをクリックします。

解答 19.5

次の手順で可能です。

1. コンソールツリーから sample データベース→［セキュリティ］フォルダの順に展開し、［ユーザー］フォルダを選択します。
2. 詳細ペインで s_peter を右クリックし、［プロパティ］を選択します。

3. ［データベースユーザー］ダイアログボックスで、［セキュリティ保護可能なリソース］ページを選択し、［セキュリティ保護可能なリソース］セクションの［追加］ボタンをクリックします。
4. ［オブジェクトの追加］ダイアログボックスが表示されたら、［特定のオブジェクト］ラジオボタンを選択し、［OK］ボタンをクリックします。
5. ［オブジェクトの選択］ダイアログボックスが表示されたら、［オブジェクトの種類］ボタンをクリックします。
6. ［オブジェクトの種類を選択］ダイアログボックスから［テーブル］を選択して［OK］ボタンをクリックします。
7. ［オブジェクトの選択］ダイアログボックスに戻ったら、［参照］ボタンをクリックします。
8. ［オブジェクトの参照］ダイアログボックスから employee テーブルを選択して［OK］ボタンをクリックします。
9. ［オブジェクトの選択］ダイアログボックスに戻ったら、［OK］ボタンをクリックします。
10. ［データベースユーザー］ダイアログボックスに戻ったら、［dbo.employee の明示的な権限］セクションで［Update］権限の［許可］にチェックを入れ、［列権限］ボタンをクリックします。
11. ［列権限］ダイアログボックスの［列権限］セクションで［emp_fname］列と［emp_lname］列の［許可］にチェックを入れ、［OK］ボタンをクリックします。
12. ［データベースユーザー］ダイアログボックスに戻ったら、［OK］ボタンをクリックします。

解答 19.6

次の手順で managers ロールに権限を設定します。

1. コンソールツリーから sample データベース→［セキュリティ］フォルダ→［ロール］フォルダの順に展開し、［データベースロール］フォルダを選択します。
2. 詳細ペインで managers を右クリックし、［プロパティ］を選択します。
3. ［データベースロールのプロパティ］ダイアログボックスで、［セキュリティ保護可能なリソース］ページを選択し、［セキュリティ保護可能なリソース］セクションの［追加］ボタンをクリックします。
4. ［オブジェクトの追加］ダイアログボックスが表示されたら、［特定のオブジェクト］ラジオボタンを選択し、［OK］ボタンをクリックします。
5. ［オブジェクトの選択］ダイアログボックスが表示されたら、［オブジェクトの種類］ボタンをクリックします。
6. ［オブジェクトの種類を選択］ダイアログボックスから［テーブル］を選択して［OK］ボタンをクリックします。
7. ［オブジェクトの選択］ダイアログボックスに戻ったら、［参照］ボタンをクリックします。
8. ［オブジェクトの参照］ダイアログボックスから project テーブルを選択して［OK］ボタンをクリックします。
9. ［オブジェクトの選択］ダイアログボックスに戻ったら、［OK］ボタンをクリックします。
10. ［データベースロールのプロパティ］ダイアログボックスに戻ったら、［dbo.project の明示的な権限］セクションで［Insert］権限の［許可］にチェックを入れ、［OK］ボタンをクリックします。

第 20 章

解答 20.1

差分バックアップの長所は、復元プロセスがシンプルになることです。なぜなら、データベースを完全に復旧する際、データベース全体のバックアップの他に最新の差分バックアップだけを使用すればよいからです。同じシナリオでトランザクションログを使用する場合、データベース全体のバックアップの他に既存のすべてのトランザクションログを使用しないと、データベースが整合性のある状態に戻りません。
差分バックアップの短所は、データを特定の時点まで復旧できないことです。なぜなら、差分バックアップにはデータベースに対して行われた変更プロセスが含まれないからです。

解答 20.2

実稼働データベースをバックアップすべきタイミングは、データベースのサイズや変更操作の回数など、いくつかの要因によって決まります。

解答 20.3

master データベースに対して、差分バックアップ／トランザクションログバックアップ／ファイルバックアップを実行することはできません。したがって、master データベースの復元は、必ず完全バックアップを使って行います。

解答 20.4

よく使われる方法は、データファイルを RAID 0 ドライブに置き、トランザクションログとバックアップをミラードライブ（RAID 1）に置くというものです。データをすばやく回復できるようにしたい場合は、データファイルに RAID 5 を使用し、トランザクションログファイルに RAID 1 を使用します。

解答 20.5

自動復旧はシステムによって自動的に行われますが、手動復旧はシステム管理者が行わなければなりません。

解答 20.6

RESTORE VERIFYONLY ステートメントを使用すると、実際に復元をせずにバックアップの内容を検証できます。

解答 20.7

復旧モデルには、完全復旧モデル、一括ログ復旧モデル、単純復旧モデルの 3 つがあります。
完全復旧モデルでは、データファイルの消失や損傷によってそれまでの作業内容が失われる危険性を最小にすることができます。ただし、すべての変更操作が逐一記録されるため、トランザクションログが非常に大きくなる可能性があります。

一括ログ復旧モデルでは、大規模操作に関して最小限のログ領域しか使わないため、トランザクションログを比較的小さく保ちながら、作業内容が失われる危険性を最小に抑えることができます。ただし、完全復旧モデルと異なり、任意の時点までの復旧機能に制限があります。
単純復旧モデルのバックアップ戦略は最も単純ですが、最新のデータベースバックアップまたは差分バックアップより後に行われた変更をすべて手動でやり直さなければならなくなります。

第 21 章

解答 21.1

データ転送、データベース／トランザクションログのバックアップ、インデックスの保守などの管理タスクを自動化できます。

解答 21.2

トランザクションログをバックアップするためのジョブを 1 つ作成し、2 つのスケジュールを指定します。

解答 21.3

SQL Server のエラーメッセージは、次の要素から構成されます。

- 一意なエラーメッセージ番号
- エラーの重大度レベルを表す 0 から 25 までの番号
- エラーが起きた行を表す行番号
- エラーテキスト

解答 21.4

sys.messages カタログビューの最も重要な列は、message_id、severity、text です。message_id 列にはそのエラーメッセージのエラー番号、severity 列にはエラーの重大度レベル、text 列にはそのエラーメッセージのテキストがそれぞれ格納されます。

第 24 章

解答 24.1

パブリッシュされたテーブルの行を一意に識別するのに主キーが必要です。トランザクションレプリケーションを使用するすべてのテーブルには、明示的に主キーが含まれていなければなりません。

解答 24.2

レプリケート対象となるテーブルを分割したり、データにフィルタをかけたりします。

解答 24.3

サブスクライバが更新できるデータの範囲を一定のサブセットに限定すれば、更新の衝突を最小限に抑えることができます。

解答 24.4

ログリーダーエージェントは、トランザクションレプリケーションで使われるレプリケーションエージェントです。レプリケーション対象としてマークされたトランザクションを探し出して、パブリッシャのトランザクションログから distribution データベースにコピーします。
マージエージェントは、マージレプリケーションで使われるレプリケーションエージェントです。すべてのサイト間での同期化処理を行います。
スナップショットエージェントは、パブリッシュされたテーブルのスキーマとデータを収めたスナップショットファイルを生成します。このスナップショットファイルは、各種のレプリケーションで使用されます。

第 25 章

解答 25.1

OLTP システムは、ユーザー数が多く、短いトランザクションが多数実行されるシステムです。読み取り操作や書き込み操作が絶え間なく行われ、データベースのデータ量は中程度です。データウェアハウスシステムは、ユーザー数が少なく、データベースに大量のデータが格納され、ロードプロセスの後は読み取り操作しか行われません。

解答 25.2

ER モデルは高度に正規化されていますが、ディメンションモデルは冗長データを使用するので非正規化されているのが普通です。また、ER モデルでは、データベースが大きくなるとデータベース設計が非常に複雑になります。

解答 25.3

ETL は、抽出（Extracting）、変換（Transforming）、ロード（Loading）の 3 つの処理から成るプロセスです。抽出は、各種のオペレーショナルシステムからソースデータを一時的な作業領域にロードする処理です。変換は、さまざまなソースから抽出されたデータを修正し整えて、情報を使いやすくする処理です。ロードは、クリーニング済みのデータをデータウェアハウスにロードする処理です。

解答 25.4

ディメンションモデルでは、1 つのファクトテーブルに対して複数のディメンションテーブルが存在します。ファクトテーブルには、データウェアハウス内のデータのうち、かなりの割合（70% ほど）が格納されます。また、ファクトテーブルの列は、数値とその他の付加的要素から成ります。

解答 25.5

MOLAP ／ ROLAP ／ HOLAP は、キューブおよびディメンションの格納方法を表します。
MOLAP 構造では、データ分析のクエリパフォーマンスが最も高くなります。なぜなら、ソースデータのコピーと集計データがマルチディメンション構造に格納されるので、高速なクエリプロセッサでデータをすばやく取得できるからです。ただし、データがキューブ内で重複するため、記憶領域を最も多く消費します。
ROLAP 構造では、標準的な Transact-SQL ステートメントを使ってリレーショナルテーブルに照会を行えます。また、データの重複を避けることができ、余分な記憶領域を必要としません。しかし、クエリのパフォーマンスは MOLAP や HOLAP よりも劣ります。
HOLAP 構造は、MOLAP と ROLAP の中間の性質を持ったストレージタイプです。Analysis Services 内にマルチディメンション構造で集計データを保持しますが、ソースデータのコピーは格納しません。そのため、必要となる記憶領域が MOLAP より少なくなると共に、集計レベルのデータにしかアクセスしないクエリでは MOLAP と同様のパフォーマンスを発揮します。ただし、ドリルダウンなど集計データ以外のデータにアクセスする場合のパフォーマンスは低下します。

解答 25.6

データウェアハウスには大量のデータが格納されるので、集計をそのつど行っていたのでは、クエリの実行時間が非常に長くなってしまうからです。

第 28 章

解答 28.1

問題のレポートを作成するためのクエリは、次のとおりです。

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

SQL Server Business Intelligence Development Studio を使ってレポートを作成する手順、および作成したレポートを表示する手順については、本文を参照してください。

解答 28.2

問題のレポートを作成するためのクエリは、次のとおりです。

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

SQL Server Business Intelligence Development Studio を使ってレポートを作成する手順、および作成したレポートを表示する手順については、本文を参照してください。