

Troy Technologies USA



MCSE

STUDY GUIDE

SQL 7.0 Design & Implementation
Exam 70-29
Edition 1

Congratulations!!

You have purchased a ***Troy Technologies USA*** Study Guide.

This study guide is a selection of questions and answers similar to the ones you will find on the official SQL 7.0 Design & Implementation MCSE exam. Study and memorize the following concepts, questions and answers for approximately 10 to 15 hours and you will be prepared to take the exams. We guarantee it!

Remember, average study time is 10 to 15 hours and then you are ready!!!

GOOD LUCK!

Guarantee

If you use this study guide correctly and still fail the exam, send your official score notice and mailing address to:

Troy Technologies USA
8200 Pat Booker Rd. #368
San Antonio, TX 78233

We will gladly refund the cost of this study guide. However, you will not need this guarantee if you follow the above instructions.

Ó Copyright 1999 Troy Technologies USA. All Rights Reserved.

SQL Design and Implementation Concepts

The Relational Database Model

The whole basis for the relational model follows this train of thought: data is stored in tables, which are composed of rows and columns. Tables of independent data can be linked, or related, to one another if all have columns of data that represent the same data value, called keys.

Logical Data Model

Designing an optimal relational database is typically accomplished by using entity-relationship modeling and normalization. These two processes or methodologies aid in defining the logical data model.

Entity-relationship modeling is the identification of data entities or objects and the relationships between these entities. The entity-relationship modeling process usually progresses in the following manner:

1. Entities or objects are identified.
2. Identifiers or primary keys that uniquely identify each entity are identified.
3. Data elements or attributes associated with each entity are identified.
4. Relationships between entities are identified, thereby resulting in the identification of primary key/foreign key relationships.

Having completed this process, the resulting entity-relationship model now represents a logical view of the data entities and associated relationships upon which the physical database design will be based.

Normalization

The process of normalization is characterized by levels of adherence called normal form. If a table design conforms only to the lowest level of normalization, it is said to be in First Normal Form, which is abbreviated as 1NF. If a table design conforms to the next higher level, it is in Second Normal Form (2NF), and so on. It is uncommon and probably never necessary to take a table design beyond Third Normal Form (3NF). In fact, in some cases, it is actually advantageous in terms of performance to denormalize a table from a higher normal form back to a lower normal form, like from 3NF to 2NF. It is also important to remember that normalizing tables is not full database design; rather, it is an important step in the analysis process.

Assume that we are keeping a database that contains information about all of our circus animals.

Unnormalized.

The data, when completely unnormalized, looks like this:

Circus_Info Table, Unnormalized

1. animal_nbr (the primary key)
2. animal_name
3. tent_nbr
4. tent_name
5. tent_location
6. trick_nbr1
7. trick_name1
8. trick_learned_at1
9. trick_skill_level1
10. trick_nbr2...(and on through 16)
11. trick_name2...(and on through 16)

12. trick_learned_at2...(and on through 16)
13. trick_skill_level2...(and on through 16)

In this example, the animal_nbr column is our primary key, or the value that uniquely identifies each row. This is an unnormalized structure. But why go to the trouble of normalizing it? Well, otherwise, there would be a number of logical anomalies to deal with. In fact, there would be logic faults whenever you attempted insert, delete, or update operations on this table. The reason there are so many problems with this table design is that much of the data is redundant and does not wholly depend on the key of the table (animal_nbr). These are the main problems:

Insert problem

Suppose that a chimpanzee has been acquired and has to be added to the table. It would be impossible to add the new chimp without also adding a trick. Conversely, a new trick couldn't be added to the database without also assigning the trick to a circus animal, which might not reflect the actual business rule.

Delete problem

What happens if a trick is deleted from the table? Unfortunately, important data about a circus animal would also be deleted. The situation also falters when an animal is deleted. This erases information about a trick that may still be used by the circus.

Update problem

Updates in the unnormalized table could cause actions on multiple records. Great care must be taken to properly identify the appropriate record when making a change.

First Normal Form (1NF)

To put this in *First Normal Form (1NF)*, we must eliminate repeating groups. That means that any subgroups of data that appear within the record should be split into separate tables. In this case, we have at least two major repeating groups to split. Thus, our unnormalized structure becomes two tables in 1NF:

Circus_Animals Table in 1NF

1. animal_nbr (the primary key)
2. animal_name
3. tent_nbr
4. tent_name
5. tent_location

Tricks Table in 1NF

1. animal_nbr (the concatenated key)
2. trick_nbr (the concatenated key)
3. trick_name
4. trick_learned_at
5. trick_skill_level

Now that our tables are in 1NF, we have the obvious advantage of reduced space consumption. Plus, an animal that only knows a few tricks doesn't use up the space allotted for 16 tricks in the unnormalized table. On the flip side, the animal can also know more than 16 tricks. Also note that in the Tricks table, the key had to be expanded to include both animal_nbr and trick_nbr. Concatenated keys are often the byproduct of the normalization process.

Second Normal Form (2NF)

Let's take our 1NF circus tables to Second Normal Form (2NF) by eliminating partial dependencies. A partial dependency is a fancy word for data that doesn't depend on the primary key of the table to uniquely identify

it. For example, the trick name appears in every animal record, but the `trick_nbr` should be sufficient since we've already got the name in the Tricks table.

So this 1NF table:

Tricks Table

1. `animal_nbr`
2. `trick_nbr`
3. `trick_name`
4. `trick_learned_at`
5. `trick_skill_level`

becomes these 2NF tables:

Tricks Table

1. `trick_nbr`
2. `trick_nam`

Animal_Tricks Table

1. `animal_nbr`
2. `trick_nbr`
3. `trick_learned_at`
4. `trick_skill_level`

Animal Table in 2NF

1. `animal_nbr`
2. `animal_name`
3. `tent_nbr`
4. `tent_name`
5. `tent_location`

Unfortunately 2NF, like 1NF, has its own logical faults when attempting data manipulation:

Insert problem

Assume that we want to create a new record to represent a tiger assigned to the "Big Cats" tent. It's impossible to create an additional tiger record for the Animal table without first creating a new tent. This is due to the fact the `tent_nbr` column has a transitive dependency to the `animal_nbr` column, which more uniquely identifies its information than does the `tent_nbr` column.

Delete problem

Deleting a particular animal might result in the loss of an entire tent, even though you still want to retain information about that tent. For example, deleting the sole ostrich in the circus completely removes all traces of the "Big Birds" tent, since you only have one of these large birds.

Update problem

Tent information is redundant in the Animal table, so any change to tent information require a search of the entire table to locate the records needing alteration. Plus, the number of the records requiring the update may vary over time.

An additional step of normalization is needed to eliminate these logic faults. This added step converts a 2NF table into a 3NF table.

Third Normal Form (3NF)

To take this motley crew of tables to Third Normal Form (3NF), we must now take 2NF tables and eliminate all transitive (i.e., hidden) dependencies. In other words, every column that isn't a part of the key must

depend on the key for its informational value. Non-key columns should have no informational value if they are separated from the key of a 3NF table. Remember that the term "eliminate" doesn't mean "delete." Instead, it means to split into separate tables.

Our tables in 3NF would now look like this:

Tricks Table (unchanged, it was already 3NF!)

1. trick_nbr
2. trick_name

Animal_Tricks Table (unchanged, it was already 3NF!)

1. animal_nbr
2. trick_nbr
3. trick_learned_at
4. trick_skill_level

The Animal table becomes two new 3NF tables:

Animal_Lodging Table in 3NF

1. animal_nbr
2. animal_name
3. tent_nbr

Tents Table in 3NF

1. tent_nbr
2. tent_name
3. tent_location

2NF and 3NF are all about the relationship between non-key and key fields. A 3NF table should provide a single-value fact about the key and should use the whole key (if concatenated), and nothing but the key, to derive its uniqueness.

By the time you get to 2NF, the columns all depend on the primary key for their identity. However, the columns may have such a distinction from other columns in the table that they should be split out into a separate table. In the Animal_Lodging table, the tent_nbr still appears as a non-key column in the table, even though it is the primary key of the Tents table. Thus, tent_nbr is a foreign key within the animal_Lodging table. Using tent_nbr as a foreign key, we are able to join the data together from the two separate tables to form a single data set.

Remember, normalizing data means eliminating redundant information from a table and organizing the data so that future changes to the table are easier. A table is in First Normal Form (1NF) when each field contains the smallest meaningful data and the table contains no repeating fields. Second Normal Form (2NF) refers only to tables with a multiple-field primary key. Each non-key field should relate to all of the fields making up the primary key. Third Normal Form (3NF) refers only to tables with a single-key field and requires that each non-key field be a direct description of the primary key field.

Denormalization

Designing table structures to 3NF reduces data redundancy and provides clean, logical table structures.

The main benefit of denormalization is improved performance, although simplified data retrieval and manipulation are sometimes lesser benefits. Both of these benefits are rendered through the reduction in the number of joins needed for the proper functionality of your application. Joins merge the data of two related tables into a single result set, presenting a denormalized view of the data. Joins, however, are rather time consuming and utilize a lot of CPU cycles. So, to reduce the expense of joins, many developers preempt what would otherwise be a very common join with a denormalized table. The table is now exposed to the vulnerabilities inherent in its lower normal form, but considerable time is saved because the application no

longer has to join multiple tables together in a temporary workspace. Denormalization can also save processing time that might have been spent on the creation of summary reports.

The first thing you need to remember about denormalization is that you must thoroughly understand the needs and behavior of the data. There are some other tips to keep in mind when considering denormalizing tables:

- Denormalization means that more storage space will be needed for the redundant data.
- Redundant data usually speeds up queries but can slow down updates since multiple instances of the data item must be altered.
- Denormalized tables can introduce logic faults that must be compensated for in the application code.
- Maintaining integrity can be more difficult with redundant data.
- Denormalization yields the best improvements on tables that are frequently queried, such as validation tables or reporting tables. Tables in which records are frequently inserted, updated, and deleted are less viable candidates for denormalization.

Denormalization is usually accomplished by adding columns and foreign keys from other tables into the denormalized table or by adding aggregate columns to the denormalized table. If every query issued against a given table commonly has to join to columns from one or more other tables, you're looking at a candidate for denormalization.

Relational Database Design

When designing a database, you have to make decisions regarding how best to take some system in the real world and model it in a database. This consists of deciding which tables to create, what columns they will contain, as well as the relationships between the tables.

The benefits of a database that has been designed according to the relational model are numerous. Some of them are:

- Data entry, updates and deletions will be efficient.
- Data retrieval, summarization and reporting will also be efficient.
- Since the database follows a well-formulated model, it behaves predictably.
- Since much of the information is stored in the database rather than in the application, the database is somewhat self-documenting.
- Changes to the database schema are easy to make.

Tables, Uniqueness and Keys

Tables in the relational model are used to represent “things” in the real world. Each table should represent only one thing. These things (or entities) can be real-world objects or events. For example, a real-world object might be a customer, an inventory item, or an invoice. Examples of events include patient visits, orders, and telephone calls. Tables are made up of rows and columns.

The relational model dictates that each row in a table be unique. If you allow duplicate rows in a table, then there's no way to uniquely address a given row via programming. This creates all sorts of ambiguities and problems that are best avoided. You guarantee uniqueness for a table by designating a primary key—a column that contains unique values for a table. Each table can have only one primary key, even though several columns or combination of columns may contain unique values. All columns (or combination of columns) in a table with unique values are referred to as candidate keys, from which the primary key must be drawn. All other candidate key columns are referred to as alternate keys. Keys can be simple or composite. A simple key is a key made up of one column, whereas a composite key is made up of two or more columns.

The decision as to which candidate key is the primary one rests in your hands—there’s no absolute rule as to which candidate key is best. Say that a company has a table of customers called `tblCustomer`, which looks like the table shown in Figure 1.

Table: tblCustomer							
CustomerId	LastName	FirstName	Address	City	State	ZipCode	Phone#
1	Jones	Paul	1313 Mockingbird Lane	Seattle	WA	98117	2068886902
2	Nelson	Greg	45-39 173rd St	Redmond	WA	98119	2069809099
3	Madison	Ken	2345 16th NE	Kent	WA	98109	2067837890
4	Jones	Geoff	1313 Mockingbird Lane	Seattle	WA	98117	2068886902
*							

Record: 1 of 4

Figure 1. The best choice for primary key for `tblCustomer` would be `CustomerId`.

Candidate keys for `tblCustomer` might include `CustomerId`, (`LastName` + `FirstName`), `Phone#`, (`Address`, `City`, `State`), and (`Address` + `ZipCode`). You would rule out the last three candidates because addresses and phone numbers can change fairly frequently. The choice among `CustomerId` and the name composite key is less obvious and would involve tradeoffs. How likely would a customer’s name change (e.g., marriages cause names to change)? Will misspelling of names be common? How likely will two customers have the same first and last names? How familiar will `CustomerId` be to users? Most developers favor numeric primary keys because names do sometimes change and because searches and sorts of numeric columns are more efficient than of text columns.

Foreign Keys and Domains

Although primary keys are a function of individual tables, if you created databases that consisted of only independent and unrelated tables, you’d have little need for them. Primary keys become essential, however, when you start to create relationships that join together multiple tables in a database. A foreign key is a column in a table used to reference a primary key in another table.

Continuing the example presented in the last section, let’s say that you choose `CustomerId` as the primary key for `tblCustomer`. Now define a second table, `tblOrder`, as shown in Figure 2.

Table: tblOrder			
OrderId	CustomerId	OrderDate	
1	1	5/1/94	
2	3	5/9/94	
3	1	7/4/94	
4	2	8/1/94	
5	1	8/2/94	
6	2	8/2/94	
*			

Record: 4 of 6

Figure 2. `CustomerId` is a foreign key in `tblOrder` which can be used to reference a customer stored in the `tblCustomer` table.

`CustomerId` is considered a foreign key in `tblOrder` since it can be used to refer to given customer (i.e., a row in the `tblCustomer` table).

It is important that both foreign keys and the primary keys that are used to reference share a common meaning and draw their values from the same domain. Domains are simply pools of values from which columns are drawn. For example, `CustomerId` is of the domain of valid customer ID #’s, which in this case might be Long Integers ranging between 1 and 50,000. Similarly, a column named `Sex` might be based on a one-letter domain equaling ‘M’ or ‘F’. Domains can be thought of as user-defined column types whose definition implies certain rules that the columns must follow and certain operations that you can perform on those columns.

Relationships

You define foreign keys in a database to model relationships in the real world. Relationships between real-world entities can be quite complex, involving numerous entities each having multiple relationships with each other. For example, a family has multiple relationships between multiple people—all at the same time. In a relational database, however, you consider only relationships between pairs of tables. These tables can be related in one of three different ways: one-to-one, one-to-many or many-to-many.

One-to-One Relationships

Two tables are related in a one-to-one (1–1) relationship if, for every row in the first table, there is at most one row in the second table. True one-to-one relationships seldom occur in the real world. This type of relationship is often created to get around some limitation of the database management software rather than to model a real-world situation. One-to-one relationships may be necessary in a database when you have to split a table into two or more tables because of security or performance concerns. For example, you might keep most patient information in `tblPatient`, but put especially sensitive information (e.g., patient name, social security number and address) in `tblConfidential` (see Figure 3). Access to the information in `tblConfidential` could be more restricted than for `tblPatient`. As a second example, perhaps you need to transfer only a portion of a large table to some other application on a regular basis. You can split the table into the transferred and the non-transferred pieces, and join them in a one-to-one relationship.

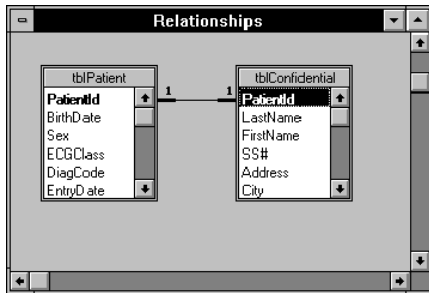


Figure 3. The tables `tblPatient` and `tblConfidential` are related in a one-to-one relationship. The primary key of both tables is `PatientId`.

Tables that are related in a one-to-one relationship should always have the same primary key, which will serve as the join column.

One-to-Many Relationships

Two tables are related in a one-to-many (1–M) relationship if for every row in the first table, there can be zero, one, or many rows in the second table, but for every row in the second table there is exactly one row in the first table. For example, each order for a pizza delivery business can have multiple items. Therefore, `tblOrder` is related to `tblOrderDetails` in a one-to-many relationship (see Figure 4). The one-to-many relationship is also referred to as a parent-child or master-detail relationship. One-to-many relationships are the most commonly modeled relationship.

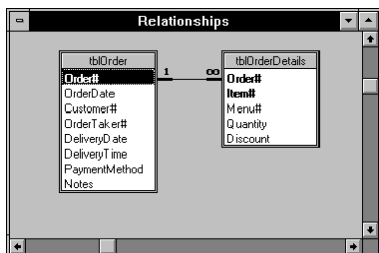


Figure 4. There can be many detail lines for each order in the pizza delivery business, so `tblOrder` and `tblOrderDetails` are related in a one-to-many relationship.

One-to-many relationships are also used to link base tables to information stored in lookup tables. For example, tblPatient might have a short one-letter DischargeDiagnosis code, which can be linked to a lookup table, tlkpDiagCode, to get more complete Diagnosis descriptions (stored in DiagnosisName). In this case, tlkpDiagCode is related to tblPatient in a one-to-many relationship (i.e., one row in the lookup table can be used in zero or more rows in the patient table).

Many-to-Many Relationships

Two tables are related in a many-to-many (M–M) relationship when for every row in the first table, there can be many rows in the second table, and for every row in the second table, there can be many rows in the first table. Many-to-many relationships can't be directly modeled in relational database programs. These types of relationships must be broken into multiple one-to-many relationships. For example, a patient may be covered by multiple insurance plans and a given insurance company covers multiple patients. Thus, the tblPatient table in a medical database would be related to the tblInsurer table in a many-to-many relationship. In order to model the relationship between these two tables, you would create a third, linking table, perhaps called tblPtInsurancePgm that would contain a row for each insurance program under which a patient was covered (see Figure 5). Then, the many-to-many relationship between tblPatient and tblInsurer could be broken into two one-to-many relationships (tblPatient would be related to tblPtInsurancePgm and tblInsurer would be related to tblPtInsurancePgm in one-to-many relationships).

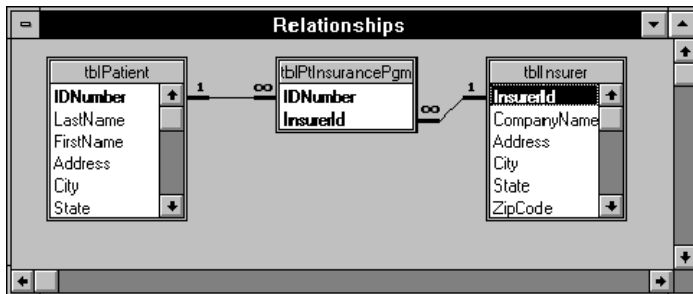


Figure 5. A linking table, **tblPtInsurancePgm**, is used to model the many-to-many relationship between **tblPatient** and **tblInsurer**.

Integrity Rules

The relational model defines several integrity rules that, while not part of the definition of the Normal Forms are nonetheless a necessary part of any relational database. There are two types of integrity rules: general and database-specific.

General Integrity Rules

The relational model specifies two general integrity rules. They are referred to as general rules, because they apply to all databases. They are: entity integrity and referential integrity.

The entity integrity rule is very simple. It says that primary keys cannot contain null (missing) data. The reason for this rule should be obvious. You can't uniquely identify or reference a row in a table, if the primary key of that table can be null. It's important to note that this rule applies to both simple and composite keys. For composite keys, none of the individual columns can be null.

The referential integrity rule says that the database must not contain any unmatched foreign key values. This implies that:

- A row may not be added to a table with a foreign key unless the referenced value exists in the referenced table.
- If the value in a table that's referenced by a foreign key is changed (or the entire row is deleted), the rows in the table with the foreign key must not be "orphaned."

In general, there are three options available when a referenced primary key value changes or a row is deleted. The options are:

- **Disallow.** The change is completely disallowed.
- **Cascade.** For updates, the change is cascaded to all dependent tables. For deletions, the rows in all dependent tables are deleted.
- **Nullify.** For deletions, the dependent foreign key values are set to Null.

Database-Specific Integrity Rules

All integrity constraints that do not fall under entity integrity or referential integrity are termed database-specific rules or business rules. These type of rules are specific to each database and come from the rules of the business being modeled by the database. It is important to note that the enforcement of business rules is *as* important as the enforcement of the general integrity rules discussed in the previous section. Without the specification and enforcement of business rules, bad data will get in the database.

Objects

The data in a Microsoft SQL Server database is organized into several different objects. These objects are what a user can see when they connect to the database.

In SQL Server, these components are defined as objects:

<i>Constraints</i>	<i>Tables</i>
<i>Defaults</i>	<i>Triggers</i>
<i>Indexes</i>	<i>User-defined data types</i>
<i>Keys</i>	<i>Views</i>
<i>Stored Procedures</i>	

All the data in Microsoft SQL Server databases is contained in objects called tables. Each table represents some type of object meaningful to the users.

SQL Server tables have two main components:

- **Columns**
Each column represents some attribute of the object modeled by the table, such as a parts table having columns for ID, color, and weight.
- **Rows**
Each row represents an individual occurrence of the object modeled by the table. For example, the

The parts table would have one row for each part carried by the company.

parts table		
ID	color	weight
AB123	Blue	10.5
CD456	Red	8.0
EF789	Green	9.25
GH012	Yellow	8.0
IJ341	Blue	1.0

Because each column represents one attribute of an object, the data in each occurrence of the column is similar. One of the properties of a column is called its data type, which defines the type of data the column can hold. SQL Server has several base data types.

<i>Binary</i>	<i>bit</i>	<i>char</i>	<i>datetime</i>	<i>decimal</i>
<i>float</i>	<i>image</i>	<i>int</i>	<i>money</i>	<i>nchar</i>
<i>ntext</i>	<i>nvarchar</i>	<i>numerical</i>	<i>smalldatetime</i>	<i>smallint</i>
<i>smallmoney</i>	<i>sysnametext</i>	<i>timestamp</i>	<i>tinyint</i>	<i>varbinary</i>
<i>varchar</i>	<i>uniqueidentifier</i>			

User-Defined Data Type

A user-defined data type makes a table structure more meaningful to a reader and helps ensure that columns holding similar classes of data have the same base data type.

A domain is the set of all allowable values in a column. It includes not only the concept of enforcing data types, but also the values allowed in the column. For example, a part color domain would include both the data type, such as **char**(6), and the character strings allowed in the column, such as Red, Blue, Green, Yellow, Brown, Black, White, Teal, Grey, and Silver. Domain values can be enforced through mechanisms such as CHECK constraints and triggers.

Columns can either accept or reject NULL values. NULL is a special value in databases which represents the concept of an unknown value. NULL is not the same as a blank character or 0. Blank is actually a valid character, and zero is a valid number, while NULL just represents the idea that we do not know what this value is. NULL is also different from a zero length string. If a column definition contains the NOT NULL clause, you cannot insert rows having the value NULL for that row. If the column definition just has the NULL keyword, it accepts NULL values.

Allowing NULL values in a column can increase the complexity of any logical comparisons using the column. The SQL-92 standard states that any comparison against a NULL value does not evaluate to TRUE or FALSE, it evaluates to UNKNOWN.

Views

A view can be thought of as either a virtual table or a stored query. The data accessible through a view is not stored in the database as a distinct object. What is stored in the database is a SELECT statement. The result set of the SELECT statement forms the virtual table returned by the view. A user can use this virtual table by referencing the view name in Transact-SQL statements the same way a table is referenced. A view is used to do any or all of these functions:

- Restrict a user to specific rows in a table.
- Restrict a user to specific columns.
- Join columns from multiple tables so that they look like a single table.
- Aggregate information instead of supplying details.

Views are created by defining the SELECT statement that retrieves data to be presented by the view. The data tables referenced by the SELECT statement are known as the base tables for the view. **titleview** in the **pubs** database is an example of a view that selects data from three base tables to present a virtual table of commonly needed data.

```
CREATE VIEW titleview
AS
SELECT title, au_ord, au_lname, price, ytd_sales, pub_id
FROM authors AS a
JOIN titleauthor AS ta ON (a.au_id = ta.au_id)
```

```
JOIN titles AS t ON (t.title_id = ta.title_id)
```

You can then reference **titleview** in statements in the same way you would reference a table.

```
SELECT * FROM titleview
```

A view can reference another view. For example, **titleview** presents information that is useful for managers, but a company typically only discloses year-to-date figures in quarterly or annual financial statements. A view can be built that selects all the **titleview** columns except **au_ord** and **ytd_sales**. This new view can be used by customers to get lists of available books without seeing the financial information:

```
CREATE VIEW Cust_titleview
AS
SELECT title, au_lname, price, pub_id
FROM titleview
```

Views in Microsoft SQL Server are updatable (can be the target of UPDATE, DELETE, or INSERT statements) so long as the modification only affects one of the base tables referenced by the view.

```
-- Increase the prices for publisher '0736' by 10%.
UPDATE titleview
SET price = price * 1.10
WHERE pub_id = '0736'
GO
```

Stored Procedures

A stored procedure is a group of Transact-SQL statements compiled into a single execution plan.

Microsoft SQL Server stored procedures return data in four ways:

- Output parameters, which can return either data (such as an integer or character value) or a cursor variable (cursors are result sets that can be retrieved one row at a time).
- Return codes, which are always an integer value.
- A result set for each SELECT statement contained in the stored procedure or any other stored procedures called by the stored procedure.
- A global cursor that can be referenced outside the stored procedure.

Stored procedures assist in achieving a consistent implementation of logic across applications. The SQL statements and logic needed to perform a commonly performed task can be designed, coded, and tested once in a stored procedure. Each application needing to perform that task can then simply execute the stored procedure. Coding business logic into a single stored procedure also offers a single point of control for ensuring that business rules are correctly enforced.

Stored procedures can also improve performance. Many tasks are implemented as a series of SQL statements. Conditional logic applied to the results of the first SQL statements determines which subsequent SQL statements are executed. If these SQL statements and conditional logic are written into a stored procedure, they become part of a single execution plan on the server. The results do not have to be returned to the client to have the conditional logic applied; all of the work is done on the server. The IF statement in this example shows embedding conditional logic in a procedure to keep from sending a result set to the application:

```
IF (@QuantityOrdered < (SELECT QuantityOnHand
                        FROM Inventory
                        WHERE PartID = @PartOrdered) )
```

```

        BEGIN
        -- SQL statements to update tables and process order.
        END
ELSE
    BEGIN
    -- SELECT statement to retrieve the IDs of alternate items
    -- to suggest as replacements to the customer.
    END

```

If a set of stored procedures supports all of the business functions users need to perform, users never need to access the tables directly; they can just execute the stored procedures that model the business processes with which they are familiar.

An illustration of this use of stored procedures is the SQL Server system stored procedures used to insulate users from the system tables. SQL Server includes a set of system stored procedures whose names usually start with **sp_**. These system stored procedures support all of the administrative tasks required to run a SQL Server system. You can administer a SQL Server system using the Transact-SQL administration-related statements (such as CREATE TABLE) or the system stored procedures, and never need to directly update the system tables.

Integrity

Table columns have properties other than just data type and size. The other properties form an important part of the support for ensuring the integrity of data in a database.

- Data integrity refers to each occurrence of a column having a correct data value.
The data values must be of the right data type and in the correct domain.
- Referential integrity indicates that the relationships between tables have been properly maintained.
Data in one table should only point to existing rows in another table; it should not point to rows that do not exist.

Objects used to maintain both types of integrity include:

- Constraints
- Rules
- Defaults
- Triggers

Constraints

Constraints offer a way to have Microsoft SQL Server enforce the integrity of a database automatically. Constraints define rules regarding the values allowed in columns and are the standard mechanism for enforcing integrity, preferred over triggers, rules, and defaults. They are also used by the query optimizer to improve performance in selectivity estimation, cost calculations, and query rewriting.

There are five classes of constraints.

- NOT NULL specifies that the column does not accept NULL values.
- CHECK constraints enforce domain integrity by limiting the values that can be placed in a column.

A CHECK constraint specifies a Boolean (evaluates to TRUE or FALSE) search condition that is applied to all values entered for the column; all values that do not evaluate to TRUE are rejected. You can specify multiple CHECK constraints for each column. This sample shows the creation of a named constraint, **chk_id**, that further enforces the domain of the primary key by ensuring that only numbers within a specified range are entered for the key.

```
CREATE TABLE cust_sample
(
    cust_id                int                PRIMARY KEY,
    cust_name              char(50),
    cust_address           char(50),
    cust_credit_limit      money,
    CONSTRAINT chk_id CHECK (cust_id BETWEEN 0 and 10000 )
)
```

- UNIQUE constraints enforce the uniqueness of the values in a set of columns.

No two rows in the table are allowed to have the same nonNULL values for the columns in a UNIQUE constraint. Primary keys also enforce uniqueness, but primary keys do not allow NULL values. A UNIQUE constraint is preferred over a unique index.

- PRIMARY KEY constraints identify the column or set of columns whose values uniquely identify a row in a table.

No two rows in a table can have the same primary key value. You cannot enter a NULL value for any column in a primary key. NULL is a special value in databases that represents an unknown value, which is distinct from a blank or 0 value. Using a small, integer column as a primary key is recommended. Each table should have a primary key.

```
CREATE TABLE part_sample
(
    part_nmbr              int                PRIMARY KEY,
    part_name              char(30),
    part_weight            decimal(6,2),
    part_color             char(15) )
```

- FOREIGN KEY constraints identify the relationships between tables.

A foreign key in one table points to a candidate key in another table. You cannot insert a row with a foreign key value (except NULL) if there is no candidate key with that value. You cannot delete a row from the referenced table if there are any foreign key values referencing that candidate key. In the following sample, the **order_part** table establishes a foreign key referencing the **part_sample** table defined earlier. Normally, **order_part** would also have a foreign key against an order table, but this is a simple example.

```
CREATE TABLE order_part
(
    order_nmbr             int,
    part_nmbr              int
    FOREIGN KEY REFERENCES part_sample(part_nmbr),
    qty_ordered            int)
GO
```

Constraints can be column constraints or table constraints:

- A column constraint is specified as part of a column definition and applies only to that column (the constraints in the earlier samples are column constraints).
- A table constraint is declared independently from a column definition and can apply to more than one column in a table.

Table constraints must be used when more than one column must be included in a constraint.

For example, if a table has two or more columns in the primary key, you must use a table constraint to include both columns in the primary key. Consider a table that records events happening in a machine in a factory. Assume that events of several types can happen at the same time, but that no two events happening at the same time can be of the same type. This can be enforced in the table by including both the **type** and **time** columns in a two-column primary key.

```
CREATE TABLE factory_process
```

```

        (event_type      int,
         event_time      datetime,
         event_site      char(50),
         event_desc      char(1024),
 CONSTRAINT event_key PRIMARY KEY (event_type, event_time) )

```

Rules

Rules are a backward compatibility feature that perform some of the same functions as CHECK constraints. CHECK constraints are the preferred, standard way to restrict the values in a column. CHECK constraints are also more concise than rules; there can only be one rule applied to a column, but multiple CHECK constraints can be applied. CHECK constraints are specified as part of the CREATE TABLE statement, while rules are created as separate objects and then bound to the column.

This example creates a rule that performs the same function as the CHECK constraint example in the preceding topic. The CHECK constraint is the preferred method to use in Microsoft SQL Server.

```

CREATE RULE id_chk AS @id BETWEEN 0 and 10000
GO
CREATE TABLE cust_sample
(
    cust_id                int                PRIMARY KEY,
    cust_name              char(50),
    cust_address           char(50),
    cust_credit_limit      money,
)
GO
sp_bindrule id_chk, 'cust_sample.cust_id'
GO

```

Defaults

Defaults specify what values are used in a column if you do not specify a value for the column when inserting a row. Defaults can be anything that evaluates to a constant:

- Constant
- Built-in function
- Mathematical expression

There are two ways to apply defaults:

- Create a default definition using the DEFAULT keyword in CREATE TABLE to assign a constant expression as a default on a column.
This is the preferred, standard method. It is also the more concise way to specify a default.
- Create a default object using the CREATE DEFAULT statement and bind it to columns using the **sp_bindefault** system stored procedure.
This is a backward compatibility feature.

This example creates a table using one of each type of default. It creates a default object to assign a default to one column, and binds the default object to the column. It then does a test insert without specifying values for the columns with defaults and retrieves the test row to verify the defaults were applied.

```

USE pubs
GO
CREATE TABLE test_defaults
    (keycol      smallint,
     process_id  smallint DEFAULT @@SPID,      --Preferred default definition

```



```

        date_ins datetime DEFAULT getdate(),      --Preferred default definition
        mathcol      smallint DEFAULT 10 * 2,    --Preferred default definition
        char1        char(3),
        char2        char(3) DEFAULT 'xyz' --Preferred default definition
GO
/* Illustration only, use DEFAULT definitions instead.*/
CREATE DEFAULT abc_const AS 'abc'
GO
sp_bindefault abc_const, 'test_defaults.char1'
GO
INSERT INTO test_defaults(keycol) VALUES (1)
GO
SELECT * FROM test_defaults
GO

```

Triggers

Triggers are a special class of stored procedure defined to execute automatically when an UPDATE, INSERT, or DELETE statement is issued against a table. Triggers are a powerful tool that allows each site to enforce their business rules automatically when data is modified. Triggers can extend the integrity checking logic of Microsoft SQL Server constraints, defaults, and rules, although constraints and defaults should be used instead whenever they provide all the needed functionality.

Tables can have multiple triggers. The CREATE TRIGGER statement can be defined with the FOR UPDATE, FOR INSERT, or FOR DELETE clauses to target a trigger to a specific class of data modification actions. When FOR UPDATE is specified, the IF UPDATE (*column_name*) clause can be used to target a trigger to updates affecting a particular column. SQL Server allows you to specify multiple triggers for a specific action (UPDATE, INSERT, or DELETE) on a single table.

Triggers can automate a company's processing. In an inventory system, update triggers can detect when a stock level reaches a reorder point and generate an order to the supplier automatically. In a database recording the processes in a factory, triggers can e-mail or page operators when a process exceeds defined safety limits.

The following trigger generates an e-mail whenever a new title is added in the **pubs** database.

```

CREATE TRIGGER reminder
ON titles
FOR INSERT
AS
    EXEC master..xp_sendmail 'MaryM',
        'New title, mention in the next report to distributors.'

```

Triggers contain Transact-SQL statements, much the same as stored procedures. Triggers, like stored procedures, return the result set generated by any SELECT statements in the trigger. Including SELECT statements in triggers, except ones that only fill parameters, is not recommended because users do not expect to see any result sets on an UPDATE, INSERT, or DELETE statement.

Triggers execute after the statement that triggered them completes. If the statement fails with an error, such as a constraint violation or syntax error, the trigger is not executed.

Indexes

A Microsoft SQL Server index is a structure associated with a table that speeds retrieval of the rows in the table. An index contains keys built from one or more columns in the table. These keys are stored in a structure that allows SQL Server to find the row or rows associated with the key values quickly and efficiently.

If a table is created with no indexes, the data rows are not stored in any particular order. This structure is called a heap.

The two types of SQL Server indexes are:

Clustered

Clustered indexes sort and store the data rows in the table based on their key values. Because the data rows are stored in sorted order on the clustered index key, clustered indexes are efficient for finding rows. There can only be one clustered index per table, because the data rows themselves can only be sorted in one order. The data rows themselves form the lowest level of the clustered index.

The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. If a table has no clustered index, its data rows are stored in a heap.

Nonclustered

Nonclustered indexes have a structure that is completely separate from the data rows. The lowest rows of a nonclustered index contain the nonclustered index key values and each key value entry has pointers to the data rows containing the key value. The data rows are not stored in order based on the nonclustered key.

The pointer from an index row in a nonclustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or are clustered. For a heap, a row locator is a pointer to the row. For a table with a clustered index, the row locator is the clustered index key.

Indexes can be unique, which means no two rows can have the same value for the index key. Otherwise, the index is not unique and multiple rows can share the same key value.

There are two ways to define indexes in SQL Server. The CREATE INDEX statement creates and names an index. The CREATE TABLE statement supports the following constraints that create indexes:

- PRIMARY KEY creates a unique index to enforce the primary key.
- UNIQUE creates a unique index.
- CLUSTERED creates a clustered index.
- NONCLUSTERED creates a nonclustered index.

A fill factor is a property of a SQL Server index that controls how densely the index is packed when created. The default fill factor usually delivers good performance, but in some cases it may be beneficial to change the fill factor. If the table is going to have many updates and inserts, create an index with a low fill factor to leave more room for future keys. If the table is a read-only table that will not change, create the index with a high fill factor to reduce the physical size of the index, which lowers the number of disk reads SQL Server uses to navigate through the index. Fill factors are only applied when the index is created. As keys are inserted and deleted, the index will eventually stabilize at a certain density.

Indexes not only speed up the retrieval of rows for selects, they also usually increase the speed of updates and deletes. This is because SQL Server must first find a row before it can update or delete the row. The increased efficiency of using the index to locate the row usually offsets the extra overhead needed to update the indexes, unless the table has a lot of indexes.

This example shows the Transact-SQL syntax for creating indexes on a table.

```
USE pubs
```

```
GO
```

```
CREATE TABLE emp_sample
    (emp_id          int          PRIMARY KEY CLUSTERED,
     emp_name        char(50),
     emp_address     char(50),
```

```

emp_title          char(25)          UNIQUE NONCLUSTERED )
GO
CREATE NONCLUSTERED INDEX sample_nonclust ON emp_sample(emp_name)
GO

```

Deciding which particular set of indexes will optimize performance depends on the mix of queries in the system. Consider the clustered index on **emp_sample.emp_id**. This works well if most queries referencing **emp_sample** have equality or range comparisons on **emp_id** in their WHERE clauses. If the WHERE clauses of most queries reference **emp_name** instead of **emp_id**, performance could be improved by instead making the index on **emp_name** the clustered index.

Many applications have a complex mix of queries that is difficult to estimate by interviewing users and programmers. SQL Server version 7.0 provides an Index Tuning Wizard to help design indexes in a database. The easiest way to design indexes for large schemas with complex access patterns is to use the Index Tuning Wizard.

You provide the Index Tuning Wizard with a set of SQL statements. This could be a script of statements you build to reflect a typical mix of statements in the system, but it is usually a SQL Server Profiler trace of the actual SQL statements processed on the system during a period of time that reflects the typical load on the system. The Index Tuning Wizard analyzes the workload and the database, then recommends an index configuration that will improve the performance of the workload. You can choose to either replace the existing index configuration, or to keep the existing index configuration and implement new indexes to improve the performance of a slow-running subset of the queries.

Full-Text Indexing

A Microsoft SQL Server full-text index provides efficient support for sophisticated word searches in character string data. The full-text index stores information about significant words and their location within a given column. This information is used to quickly complete full-text queries that search for rows with particular words or combinations of words.

Full-text indexes are contained in full-text catalogs. Each database can contain one or more full-text catalogs. A catalog cannot belong to multiple databases and each catalog can contain full-text indexes for one or more tables. A table can only have one full-text index, so each table with a full-text index belongs to only one full-text catalog.

Full-text catalogs and indexes are not stored in the database to which they belong. The catalogs and indexes are managed separately by the Microsoft Search service.

A full-text index must be defined on a base table; it cannot be defined on a view, system table, or temporary table. A full-text index definition includes:

- A column that uniquely identifies each row in the table (primary or candidate key) and does not allow NULLs.
- One or more character string columns which are covered by the index.

The full-text index is populated with the key values. The entry for each key has information about the significant words (noise-words or stop-words are stripped out) that are associated with the key, the column they are in, and their location in the column.

Transact-SQL has two new predicates for testing rows against a full-text search condition:

- CONTAINS
- FREETEXT

Transact-SQL also has two new functions that returns the set of rows that match a full-text search condition:

- CONTAINSTABLE

FREETEXTTABLE

Internally, SQL Server sends the search condition to the Microsoft Search service. The Microsoft Search service finds all the keys that match the full-text search condition and returns them to SQL Server. SQL Server then uses the list of keys to determine which table rows are to be processed.

FILLFACTOR and PAD_INDEX

If a SQL Server database is experiencing a large amount of insert activity, you should plan to provide and maintain open space on index and data pages to prevent page splitting. Page splitting occurs when an index page or data page can no longer hold any new rows and a row must be inserted into the page because of the logical ordering of data defined in that page. When this occurs, SQL Server must divide the data on the full page and move about half of the data to a new page so that both pages have some open space. This consumes system resources and time.

The DBCC SHOWCONTIG statement can reveal excessive page splitting on a table. Scan Density, a key indicator that DBCC SHOWCONTIG provides, should be a value as close to 100 percent as possible. If this value is below 100 percent, rebuild the clustered index on that table by using the DROP_EXISTING option to defragment the table. The DROP_EXISTING option of the CREATE INDEX statement permits re-creating existing indexes and provides better index rebuild performance than dropping and re-creating the index. For more information, see *SQL Server Books Online*.

The FILLFACTOR option on the CREATE INDEX and DBCC DBREINDEX statements provides a way to specify the percentage of open space to leave on index and data pages. The PAD_INDEX option for CREATE INDEX applies what has been specified for FILLFACTOR on the nonleaf-level index pages. Without the PAD_INDEX option, FILLFACTOR mainly affects the leaf-level index pages of the clustered index. You should use the PAD_INDEX option with FILLFACTOR.

Transact-SQL

Transact-SQL is the main enabler of programmatic functionality within the relational database. Transact-SQL is very closely integrated with SQL while adding programming capabilities not already standardized within the SQL database programming language. At the same time Transact-SQL extends SQL, it also integrates seamlessly with it.

Datatypes for Variables and Constants

Transact-SQL lets you declare and then utilize local variables and constants within a Transact-SQL object. These variables and constants must be of a datatype known to the database, like VARCHAR or INT. Special-purpose datatypes exist within Transact-SQL to provide a special functionality. For example, the IDENTITY datatype is used to store an automatically increasing numeric counter within the column of a given table. TIMESTAMP is another special-purpose Transact-SQL datatype that is used to stamp each record of a given table with a unique marker.

Additionally, you can construct your own special-purpose user-defined datatypes on top of an existing system datatype. These user-defined datatypes enable you to constrain the use of, or data stored within, a variable or constant of your special datatype.

Present in Microsoft SQL Server Version 7.0 are several new Unicode datatypes, including NCHAR, NVARCHAR, and NTEXT. These new datatypes allow data storage that is independent of the local language character set. Other new datatypes found in Microsoft SQL Server 7.0 include the UNIQUEIDENTIFIER, which uniquely distinguishes each row in a table even across multiple servers, and the CURSOR datatype, which facilitates the use of cursors within a Transact-SQL program.

Programmer-Defined Temporary Objects

Transact-SQL provides several useful temporary objects that endure only as long as the current session. These temporary objects include tables and stored procedures. The objects are stored in the tempdb database and can be used for short-term applications that need exist only while a user is connected or a system operation is executing. Temporary tables can have all the trimmings of their more permanent counterparts, including indexes, constraints, and triggers. Temporary tables are very useful as a sort of "holding tank" where data can be massaged and manipulated into a more useful format.

Here's an example of the definition of a temporary table that also populates the temporary table with data:

```
SELECT *
INTO #temp_titles
FROM titles
WHERE NOT EXISTS
    (SELECT * FROM sales WHERE title_ID = titles.title_ID)
```

Using the speedy SELECT . . . INTO syntax enables you to quickly create and populate a temporary (or permanent) table with data. There are good and bad times to use the SELECT . . . INTO statement. Temporary tables are designated by a single pound sign (#) as a prefix. They are usable only by the user or process that created them. Global temporary tables are designated by double pound signs (##). Global temporary tables are available to all users or processes as long as the creating user or process remains connected to the database server.

Specialized Functions and Global Variables

Unlike local variables, which are declared by the programmer in the body of a Transact-SQL program, global variables and functions are provided by Microsoft to augment your programming toolset. Global variables enable you to immediately retrieve a particular piece of system information. For example, the global variable @@connections allows you to see how many connections have been attempted since the last time the server was started. The global variable @@error shows the last error number generated within the current user's session (or a 0 if the last command was successful).

Functions, on the other hand, allow you shortcuts to manipulate and manage data. There are a large number of functions to choose from, but they fall into these basic categories:

Aggregate functions

Aggregate functions compute a single value result, such as the sum or average value of a given column in a table.

Conversion functions

There are a few versatile functions within Transact-SQL that allow you to convert data from one datatype to another or to reformat data.

Date functions

Date functions manipulate DATETIME and SMALLDATETIME values.

Mathematical functions

These functions provide statistical, trigonometric, logarithmic, and exponential capabilities when manipulating numbers.

String functions

A number of string functions allow you to analyze and modify CHAR, VARCHAR, and, in some cases, TEXT columns and variables.

System (iladic) functions

System functions allow you to retrieve information from system tables and parameters. In this usage, they are somewhat like global variables. Some system functions also provide a way to perform logical comparison of data.

Text and image functions

Manipulation of TEXT and IMAGE data differs from other types, because they have several usage restrictions. Transact-SQL provides several specialized functions to support the use and manipulation of these datatypes.

System and Extended Stored Procedures

In addition to the wide variety of system functions and global variables, Microsoft supplies system stored procedures and extended stored procedures, which provide ready-made programs and/or powerful extensions to Transact-SQL. Here are some examples of system and extended stored procedures:

sp_adduser

A system stored procedure that allows you to add a new user to a database

sp_help

A system stored procedure that shows detailed information about a wide variety of database objects

sp_lock

A system stored procedure that details the locking by user and system processes within the database server

sp_who

A system stored procedure that shows what user and system processes are active within the database server

xp_cmdshell

An extended stored procedure that allows you to execute any operating-system command-line executable

Control-of-Flow Operations

Transact-SQL provides procedural extensions to SQL that provide several different control-of-flow structures:

- Conditional processing with IF . . . ELSE
- Iterative processing with WHILE
- Branching control with GOTO
- Delay control with WAITFOR

Conditional processing with the IF . . . ELSE construct allows your Transact-SQL program to make choices. You can nest your IF . . . ELSE conditionals as deep as you'd like. The IF . . . ELSE construct requires the use of a BEGIN . . . END block if more than one SQL or Transact-SQL command is dependent upon the condition. The result of this coding constraint is a structured, block orientation. Here's an example of an IF . . . ELSE statement:

```
-- Simple Boolean comparison with ELSE clause.  
IF (user_name() = 'sa')  
    PRINT 'Congratulations. You are SA on this system.'  
ELSE
```

```
PRINT 'You are not the SA on this system.'
```

The Transact-SQL WHILE extension provides iterative controls that allow a single block of Transact-SQL code to be repeated. You can even nest loops within other loops. In the simple example that follows, an output line is constructed and printed 12 times, once for each month. This example shows what a WHILE loop looks like:

```
DECLARE @month INT, @full_date VARCHAR(30), @line VARCHAR(255)
SELECT @month=0
WHILE @month <= 12
BEGIN
    -- increment a variable for the month
    SELECT @month = @month + 1
    -- build a variable for the full date
    SELECT @full_date = RTRIM(CONVERT(CHAR(2), @month)) + '/01/99'
    -- build the output line
    SELECT @line = 'Processing for date: ' + @full_date
    -- print the output line
    PRINT @line
END
```

You have some other useful control-of-flow statements in the GOTO command and the WAITFOR command. GOTO transfers control from one executable statement to another labeled section in the current Transact-SQL program. The WAITFOR command provides Transact-SQL programs with delay controls. Here's a Transact-SQL program that illustrates both commands:

```
-- The line with the colon at the end is a GOTO label.
start_of_routine:

-- Let's check today's date.
IF GETDATE() > 'Jan 1 1999 12:00 AM'
BEGIN
    PRINT 'Ending'
    GOTO end_of_routine
END

ELSE
BEGIN
    PRINT 'Waiting'
    -- The WAITFOR DELAY command will make the program wait 1 hour.
    WAITFOR DELAY '01:00:00'
    GOTO start_of_routine
END

end_of_routine:
```

Row-Based Operations Using Cursors

As we discussed earlier, one of the most potent features of relational database management systems is their ability to process data in sets of records rather than on a row-by-row basis. Set-based processing is very fast, but it is limiting in that you cannot selectively modify a single record within a given result set. Transact-SQL has extended this set-based functionality to row-based operations using cursors.

Cursors allow you a high degree of control over the manipulation of data within your database. In a sense, a cursor is a SELECT statement that steps through its result set one record at a time. To properly use a cursor, you must declare, open, fetch from, and close a cursor. In many cases, you also should deallocate the

cursor. Interestingly, you can use a cursor to scroll forward or backward through a result set. You also can set a cursor to step through a table one record at a time or to skip along several records per step.

The following example shows you all the steps needed to properly utilize a cursor:

```
-- declare the cursor
DECLARE titles_cursor CURSOR
FOR
SELECT title_id
FROM titles

-- declare the variables that will hold values retrieved by the cursor
DECLARE @title_id CHAR(6)

-- fetch the next set of values retrieved by the cursor
FETCH titles_cursor INTO @title_id

SELECT @title AS 'Name of book'

-- close and deallocate the cursor
CLOSE titles_cursor
DEALLOCATE CURSOR titles_cursor
```

Error Handling

Transact-SQL allows you to detect the occurrence of errors in the course of executing a program. Errors in Transact-SQL fall into three categories: informational, warning, and fatal.

- Informational errors output a message but do not cause the program to abort.
- Warning messages output an error message and abort the currently executing SQL or Transact-SQL statement but do not abort the entire program or Transact-SQL batch.
- Fatal errors are baaaad. They send an error message and a notification to the operating system error log. Furthermore, fatal errors terminate the Transact-SQL program where the error occurred.

Transact-SQL uses a linear code model for error handling. So, if you don't check for a particular level or type of error, Transact-SQL will not provide any special response to it. For that reason, it is very important for you to properly check and evaluate error conditions throughout the execution of your Transact-SQL programs.

The following Transact-SQL block checks to see if certain data exists within the database. If the data doesn't exist, then an ad hoc error message is raised:

```
IF EXISTS (SELECT * FROM authors WHERE au_fname = 'Elmer'
AND au_lname 'Fudd')
PRINT "Happy Hunting, it's duck season!"
ELSE RAISERROR('Warning! Duck hunting widout a wicense!',16,1)
```

Microsoft also provides you with the ability to add your own user-defined error messages into the system error message table, sysmessages. This capability allows you to define a set of error messages specifically for your application and embed them in the database, rather than solely within the Transact-SQL program logic.

Transact-SQL Objects and Transaction Control

One of the key features of Microsoft SQL Server is the ability to store Transact-SQL objects (triggers, views, and stored procedures) within the database itself. These blocks of Transact-SQL code can then be invoked

by any user or system session that has adequate privileges to perform the task. In fact, Microsoft supports the use of remote procedure calls (RPCs), allowing a single local server to invoke a stored procedure on a remote server. Through Transact-SQL objects, the database becomes a repository for both application code and data. Transact-SQL objects are stored in their own memory area, the procedure cache, within the database's memory space.

At a more elemental level, transactions can be closely controlled and monitored in a Transact-SQL program using the global variables @@transtate and @@trancount. A transaction is a SQL or Transact-SQL statement that produces a measurable unit of work by retrieving, inserting, deleting, or modifying data. You can allow Transact-SQL to use implicit transactions, or you can explicitly define a transaction using the BEGIN TRAN, COMMIT TRAN, and ROLLBACK TRAN statements. By properly gauging and constructing your transactions, you can greatly reduce or even eliminate the possibility of blocks in an application supporting multiple concurrent connections.

Tracing and Debugging

A wide variety of tracing and debugging features are supplied by Transact-SQL using the DBCC and SET commands. In addition to providing tracing and debugging capabilities, these commands allow you to control server behavior, monitor performance, and alter the session environment and behavior.

The DBCC command offers functions that fall into several general categories:

Database consistency checks

Commands like DBCC NEWALLOC and DBCC UPDATEUSAGE check a given database for inconsistencies in the data and, in some cases, take remedial action.

Database and server configuration

In dire straits, you can use DBCC command to drop or shrink databases.

Debugging

Common debugging applications for the DBCC command include controlling tracing functions and examining the input and output buffer of various system and user sessions on the database server.

Performance tuning and monitoring

A wide variety of DBCC commands allow you to monitor the behavior and performance of the server. Other commands enable you to pin tables into memory, rebuild indexes, reestablish fillfactors, or display the selectivity and value of a given index.

Conversely, the SET command enables you to mandate specific behaviors for a given session or user connection. You can use the SET command to enable or disable certain ANSI standards, determine deadlock resolution preference, and perform a wide range of other functionalities.

Special Aggregate Operators

Transact-SQL has grown to support new aggregate operators, such as ROLLUP and CUBE, that are specifically optimized for very large databases, such as those found in data marts and data warehouses.

CUBE and ROLLUP are optional operators of the GROUP BY clause of a SELECT statement. A SELECT query using GROUP BY WITH CUBE returns all rows that it would return without the CUBE switch. In addition, it produces summary rows for every possible combination of GROUP BY keys. A query using WITH ROLLUP returns just a subset of the rows produced by WITH CUBE. ROLLUP is useful in computing running sums and running averages. Unlike CUBE, which computes aggregates for any possible combination of keys of GROUP BY, ROLLUP is sensitive to the order of items in the GROUP BY.

Here's an example of a SELECT statement and result set with a SUM aggregation and a simple GROUP BY clause:

```
SELECT type, pub_id, SUM(ytd_sales)
FROM titles
GROUP BY type, pub_id
```

The query returns this result set:

type	pub_id	Sum
business	0736	18722
business	1389	12066
mod_cook	0877	24278
popular_comp	1389	12875
psychology	0736	9564
psychology	0877	375
trad_cook	0877	19566
UNDECIDED	0877	(null)

Yet if we add one simple CUBE operator, we get this:

```
SELECT type, pub_id, SUM(ytd_sales)
FROM titles
GROUP BY type, pub_id
WITH CUBE
```

The result set now contains extra rows with the type of NULL per pub_id and a grand total of all titles sold:

type	pub_id	Sum
business	0736	18722
business	1389	12066
business	(null)	30788
mod_cook	0877	24278
mod_cook	(null)	24278
popular_comp	1389	12875
popular_comp	(null)	12875
psychology	0736	9564
psychology	0877	375
psychology	(null)	9939
trad_cook	0877	19566
trad_cook	(null)	19566
UNDECIDED	0877	(null)
UNDECIDED	(null)	(null)
(null)	(null)	97446
(null)	0736	28286
(null)	0877	44219
(null)	1389	24941

Finally, if we switch to the ROLLUP functions (CUBE and ROLLUP are mutually exclusive), we get this:

```
SELECT type, pub_id, SUM(ytd_sales)
FROM titles
GROUP BY type, pub_id
WITH ROLLUP
```

returning this result set:

type	pub_id	Sum
business	0736	18722

```

business 1389 12066
business (null) 30788
mod_cook 0877 24278
mod_cook (null) 24278
popular_comp 1389 12875
popular_comp (null) 12875
psychology 0736 9564
psychology 0877 375
psychology (null) 9939
trad_cook 0877 19566
trad_cook (null) 19566
UNDECIDED 0877 (null)
UNDECIDED (null) (null)
(null) (null) 97446

```

Queries

Designing queries that minimize physical and logical I/O as well as balance processor and I/O time is the goal of efficient query design. In essence, this means you want to design queries that result in the use of indexes, result in the fewest disk reads and writes, and make the most efficient use of memory and CPU resources.

The following guidelines, which are derived from the optimization strategies of the SQL Server optimizer, will aid in the design of efficient queries. Before we can discuss the query design guidelines, a few definitions are necessary.

Definitions

Table Scan—A table scan occurs when the SQL Server optimizer can find no efficient index to satisfy a clause or when a clause is nonoptimizable. When the table scan method is used, execution begins with the first row in the table. Each row is retrieved and compared with the conditions in the WHERE clause and then returned to the client if it meets the given criteria. Regardless of how many rows qualify, every row in the table must be looked at; so for very large tables, a table scan can be costly in terms of page I/Os.

Worktable—For some types of queries, such as those that require the results to be ordered or displayed in groups, the SQL Server query optimizer might determine that it is necessary to create its own temporary worktable. The worktable holds the intermediate results of the query, at which time the results can be ordered or grouped, and then the final results can be selected from that worktable. When all results have been returned, the worktable is automatically dropped.

Worktables are always created in the **tempdb** database, so it is possible that the system administrator might have to increase the size of **tempdb** to accommodate queries that require very large worktables. Because the query optimizer creates these worktables for its own internal use, the names of the worktables will not be listed in the **tempdb.sysobjects** table.

Designing Queries with Transact-SQL

Below is a summary of the common Transact-SQL statements that you would use to construct basic queries.

ALTER DATABASE- Adds or removes files and filegroups for a database. Can also be used to modify the attributes of files and filegroups, such as changing the name or size of a file.

ALTER DATABASE *database*

ALTER PROCEDURE - Alters a previously created procedure (created by executing the CREATE PROCEDURE statement) without changing permissions and without affecting any dependent stored procedures or triggers.

ALTER PROC[EDURE] *procedure_name* [*;number*]

ALTER TABLE - Modifies a table definition by altering, adding, or dropping columns and constraints, or by disabling or enabling constraints and triggers.

ALTER TABLE *table*

ALTER TRIGGER - Alters the definition of a trigger created previously by the CREATE TRIGGER statement. For more information about the parameters used in the ALTER TRIGGER statement, see “CREATE TRIGGER” in this volume.

ALTER TRIGGER *trigger_name*

ALTER VIEW - Alters a previously created view (created by executing CREATE VIEW) without affecting dependent stored procedures or triggers and without changing permissions. For more information about the parameters used in the ALTER VIEW statement, see “CREATE VIEW” in this volume.

ALTER VIEW *view_name* [(*column* [,...*n*])]

CREATE DATABASE - Creates a new database and the files used to store the database, or attaches a database from the files of a previously created database.

CREATE DATABASE *database_name*

CREATE DEFAULT - Creates an object called a default. When bound to a column or a user-defined data type, a default specifies a value to be inserted into the column to which the object is bound (or into all columns, in the case of a user-defined data type) when no value is explicitly supplied during an insert. Defaults, a backward compatibility feature, perform some of the same functions as default definitions created using the DEFAULT keyword of ALTER or CREATE TABLE statements. Default definitions are the preferred, standard way to restrict column data because the definition is stored with the table and automatically dropped when the table is dropped. A default is beneficial, however, when the default is used multiple times for multiple columns.

CREATE DEFAULT *default*
AS *constant_expression*

CREATE INDEX - Creates an index on a given table. This statement either changes the physical ordering of the table or provides the optimizer with a logical ordering of the table to increase query efficiency. When an index is created for the primary key, use the table- and column-level PRIMARY KEY constraints by specifying the PRIMARY KEY keywords provided with either the CREATE TABLE or ALTER TABLE statements.

Only the table owner can create indexes on that table. The owner of a table can create an index at any time, whether or not there is data in the table. Indexes can be created on tables in another database by specifying a qualified database name.

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX *index_name* ON *table* (*column* [,...*n*])

CREATE PROCEDURE - Creates a stored procedure (a saved collection of Transact-SQL statements) that can take and return user-supplied parameters. Procedures can be created for permanent use or for temporary use within a user’s session (local temporary procedure) or for temporary use within all user’s sessions (global temporary procedure). Stored procedures can also be created to run automatically when Microsoft SQL Server starts.

CREATE PROC[EDURE] *procedure_name* [*;number*]

CREATE TABLE - Creates a new table.

CREATE TABLE

CREATE TRIGGER - Creates a trigger, which is a special kind of stored procedure that executes automatically when a user attempts the specified data-modification statement on the specified table. Microsoft SQL Server allows the creation of multiple triggers for any given INSERT, UPDATE, or DELETE statement.

CREATE TRIGGER *trigger_name*
ON *table*

CREATE VIEW - Creates a virtual table that represents the data in one or more tables in an alternative way. Views can be used as security mechanisms by granting permission on a view but not on the underlying (base) tables.

CREATE VIEW *view_name* [(*column* [...*n*])]

FROM - Specifies the tables, views, derived tables, and joined tables used in DELETE, SELECT, and UPDATE statements.

[FROM {<table_source>} [...*n*]]

GROUP BY - Divides a table into groups. Groups can consist of column names or results or computed columns. For more information, see “SELECT” in this volume.

GROUP BY <variable>

INSERT - Adds a new row to a table or a view.

INSERT [INTO]

NEWID - Creates a unique value of type **uniqueidentifier**.

NEWID()

ORDER BY - Specifies the sort order used on columns returned in a SELECT statement. For more information, see “SELECT” in this volume.

ORDER BY <variable>

PERMISSIONS - Returns a value containing a bitmap that indicates the statement, object, or column permissions for the current user.

PERMISSIONS([*objectid* [, '*column*']])

SELECT - Retrieves rows from the database and allows the selection of one or many rows or columns from one or many tables.

SELECT statement ::=

UNION - Combines the results of two or more queries into a single result set consisting of all the rows belonging to all queries in the union. For more information, see “SELECT” in this volume.

UNION [ALL]

UPDATE - Changes existing data in a table.

UPDATE *table_name* WITH (<table_hint_limited> [...*n*])
| *view_name*
| *rowset_function_limited*

WHERE - Specifies the condition for the rows returned by a query.

WHERE <search_condition>

WHILE- Sets a condition for the repeated execution of an SQL statement or statement block. The statements are executed repeatedly as long as the specified condition is true. The execution of statements in the WHILE loop can be controlled from inside the loop with the BREAK and CONTINUE keywords.

WHILE *Boolean_expression*
 {sql_statement | statement_block}

SQL Server Performance Tuning Tools

Microsoft SQL Server version 7.0 includes several tools that can assist database administrators with performance tuning.

SQL Server Profiler

SQL Server Profiler records detailed information about activity occurring on the database server. SQL Server Profiler can be configured to watch and record one or many users executing queries on SQL Server and to provide a widely configurable amount of performance information, including I/O statistics, CPU statistics, locking requests, Transact-SQL and RPC statistics, index and table scans, warnings and errors raised, database object create/drop, connection connect/disconnects, stored procedure operations, cursor operation, and more. For more information about what SQL Server Profiler can record, see SQL Server Books Online.

Using SQL Server Profiler with Index Tuning Wizard

SQL Server Profiler and Index Tuning Wizard can be used together to help database administrators create proper indexes on tables. SQL Server Profiler records resource consumption for queries into a .trc file. The .trc file can be read by Index Tuning Wizard, which evaluates the .trc information and the database tables, and then provides recommendations for indexes that should be created. Index Tuning Wizard can either automatically create the proper indexes for the database by scheduling the automatic index creation or generate a Transact-SQL script that can be reviewed and executed later.

SQL Server Profiler and Index Tuning Wizard are powerful tools in database server environments in which there are many tables and queries. Use SQL Server Profiler to record a .trc file while the database server is experiencing a representative set of queries. Then load the .trc file into Index Tuning Wizard to determine the proper indexes to build. Follow the prompts in Index Tuning Wizard to automatically generate and schedule index creation jobs to run at off-peak times. Run SQL Server Profiler and Index Tuning Wizard regularly (perhaps weekly) to see if queries executed on the database server have changed significantly, thus possibly requiring different indexes. Regular use of SQL Server Profiler and Index Tuning Wizard can keep SQL Server running in top form as query workloads change and database size increase over time.

SQL Server Query Analyzer

After the information is recorded into the SQL Server table, you can use SQL Server Query Analyzer to determine which queries on the system are consuming the most resources, and database administrators can concentrate on improving the queries that need the most help. For example, this query is typical of the analysis performed on data recorded from SQL Server Profiler into a SQL Server table:

```
SELECT TOP 3 TextData,CPU,Reads,Writes,Duration FROM profiler_out_table ORDER BY cpu desc
```

The query retrieves the top three consumers of CPU resources on the database server. Read and write I/O information, along with the duration of the queries in milliseconds is returned. If a large amount of

information is recorded with the SQL Server Profiler, you should create indexes on the table to help speed analysis queries. For example, if CPU is going to be an important criteria for analyzing this table, you should create a nonclustered index on CPU column.

Showplan

Showplan can be used to display detailed information about what the query optimizer is doing. SQL Server 7.0 provides text and graphical versions of Showplan. Graphical Showplan output can be displayed in the Results pane of SQL Server Query Analyzer by executing a Transact-SQL query with **Ctrl+L**. Icons indicate the operations that the query optimizer will perform if it executes the query. Arrows indicate the direction of data flow for the query. Details about each operation can be shown by holding the mouse pointer over the operation icon. The equivalent information can be returned in text-based showplan by executing SET SHOWPLAN_ALL ON. To reduce the query optimizer operation details from text-based showplan, execute SET SHOWPLAN_TEXT ON.

Examples of Showplan Output

This section shows sample showplan plan output using the following example queries and the **set showplan_text on** option in SQL Server Query Analyzer.

Query:

```
SELECT ckey1,col2 FROM testtable WHERE ckey1 = 'a'
```

Text-based showplan output:

```
--Clustered    Index    Seek(OBJECT:([test].[dbo].[testtable].[testtable2]),    SEEK:([testtable].[ckey1]='a')
ORDERED)
```

This query takes advantage of the clustered index on column **ckey1**, as indicated by Clustered Index Seek.

SQL 7.0 Design QUESTIONS

1: You create a stored procedure and a Visual Basic user with SQL authentication is trying to execute it, but can not. Why?

A: Need to make user part of a NT group that has access to the stored procedure.

2: You have a query that is run frequently and has a lot of overhead. What could you do to increase performance?

A: Make it a stored procedure.

3: You would like to create a view of a table for users to input data. The view must restrict the maximum entry to 100,000. How can this be done without the use of a database trigger?

A: Create and bind a rule

4: You have an OLTP server. You want to create an decision support database without loading any more on the OLTP server. How could you do this?

A: Create a second database on another server and use replication.

5: You have to transfer data into a table with identity from a text file that has a combined person name. Your table has separate lastname, firstname fields. How would you do this?

A: DTS

6: In your results you want to display the character string 'The name of this product is' immediately before the product name. Which of the following SQL SELECT statements could you use?

A: *SELECT 'The name of this product is', prodname FROM products*

7: In the pubs database the titleauthor table is used to define the many-to-many relationships between authors and books. Which of the following SQL SELECT statements will show which books (titles) have more than one author?

A: *SELECT DISTINCT au_id, title_id
FROM titleauthor
WHERE title_id = title_id AND au_id <> au_id*

8: Your company has a HQ office and 3 Regional offices (RG1, RG2, RG3). Each office has an employee tables with the following columns: emp_id, salary, hire_date, department, s_number, job_id . You want to write a view that gives the following results:

Office emp_id department hire_date

Which of the following would you use to create a view?

A: *SELECT 'HQ' Office , emp_id , department , hire_date
FROM HQ
UNION ALL
SELECT 'RG1' Office , emp_id , department , hire_date
FROM RG1
UNION ALL
SELECT 'RG2' Office , emp_id , department , hire_date
FROM RG2
UNION ALL
SELECT 'RG3' Office , emp_id , department , hire_date
FROM RG3*

9: You are creating a table named recruit to track the status of potential new employees. The SocialSecurityNo column must not allow null values. However, a value for a potential employee's Social Security number is not always known at the time of initial entry. You want the database to populate the SocialSecurityNo column with a value of UNKNOWN when a recruiter enters the name of a new potential employee without a Social Security number. How can you accomplish this task?

A: *Create a default definition on the SocialSecurityNo column.*

10: You are designing a data model to track research projects. A project might be undertaken in one research institution or in multiple institutions. Some research institutions are part of a university or a larger research institution. Each project is assigned a group of scientists, who might perform different jobs in each project. A scientist who is a member of the staff at one institution might also be assigned to a project that is being undertaken at another institution.

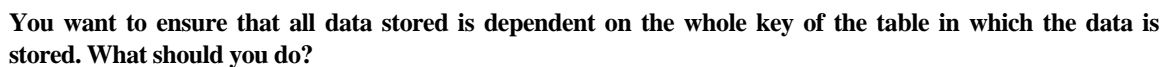
You want to accomplish the following goals:

**Ensure that all the scientists conducting research for any specific project can be reported.
Ensure that a scientist's job for a specific project in a specific institution can be reported.
Ensure that all the institutions participating in any specific project can be reported.**

You design the logical model as shown in the exhibit.



11: Your database is used to store information about each employee in your company and the department in which each employee works. An employee can work in only one department. The database contains two tables, which are named Department and Employee. The tables are modeled as shown in the exhibit.



12: You are developing a Personnel database for your company. This database includes an employee table that is defined as follows:

31

Your company provides each employee with a telephone extension number and an e-mail address. Each employee must have a unique telephone extension number and a unique e-mail address. In addition, you must prevent duplicate Social Security numbers from being entered into the database. How can you alter the table to meet all of the requirements?

A: *ALTER TABLE employee*

ADD CONSTRAINT u_nodupssn UNIQUE NONCLUSTERED (SocialSecurityNo)

ALTER TABLE employee

ADD CONSTRAINT u_nodupext UNIQUE NONCLUSTERED (Extension)

ALTER TABLE employee

ADD CONSTRAINT u_nodupemail UNIQUE NONCLUSTERED (EmailAddress)

13: You are implementing a logical data model for an online transaction processing (OLTP) application. One entity from the logical model is in third normal form and currently has 10 attributes. One attribute is the primary key. Six of the attributes are foreign key references into six other entities. The last three attributes represent columns that hold numeric values. How should this entity from the logical model be implemented?

A: *Create the table as described in the logical data model.*

14: You are designing an inventory database application for a national automobile sales registry. This new database application will keep track of automobiles available at a participating dealerships, and will allow each dealership to sell automobiles from the inventories of other dealerships. Many makes and models of automobiles will be shown from each dealership. You want to be able to track information about each automobile. You want to normalize your database. Which tables should be included in the database application? (Choose all that apply)

A: *a table containing the list of all dealerships along with the address and identification number for each dealership*

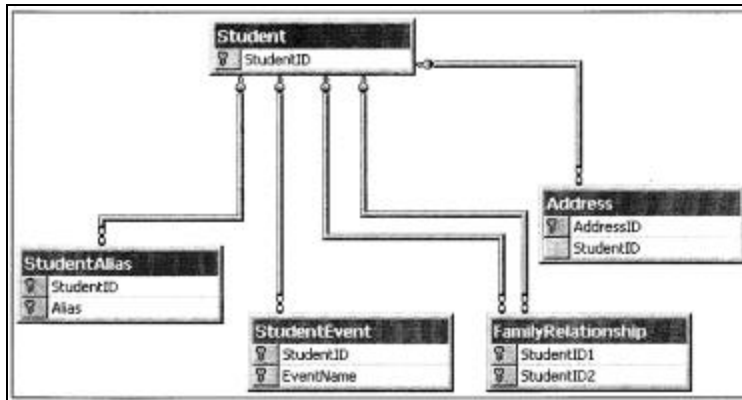
a table containing an identification number for each automobile, the owning dealership's identification number, and other information specific to each automobile.

15: You are designing the data model to maintain information about students living in a group home. A student might be known under many aliases, and the data model needs to associate each student with aliases and other descriptors such as hair color, weight, religion, physical handicaps, and ethnicity. Some students have siblings or other relatives who are also in the group home, and these family relationships need to be tracked. Multiple addresses might be associated with an individual student, such as the current address, a school address, and addresses of significant relatives. You also need to track significant events in a student's life, which might include attendance at a special training session, medical treatment, a graduation, or a death in the family.

You want to accomplish the following goals:

- Ensure that any kind of descriptor can be associated with a student.
- Ensure that multiple addresses can be associated with multiple students, and that the address usage can be reported.
- Ensure that any family relationship with another student can be reported.
- Ensure that all known aliases for a student can be reported.
- Ensure that significant events in a student's life can be reported.

You design the logical model as shown in the exhibit. Which result or results does this model produce? (Choose all that apply.)



- A: Any kind of descriptor can be associated with a student.
 Multiple addresses can be associated with multiple students, and the address usage can be reported.
 Any family relationship with another student can be reported.
 All known aliases for a Student can be reported.
 Significant events in a student's life can be reported.

16: You are designing a distributed data model for an international importing and exporting company. The company has sales offices in London, Nairobi, and Cairo, with headquarters in New York. Sales are recorded in the local currency in the local database, but the United States dollar is used as a common currency. London and Nairobi have a value-added tax, but Cairo does not.

The company's New York headquarters credits a sale to the local office in United States dollars at the time of the sale, but reports its monthly sales in United States dollars converted at the end of the month. Because of fluctuations in the exchange rate, there can be differences in the United States dollar value of a sale between the date of the sale and the date of the monthly report.

You want to accomplish the following goals:

- Ensure that all sales record primary keys are unique throughout the distributed database.
- Ensure that a sales record created in London or Nairobi includes a value-added tax, but that a sales record created in Cairo does not.
- Ensure that the local sales amounts can be calculated in United States dollars at the time of the sale.
- Ensure that the local sales amounts can be calculated in United States dollars at the end of the month.
- Ensure that the difference between the value of a sale in United States dollars at the time of the sale and the value at the end of the month can be calculated.

You take the following actions:

- Create a ConversionRate table with a currency column, a Rate Column; and a RateValidDate column.
- Add a PRIMARY KEY constraint to the ConversionRate table on the Currency column and the RateValidDate column.
- Add a CHECK constraint to the ConversionRate table rejecting a RateValidDate date that is later than the current date
- Create a Sales table with a uniqueidentifier column named SalesID, a SalesAmount column, a nonnullable TaxAmount column, and a SalesDate column.

- Add a **DEFAULT** constraint to the Sales table on the SalesID column that uses the NEWID function.
- Add a **PRIMARY KEY** constraint to the Sales table on the SalesID column.

Which result or results do these actions produce? (Choose all that apply.)

*A: All sales record primary keys are unique throughout the distributed database.
A sales record created in London or Nairobi includes a value-added tax, but a sales record created in Cairo does not.*

17: You are the database manager for a manufacturing company. You are reviewing the mixture of online transaction processing (OLTP) and decision support system (DSS) activities that access the same information in a new production monitoring application. The application is designed to capture production information from the shop floor in real time. The application must report production results, trends, and variances in a timely fashion. What should you do to maximize the overall performance of the production monitoring application?

*A: Create an OLTP SQL Server and a DSS SQL Server.
Replicate tables from the OLTP SQL Server to a DSS SQL Server with the distribution database on the DSS server.
Index the tables on each SQL Server to support their respective activities.*

18: You are designing an insurance database. The policy table will be accessed and updated by several additional applications. In the policy table, you need to ensure that the value entered into the beginning_effective_date column is less than or equal to the value entered into the ending_effective_date column. What should you do?

A: Create a CHECK constraint on the policy table that compares the values.

19: You are implementing a transaction-based application for a credit card company. More than 10 million vendors accept the company's credit card, and more than 100 million people regularly use the credit card. Before someone can make a purchase by using the credit card, the vendor must obtain a credit authorization from your transaction-based application.

Vendors around the world must be able to authorize purchases in less than 30 seconds, 24 hours a day, seven days a week. Additionally, the application must be able to accommodate more vendors in more locations in the future. What should you do to implement the application to meet all of the requirements?

A: Implement an n-tier architecture in which vendors make calls to geographically dispersed Microsoft Transaction Servers (MTS), which would then obtain an authorization code from geographically dispersed SQL Servers.

20: You are testing a marketing application that gathers demographic information about each household in the country. The demographic table in the application contains more than 1,000 columns.

Users report problems regarding the performance of the application. As part of your investigation into these reports, you are reviewing how the logical data model was implemented. Most of the reports relate to long response times when users are updating or retrieving data from the demographic table. Nearly 90 percent of the users search or update 20 of the columns. The remaining columns are seldom used, but they are important.

What should you do to improve data retrieval and update performance when accessing the information in the demographic table?

A: Divide the data in the demographic table into two new tables with one table containing the 20 most accessed columns and the other containing the remaining columns.

21: Your Sales database is accessed by a Microsoft Visual Basic client/server application. The application is not using the Microsoft Windows NT Authentication security model. The Orders table has a primary key consisting of an identity column named OrderID.

When a customer cancels an order, the row for that order is deleted from the Orders table. Sometimes, a customer who canceled an order will ask that the order be reinstated. You must write a stored procedure that will insert the order back into the database with the original OrderID.

You write the following stored procedure to be called by the Visual Basic applicaion:

```
CREATE PROCEDURE    InsertReinstatedOrder
@OrderID           int,
@SalesPersonID     int,
@RegionID          int,
@OrderDate         datetime,
@OrderAmount       money,
@CustomerID        int
AS
SET IDENTITY_INSERT Orders ON
INSERT Orders (OrderID, SalesPersonID, RegionID, OrderDate, OrderAmount, CustomerID)
VALUES(@OrderID, @SalesPersonID, @RegionID, @OrderDate, @OrderAmount, @CustomerID)
SET IDENTITY_INSERT Orders OFF
RETURN
GO
GRANT EXECUTE ON InsertReinstatedOrders TO Sales
GO
```

You test the procedure and it is implemented into the Visual Basic application. A user named Andrew is assigned to the Sales role. Andrew reports that he is receiving an error message indicating that he is having a permissions problem with the procedure. What must you do to solve the problem?

A: Add Andrew to the db_owner role.

22: Your database includes an Orders table that is defined as follows:

```
CREATE TABLE Orders (
    Order ID      int IDENTITY ( 1, 1)      NOT NULL,
    SalesPersonID int                        NOT NULL,
    Region ID     int                        NOT NULL,
    OrderDate     datetime                  NOT NULL,
    OrderAmount   int                        NOT NULL)
```

You have written a stored Procedure named GetOrders that reports on the Orders table. The stored procedure currently creates a list of orders in order by SalesPersonID. You must change the stored procedure to produce a list of orders in order first by RegionID and then by SalesPersonID. Permissions have been granted on the stored procedure, and You do not want to have to grant them again. How must you change the Stored procedure?

A: ALTER PROCEDURE GetOrders
AS
SELECT SalesPersonID, RegionID, OrderID, OrderDate, OrderAmount FROM Orders

ORDER BY RegionID, SalesPersonID

23: Your database includes tables that are defined as follows:

```
CREATE TABLE SalesPerson
(SalesPersonID int IDENTITY (1, 1) NOT NULL
PRIMARY KEY NONCLUSTERED,
RegionID int NOT NULL,
LastName var char (30) NULL,
FirstName varchar (30) NULL,
MiddleName var char (30) NULL,
AddressID int NULL )
CREATE TABLE Orders
(OrderID int IDENTITY (1, 1) NOT NULL
PRIMARY KEY NONCLUSTERED,
SalesPersonID int NOT NULL,
RegionID int NOT NULL,
OrderDate datetime NOT NULL,
OrderAmount money NOT NULL)
```

You need to produce a list of the highest sale for each salesperson on September 15, 1998. The list is to be printed in the following format:

Last Name First Name Order Date Order Amount

Which query will accurately produce the list?

A: *SELECT s.LastName, s.FirstName, o.OrderDate, MAX(OrderAmount)*
FROM SalesPerson AS s
LEFT OUTER JOIN Orders AS o
ON o.SalesPersonID = s. SalesPersonID and o. OrderDate = '09/15/1998'
GROUP BY s.LastName, s.FirstName, o. OrderDate

24: You have a database that keeps track of membership in an organization. Tables in this database include the Membership table, the Committee table, the Address table, and the Phone table. When a person resigns from the organization, you want to be able to delete the membership row and have all related rows be automatically removed. What can you do to accomplish this task?

A: *Create a DELETE trigger on the Membership table that deletes any rows in the Committee, Address, and Phone tables that reference the primary key in the Membership table.*
Do not place a FOREIGN KEY constraint on the Committee, Address, and Phone tables.

25: It is time for the annual sales awards at your company. The awards include certificates awarded for the five sales of the highest dollar amounts. You need to produce a list of the five highest revenue transactions from the Orders table in the Sales database. The Orders table is defined as follows:

```
CREATE TABLE Orders (
OrderID int IDENTITY (1,1) NOT NULL,
SalesPersonID int NOT NULL,
RegionID int NOT NULL,
OrderDate datetime NOT NULL,
OrderAmount int NOT NULL)
```

Which statement will produce the report?

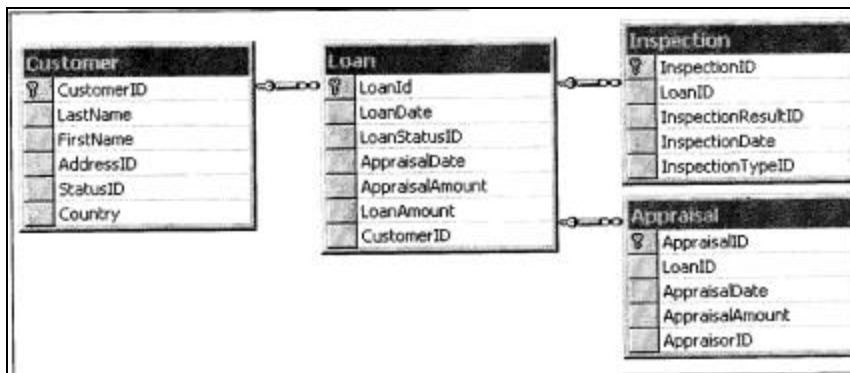
A: *SELECT TOP 5 OrderAmount, SalesPersonID FROM orders ORDER BY OrderAmount DESC*

26: You must reconcile the Checking account for your company. You have a CheckRegister table, an InvalidCheck table, and a ClearedCheck table. The ClearedCheck table lists checks that have cleared the bank.

You must update the ClearedDate column of the CheckRegister table for any checks that are on the ClearedCheck table. If the check is in the ClearedCheck table but not on the CheckRegister table, you must insert a row into the InvalidChecktable. If the amount shown for a check in the ClearedCheck table is different from the amount shown for the same check in the CheckRegister table you must insert a row into the InvalidCheck table. Each row must be deleted from the ClearedCheck table after it has been evaluated for accuracy. Which statement group should you use to accomplish this task in the shortest time?

A: *UPDATE CheckRegister SET CheckRegister.ClearedDate = ClearedCheck.ClearedDate
FROM CheckRegister
JOIN ClearedCheck ON CheckRegister.CheckNumber = ClearedCheck.CheckNumber
AND CheckRegister.CheckAmount = ClearedCheck. CheckAmount
DELETE ClearedCheck
FROM ClearedCheck
JOIN CheckRegister ON CheckRegister.CheckNumber = ClearedCheck.CheckNumber
AND CheckRegister.CheckAmount = ClearedCheck.CheckAmount
INSERT InvalidCheck (CheckNumber, CheckAmount, ClearedDate)
SELECT CheckNumber, CheckAmount, ClearedDate
FROM ClearedCheck
DELETE ClearedCheck*

27: You must write a stored procedure to perform cascading deletes on the HomeLoan database. The client application will pass a parameter containing the CustomerID of the customer to be deleted. A customer might have any number of pending loans. Each pending loan has one or more inspections and one or more appraisals associated with it. The diagram in the exhibit shows the relationship of the tables.



Which stored procedure should you use?

A: *CREATE PROCEDURE LoanCascadeDelete
@CustomerID int
AS
DELETE FROM Appraisal
FROM Appraisal
JOIN Loan ON Appraisal.LoanID = Loan. LoanID
WHERE CustomerID = @CustomerID*

```

DELETE FROM Inspection
FROM Inspection
JOIN Loan ON Inspection.LoanID = Loan.LoanID
WHERE CustomerID = @CustomerID
DELETE FROM Loan
WHERE CustomerID = @CustomerID
DELETE FROM Customer
WHERE CustomerID = @CustomerID

```

28: You have an application that captures real-time stock market information and generates trending reports. In the past, the reports were generated after the stock markets closed. The reports now need to be generated on demand during trading hours. What can you do so that reports can be generated without affecting the rest of the application? (Choose two.)

*A: Program the application to issue the following command before generating a report: set transaction isolation level read uncommitted.
On the stock transaction tables, create triggers that update summary tables instead of performing a data analysis each time a report is generated.*

29: Your Orders table is defined as follows:

```

CREATE TABLE Orders (
    OrderID int IDENTITY (1,1) NOT NULL,
    SalesPersonID int NOT NULL,
    RegionID int NOT NULL,
    OrderDate datetime NOT NULL,
    OrderAmount int NOT NULL)

```

The table is becoming too large to manage. You must delete all sales that are more than three years old. Which query will accomplish the desired result?

A: DELETE FROM Orders WHERE OrderDate < DATEADD(YY, - 3, GETDATE ())

30: Your database includes a table that is defined as follows:

```

CREATE TABLE SalesInformation
(SalesInformationID int IDENTITY (1, 1) NOT NULL,
SalesPersonID int NOT NULL,
RegionID int NOT NULL,
ReceiptID int NOT NULL,
SalesAmount money NOT NULL)

```

You want to populate the table with data from an existing application that has a numeric primary key. In order to maintain the referential integrity of the database, you want to preserve the value of the original primary key when you convert the data. What can you do to populate the table?

A: set the IDENTITY_INSERT option to ON, and then insert the data by using a SELECT statement that has a column list.

31: You increase the number of users of your customer service application from 20 to 120. With the additional users, the response time when retrieving and updating has slowed substantially. You examine all the queries and indexes and discover that they are all fully optimized. The application seems to run properly as long as the number of users is less than 50. What can you do to resolve the problem?

A: Ensure that the application is using an optimistic locking strategy instead of a pessimistic locking strategy.

32: You have two SQL Servers supporting two separate applications on your network. Each application uses stored procedures for all data manipulation, You need to integrate parts of the two applications. The changes are limited to a few stored procedures that need to make calls to remote stored procedures. What should you do to implement calls to remote stored procedures?

A: Add the remote server as a linked server. Fully qualify the remote procedure name.

33: You have a database that is accessed by many different applications written with many different development tools. Many applications were originally written to work against any relational database system that complies with ANSI 92. Each application uses a different mechanism for accessing the database. ODBC, DB-Library, and ADO are all used simultaneously.

Transaction processing is not uniform across all your applications. How can you ensure that all applications handle transactions in the same fashion?

A: Program all applications to issue the SET ANSI DEFAULTS ON command immediately after establishing a user connection.

34: Your server named Corporate has a Sales database that stores sales data for a software distribution company. Two remote servers named New York and Chicago each store sales data in a salesorder table relative only to their respective sales territories. The salesorder table on the Corporate server is updated once a week with data from the remote servers.

The salesorder table on each server, including Corporate, is defined as follows:

```
CREATE TABLE salesorder (  
Number          char (10)          NOT NULL,  
CustomerName    varChar (100)      NOT NULL,  
TerritoryName   varchar (50)       NOT NULL,  
EntryDate       datetime           NOT NULL,  
Amount          money              NOT NULL)
```

You need to create a view that shows a current list of all sales from the New York and Chicago sales territories, and the list should have the following format:

Territory Customer Date Amount

Which view can you create to show all sales from the New York and Chicago sales territories in the required format?

*A: CREATE VIEW SalesSummary_view (Territory, Customer, Date, Amount) AS
SELECT TerritoryName, CustomerName, EntryDate, Amount
FROM NewYork.Sales.dbo.salesorder
UNION ALL
SELECT TerritoryName, CustomerName, EntryDate, Amount
FROM Chicago.Sales.dbo.salesorder*

35: You have a database to keep track of sales information. Many of your queries calculate the total sales amount for a particular salesperson. You are working with a nested procedure that will pass a parameter back to the calling procedure containing the total sales as follows:

```

CREATE PROCEDURE GetSalesPersonData
    @SalesPersonID      int,
    @RegionID           int,
    @SalesAmount         money OUTPUT
AS
SELECT @SalesAmount = SUM (SalesAmount)
FROM SalesInformation
WHERE @SalesPersonID = SalesPersonID

```

Which statement will accurately execute the procedure and receive the required value?

A: *EXECUTE GetSalesPersonData 1,1,@SalesAmount OUTPUT*

36: You automate the backup and recovery process for your database application. After the database is restored, you discover that queries that use the FREETEXT and CONTAINS keywords no longer return the expected rows. What should you do?

A: *Add a job to the restoration process to re-create and populate the full-text catalog.*

37: You are developing an application for a worldwide furniture wholesaler. You need to create an inventory table on each of the databases located in New York, Chicago, Paris, London, San Francisco, and Tokyo. In order to accommodate a distributed data environment, you must ensure that each row entered into the inventory table is unique across all locations. How can you create the inventory table?

A: *CREATE TABLE inventory (*
Id uniqueidentifier NOT NULL
DEFAULT NEWID (),
ItemName varchar(100) NOT NULL,
ItemDescription varchar(255) NULL,
Quantity int NOT NULL,
EntryDate datetime NOT NULL)

38: Your database of medical information includes a table named Experiments that is defined as follows:

```

CREATE TABLE Experiments (
    ExperimetID char (32),
    Description text,
    Status Integer,
    Results Text)

```

You write the following query:

```
SELECT * FROM Experiments WHERE CONTAINS (Description, 'angina')
```

You are certain that there are matching rows, but you receive an empty result set when you run the query. What should you do? (Choose two.)

A: *Create a full-text catalog that includes the Experiments table.*
Create a scheduled job to populate the full-text catalog.

39: Your users report slow response times when they are modifying data in your transaction processing application. Response times are excellent when users are merely retrieving data. The search criteria used for modifying data are the same as the search criteria for retrieving data. All transactions are short and follow standard guidelines for coding transactions. You monitor blocking locks, and you find out that they

are not causing the problem. What is the most likely cause of the slow response times when modifying data?

A: The transaction log is placed on an otherwise busy disk drive.

40: Your shipping company has a database application that maintains an inventory of items on each vessel. When each vessel is unloaded at its destination, the inventory is counted, and the `arrived_quantity` column is updated in the database. There can be thousands of vessels en route at any one time. Each shipment is identified by a `shipment_id`. Each vessel can carry thousands of items. Each item in a shipment is identified by an `item_number`. You want to make sure the update of the `arrived_quantity` column is as fast as possible. What should you do?

*A: Create a **clustered** index on the `shipment_id` column and the `item_number` column.*

41: You are investigating reported problems regarding the performance of a query in your database. The `WHERE` clause of the query includes search arguments on `columnA`, `columnB`, and `columnC`.

You analyze the data and discover that the content of `columnA` is nearly identical in all rows. The content of `columnB` is the same in about 50 percent of the rows. The content of `columnC` is the same in about 10 percent of the rows. How should you index the table to improve query performance?

*A: Create a composite **clustered** index on `columnC`, `columnB`, `columnA`.*

42: You need to create a development database that will hold 20 MB of data and indexes and a 4-MB transaction log. There are no concerns regarding query performance or log placement with this database. The SQL Server was installed on drive E of the server computer, and there is plenty of disk space on drive E. How should you create the database?

A: `CREATE DATABASE development on primary (name = development1, filename = 'e:\mssql7\data\development1.mdf', size = 20MB) log on (name = developmentlog1, filename = 'e:\mssql7\data\developmentlog1.ldf', size = 4MB)`

43: Your database includes a table named `product`. The `product` table currently has a clustered index on the primary key of the `product_id` column. There is also a nonclustered index on the `description` column. You are experiencing very poor response times when querying the `product` table. Most of the queries against the table include search arguments on the `description` and `product_type` columns. Because there are many values in the `size` column for any product, query result sets usually contain between 200 and 800 rows. You want to improve the response times when querying the `product` table. What should you do?

A: Use the Index Tuning Wizard to identify and build any missing indexes.

44: You are building a decision support system (DSS) database for your company. The new database is expected to include information from existing data sources that are based on Microsoft Access, dBase III, Microsoft Excel, and Oracle.

You want to use SQL Server Agent to run a scheduled job to extract information from the existing data sources into a centralized database on SQL Server 7.0. You do not want to perform any additional programming outside the SQL Server environment. How must you extract the information from the existing data sources?

A: Create a Data Transformation Services package to import data from each data source.

45: You are working on a data conversion effort for a Sales database. You have successfully extracted all of the existing customer data into a tab-delimited flat file. The new customer table is defined as follows:

```
CREATE TABLE customer (  
  Id                int IDENTITY      NOT NULL,  
  LastName          varchar (50)      NOT NULL,  
  FirstName         varchar (50)      NOT NULL,  
  Phone             varchar (15)      NOT NULL,  
  Email             var char (255)    NULL )
```

You need to populate this new table with the customer information that currently exists in a tab-delimited flat file with the following format:

Name	Phone	E-mail
Adam Barr	555-555-1098	abarr@adatum.com
Karen Berge	555-555-7868	kberge~woodgrovebank.corn
Amy Jones	555-555-0192	ajones@tresearch.com

How can you transfer the data to accurately populate the customer table?

A: Import the data by using Data Transformation Services with the Transform information as it is copied to the destination option button selected.

46: You have an application that makes four connections to the SQL Server at the same time. The connections are used to execute SELECT, INSERT, UPDATE, and DELETE statements. Each connection is used to execute one of these four statements.

The application occasionally stops responding when a user is trying to update or delete rows, and then the user must close the application. The problem occurs when a user attempts to execute an UPDATE or DELETE statement after submitting a SELECT statement that retrieves a result set of more than 10,000 rows. What can you do to prevent the problem?

A: On the connection for the SELECT statement, set the transaction isolation level to READ UNCOMMITTED.

47: Your database includes a job_cost table that typically holds 100,000 rows but can grow or shrink by as much as 75,000 rows at a time. The job_cost table is maintained by a batch job that runs at night. During the day, the job_cost table is frequently joined to other tables by many different queries. Your users report that their initial queries are very slow, but then response time improves for subsequent queries. How should you improve the response time of the initial queries?

A: Run the sp_updatestats stored procedure as part of the nightly batch job.

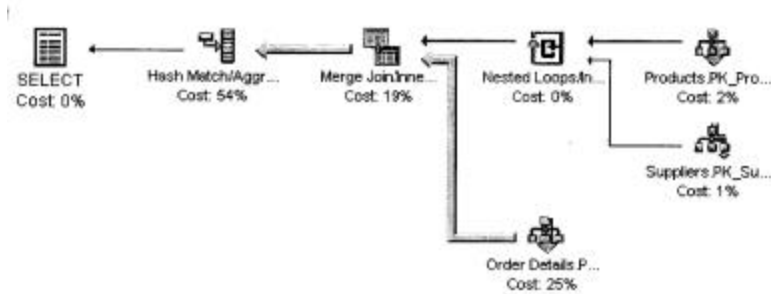
48: Your database includes a Suppliers table that contains 20 rows, a Products table that contains 80 rows, an Orders table that contains 50,000 rows, and an OrderDetails table that contains 150,000 rows. There are clustered indexes on the primary keys for each table, and the statistics are up to date.

You need to analyze the following query for performance optimization:

```
SELECT DISTINCT s.CompanyName FROM Suppliers s JOIN Products p ON  
  s. SupplierID = p.SupplierID JOIN OrderDetails od ON p.ProductID = od.ProductID
```

Which graphical execution plan might be generated by SQL Server Query Analyzer?

A:



49: Your database application includes a complex stored procedure that displays status information as it processes transactions. You obtain unexpected results in the status information when you run the stored procedure with certain input parameters. You want to use SQL Server Profiler to help find the problem in your stored procedure. Which four events should you track? (Choose four.)

A: *SQLTransaction*
SP:StmtCompleted
SQL:StmtStarting
Object:Opened

50: You have a table with a clustered primary key. New records are added by a batch job that runs at night. The table grows by 20 percent per year.

During the day, the table is used frequently for queries. Queries return ranges of rows based on the primary key. Response times for queries have become worse over time.

You run the DBCC SHOWCONTIG statement. The statement provides the following output:

- Pages Scanned: 354
- Extents Scanned: 49
- Extent Switches: 253
- Avg. Pages per Extent: 7.2
- Scan Density [Best Count:Actual Count].....: 17.79% [45:94]
- Extent Scan Fragmentation: 82.21%
- Avg. Bytes Free per Page: 485.2
- Avg. Page Density (full): 94.01%

What should you do to improve query performance?

A: *Rebuild the clustered index with a fill factor value set to 75.*

51: You are troubleshooting a process that makes use of multiple complex stored procedures that operate on a table. The process is producing an unexpected update to the table. You want to identify the specific stored procedure and statement that are causing the problem. What should you do?

A: *Use SQL Server Profiler to create and replay a trace by using single stepping.*

52: You have a decision support system (DSS) database that allows users to create and submit their own ad hoc queries against any of the DSS tables. Your users report that the response times for some queries are too long. Response times for other queries are acceptable. What should you use to identify long-running queries?

A: *SQL Server Profiler*

53: You are the database administrator for your company. You receive reports that your sales application has very poor response times.

The database includes a table that is defined as follows:

```
CREATE TABLE dbo.Orders (  
    OrderID          int      IDENTITY (1,1)      NOT NULL,  
    SalesPersonID    int              NOT NULL,  
    RegionID         int              NOT NULL,  
    OrderDate        datetime         NOT NULL,  
    OrderAmount      int              NOT NULL,  
    CustomerID       int              NULL)
```

The OrderID column is the primary key of the table. There are also indexes on the RegionID and OrderAmount columns.

You decide to run a showplan on all queries in the application. The following query, which accesses this table, is used to list total average sales by region:

```
SELECT t1.RegionID, AVG(t1.SalesTotal) AS RegionAverage  
FROM (SELECT RegionID, SalesPersonID, SUM( OrderAmount) AS SalesTotal  
FROM Orders  
GROUP BY RegionID, SalesPersonID) AS t1  
GROUP BY t1.RegionID
```

You set the SHOWPLAN TEXT option to ON and execute the query. The showplan output is as follows:

```
I--Compute Scalar(DEFINE: ([Expr1003]=If ([Expr1006]=0) then NULL else ([Expr1007]/[Expr1006] )))  
    I--Stream Aggregate (GROUP BY: ([Orders].[RegionID] )  
DEFINE: ([Expr1006]=COUNT([Expr1002]), [Expr1007]=SUM([Expr1002])))  
    I--Compute Scalar(DEFINE: ([Expr1002]=if ([Expr1004]=0) then NULL else  
[Expr1005]))  
        I--Stream Aggregate (GROUP BY: ([Orders].[RegionID],  
[Orders].[SalesPersonID]) DEFINE:([Expr1004]=Count(*),  
[Expr1005]=SUM([Orders].[OrderAmount] )))  
            I--Sort(ORDHR BY: ([Orders].[RegionID] ASC, [Orders].[SalesPersonID]  
ASC))  
                I--Table Scan(OBJECT:([ServerA].[dbo].[Orders])
```

You suspect that this query is part of the problem because the showplan indicates that the query is performing a table scan operation. What is the most likely reason that this query is performing a table scan?

A: There is no WHERE clause in the query.

54: You are building an invoicing system for your company. On each invoice, you want to show the following information:

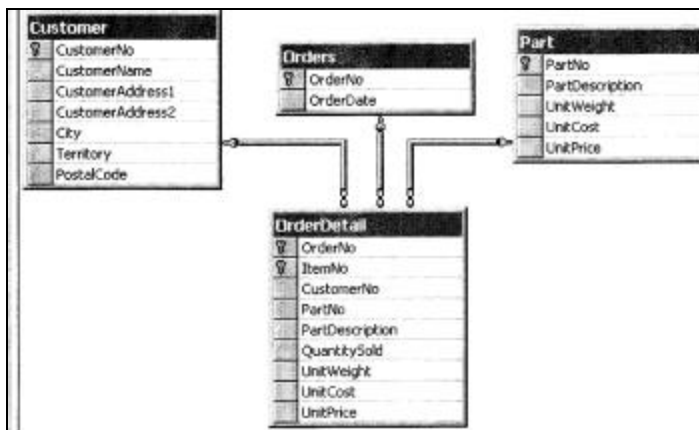
Customer Number
Customer Name
Customer Address
Customer City
Customer Territory
Customer Postal Code
Part Number
Part Description

Part Shipping Weight
 Current Cost
 Current Price
 Quantity On Hand
 Order Number
 Order Date
 Part Number
 Quantity

There can be multiple parts on one invoice. Cost and price information comes from a master list, but the history stored for each invoice should show the cost and price at time of sale. You want to make sure your database is properly normalized. You want to accomplish the following goals:

- Every table must have a primary key.
- All non-key columns must depend on the whole primary key.
- All columns must contain exactly one value.
- Each column in a table must be independent of any non-key column in the same table.

You create the logical model as shown in the exhibit. Which result or results does this model produce? (Choose all that apply.)



- A: Every table has a primary key.
 All columns contain exactly one value.

55: You are implementing a logical data model for a decision support system (DSS) database. Two of the tables in the model have a parent/child relationship. The parent table is expected to have more than 1 million rows. The child table is expected to have more than 100 million rows. Most reports present aggregate child information grouped according to each parent row. Reports containing detailed child table information are occasionally needed. How should the tables be implemented?

- A: Create the parent table that includes aggregate child information.
 Create the child table as it exists in the logical data model.

56: You are designing a database that will be used to store information about tasks assigned to various employees. Each task is assigned to only one employee. The database contains a table named Task that is modeled as shown in the exhibit. You want to use a PRIMARY KEY constraint to uniquely identify each row in the Task table. On which column or columns should you define the PRIMARY KEY constraint? (Choose all that apply.)

Task	
<input type="checkbox"/>	TaskNo
<input type="checkbox"/>	EmployeeNo
<input type="checkbox"/>	Title
<input type="checkbox"/>	Description
<input type="checkbox"/>	Status
<input type="checkbox"/>	DateCompleted

A: *TaskNo*
EmployeeNo

57: You are designing a data model that will record standardized student assessments for a school district. The school district wants the assessments to be completed online. The school district also wants each student's responses and scores to be stored immediately in the database.

Every year, each student will complete a behavior assessment and an academic assessment. The school district wants to prevent changes to assessment responses after the assessment is completed, but students should be allowed to change their responses during the course of the assessment. The school district wants to require each student to answer all items on each assessment. When a student indicates completion of the assessment, the score for the entire assessment must be computed and recorded.

You design a Student table and an Assessment table. You want to accomplish the following goals:

- Ensure that there is no redundant or derived data.
- Ensure that an assessment response cannot be changed after the assessment is completed and the score is entered.
- Ensure that all assessment items have responses when the assessment is completed and the score is entered.
- Ensure that an assessment score is computed and stored when the assessment is completed and the score is entered.

You take the following actions:

- Create a subtype table named BehaviorAssessment and a subtype table named AcademicAssessment, each with a primary key consisting of AssessmentID. Include a column in each table for the text of each assessment item.
- Create a StudentBehavior table with a foreign key referencing the Student table, a foreign key referencing the BehaviorAssessment table, and a primary key consisting of StudentID and AssessmentID. Include a nonnullable column in the StudentBehavior table for each assessment response, as well as an AssessmentScore column.
- Create a StudentAcademic table with a foreign key referencing the Student table, a foreign key referencing the AcademicAssessment table, and a primary key consisting of StudentID and AssessmentID. Include a nonnullable column in the StudentAcademic table for each assessment response, as well as an AssessmentScore column.
- Add an INSERT trigger on the StudentBehavior table computing a value for the AssessmentScore column when a row is inserted.
- Add an INSERT trigger on the StudentAcademic table computing a value for the AssessmentScore column when a row is inserted.
- Add an UPDATE trigger on the StudentBehavior table rejecting all updates to responses if the value in the AssessmentScore column is not null.
- Add an UPDATE trigger on the StudentAcademic table rejecting all updates to responses if the value in the AssessmentScore column is not null.

Which result or results do these actions produce? (Choose all that apply.)

A: There is no redundant or derived data.

An assessment response cannot be changed after the assessment is completed and the score is entered.

All assessment items have responses when the assessment is completed and the score is entered.

An assessment score is computed and stored when the assessment is completed and the score is entered.

58: You are building a database for the human resources department of your company. You want to eliminate duplicate entry and minimize data storage wherever possible. You want to track the following information for each employee:

First Name

Middle Name

Last Name

Employee Identification Number

Address

Date of Hire

Department

Salary

Name of Manager

Which table or tables should you use?

A:

Employee	
	EmployeeID
	ManagerEmployeeID
	FirstName
	MiddleName
	LastName
	Address
	DateOfHire
	Department
	Salary

59: You are the database developer for a leasing company. Your Leasing database includes a lessee table that is defined as follows:

```
CREATE TABLE lessee (  
  Id                int IDENTITY      NOT NULL  
  CONSTRAINT pk_lessee_id PRIMARY KEY NONCLUSTERED,  
  Surname           varchar(50)       NOT NULL,  
  FirstName         varchar(50)       NOT NULL,  
  SocialSecurityNo   char(9)          NOT NULL,  
  CreditRating      char(10)         NULL,  
  CreditLimit       money            NULL)
```

Each Social Security number must be unique. You want the data to be physically stored in order by Social Security number. Which constraint should you add to the SocialSecurityNo column on the lessee table?

A: a *UNIQUE CLUSTERED* constraint

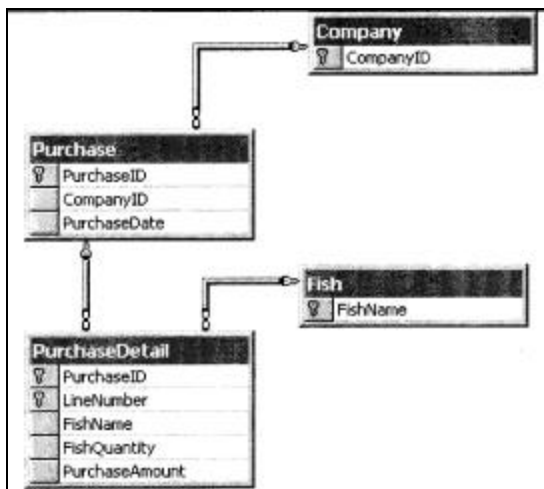
60: You are designing a data model to track purchases for a fish canning company. All transactions occur on fishing vessels that are not equipped to handle coins, and therefore all purchases must be

rounded to the nearest whole dollar amount. Purchases are made once a day, and fish must be purchased in multiples of 100 for tuna, and multiples of 50 for salmon. The government limits the purchase of salmon to 150 daily. Each month, your company must report to the government the kind, quantity, and supplier for each lot of tuna or salmon purchased. You want to accomplish the following goals:

- Ensure that all purchase amounts are rounded to the nearest whole dollar.
- Ensure that records for tuna purchased must be in multiples of 100, and that records for salmon purchased must be in multiples of 50.
- Ensure that the daily total quantity of salmon purchased does not exceed 150.
- Ensure that the required monthly government report can be produced reporting the kind, quantity, and supplier for each lot of tuna or salmon purchased.

You take the following actions:

- Create the data model as shown in the exhibit.



- Add a CHECK constraint to the PurchaseDetailtable on the FishQuantity column rejecting values not equal to 50 or 100.
- Add a trigger to the PurchaseDetail table rejecting values greater than 150 when the FishName is salmon.
- Add a trigger to the PurchaseDetail table rounding the PurchaseAmount to the nearest whole dollar amount.

Which result or results do these actions produce? (Choose all that apply.)

- A: All purchase amounts are rounded to the nearest whole dollar.
 The daily total quantity of salmon purchased cannot exceed 150.
 The monthly government report can be produced reporting the kind, quantity, and supplier for each lot
 of tuna or salmon purchased.

61: You are building a new database for the human resources department of a company. There are 10 departments within the company, and each department contains multiple employees. In addition, each employee might work for several departments. How should you logically model the relationship between the department entity and the employee entity?

A: Create a new entity. Create a one-to-many relationship from the employee entity to the new entity, and then create a one-to-many relationship from the department entity to the new entity.

62: You need to produce a sales report listing all salesperson identification numbers, sales amounts, and order dates. You want the report sorted from most recent sales to oldest sales for each day. You want the sales amounts sorted from highest to lowest.

You will be selecting this information from a table that is defined as follows:

```
CREATE TABLE SalesInformation
(SalesInformationID      int IDENTITY (1,1)      NOT NULL
                        PRIMARY KEY NONCLUSTERED,

SalesPersonID           int                      NOT NULL,
RegionID                int                      NOT NULL,
ReceiptID               int                      NOT NULL,
SalesAmount              money                   NOT NULL,
OrderDate               datetime                 NOT NULL)
```

Which query will accurately produce the report?

A: *SELECT SalesPersonID, SalesAmount, OrderDate
FROM SalesInformation
ORDER BY OrderDate DESC, SalesAmount DESC*

63: You have an Accounting application that captures batches of transactions into a staging table before being processed. Processing can be performed on individual batches or on the whole staging table. Processing includes many validations before updating any of the production tables.

You want to accomplish the following goals:

- Avoid deadlocks, or ensure that deadlocks are handled appropriately.
- Ensure that each batch of transactions is accepted or rejected.
- Allow other users to access the production tables while accounting transactions are being processed.
- Minimize resource locking.

You take the following actions:

- Begin a transaction.
- Validate each batch of accounting transactions.
- Insert new rows into production tables as appropriate by using INSERT...SELECT statements.
- Update and delete existing production tables as appropriate by joining production tables to the staging tables.
- Commit the transaction.
- Roll back the transaction if any errors are encountered:

Which result or results do these actions produce? (Choose all that apply.)

A: *Each batch of transactions is accepted or rejected.
Users can access the production tables while accounting transactions are being processed.
Resource locking is minimized.*

64: Your database stores telephone numbers. Each telephone number is stored as an integer. You must format the telephone number to print on a report in the following format:

(999) 999-9999

You have selected the phone number into a local variable as follows:

DECLARE @PhoneNumber int

Which statement will correctly format the number?

*A: SELECT 'PhoneNumber' = '(' + SUBSTRING(CONVERT(varchar(10),@PhoneNumber), 1,3) + ')'
' + SUBSTRING (CONVERT (varchar (10), @PhoneNumber) , 4,3)
+ '-' + SUBSTRING (CONVERT (varchar (10), @PhoneNumber) , 7, 4)*

65: You are developing a sales database for a company that has a 100-person sales staff. The company's policy requires that any sales orders in excess of \$100,000 be approved and entered into the database by the sales manager.

Your database includes a SalesOrder table that is defined as follows:

**CREATE TABLE SalesOrder (
Number char(10) NOT NULL,
SalesPerson varchar(50) NOT NULL,
Amount money NOT NULL)**

You need to create a view on the SalesOrder table that will prevent the sales staff from entering a sales order in excess of \$100,000. Which view should you write?

*A: CREATE VIEW SalesOrderLimit
AS SELECT Number, SalesPerson, Amount
FROM SalesOrder
WHERE Amount <= 100000
WITH CHECK OPTION*

66: Your database includes a salesperson table that tracks various data, including the sales goals and actual sales for individual salespeople. The sales manager wants a report containing a list of the five least productive salespeople, along with their goals and their actual sales production. You will use an ascending sort to order the information in the report by actual sales production. What should you do to produce this report?

A: Include a TOP 5 clause in the select list against the salesperson table.

**67: Your database includes a table named SalesInformation that tracks sales by region.
The table is defined as follows:**

**CREATE TABLE SalesInformation
(SalesInformationID int IDENTITY (1, 1) NOT NULL
PRIMARY KEY NONCLUSTERED,
SalesPersonID int NOT NULL,
RegionID int NOT NULL,
ReceiptID int NOT NULL,
SalesAmount money NOT NULL)**

Your database also includes a table named SalesPerson that is defined as follows:

```
CREATE TABLE SalesPerson
(SalesPersonID      int IDENTITY (1,1)      NOT NULL
PRIMARY KEY NONCLUSTERED,
RegionID           int                      NOT NULL,
LastName           varchar (30)      NOT NULL,
FirstName          varchar (30)      NULL,
MiddleName         varchar (30)      NULL,
AddressID          int                NULL)
```

You want to ensure that each salesperson enters sales only in the salesperson's own region. Which of the following actions can you perform to accomplish this task?

A: Create a trigger on the SalesInformation table that verifies that the region for the sale is the same as the region for the salesperson.

68: Your database includes a table that is defined as follows:

```
CREATE TABLE Orders (
Order ID           int IDENTITY (1,1)      NOT NULL,
SalesPersonID      int                    NOT NULL,
OrderDate          datetime                NOT NULL,
OrderAmount        int                    NOT NULL)
```

The sales manager wants to see a report that shows total sales by region as well as a grand total of sales. Which query can you use to create the report?

A: *SELECT SalesPersonID, RegionID, OrderAmount
FROM Orders
ORDER BY RegionID
COMPUTE SUM(OrderAmount) BY RegionID
COMPUTE SUM(OrderAmount)*

69: Your development team has just developed, tested, and deployed a new accounting application that includes many integrated modules. Users frequently encounter deadlocks whenever someone performs a function that integrates data from multiple modules. The development team never encountered deadlocks when unit testing the application. What can you do to minimize deadlocks?

A: *Ensure that all transactions modify tables in the same order.*

70: You issue an UPDATE statement and then run a SELECT query to verify that the updates were accurate. You find out that the UPDATE statement was executed properly. However, the next time you log on to the SQL Server computer, it appears that your UPDATE statement was not executed. What is the most likely cause of the problem?

A: *The IMPLICIT_TRANSACTIONS Option is set to ON.*

71: Your company has a headquarters sales office and two remote sales offices. Each sales office has a database containing an Orders table that is defined as follows:

```
CREATE TABLE Orders (
OrderID           int IDENTITY (1,1)      NOT NULL,
SalesPersonID      int                    NOT NULL,
OrderDate          datetime                NOT NULL,
OrderAmount        int                    NOT NULL)
```

You want a report in the following format that combines data for all orders from headquarters and the other two sales offices:

Office OrderID SalesPersonID OrderDate OrderAmount

The server in the headquarters office is named HQ, and the servers in the remote sales offices are named RS1 and RS2. Which query can you use to produce the report?

A: *SELECT Office = 'HQ', OrderID, SalesPersonID, OrderDate, OrderAmount FROM
HQ.Sales.dbo.Orders
UNION ALL
SELECT Office = 'RS2', OrderID, SalesPersonID, OrderDate, OrderAmount FROM
RS2.Sales.dbo.Orders
SELECT Office = 'RS1', OrderID, SalesPersonID, OrderDate, OrderAmount FROM
RS1.Sales.dbo.Orders*

72: Your company is increasing the price of all items in inventory. The owner has decided that the higher priced items should be increased by a larger percentage than the lower priced items. You must increase the prices of the items in inventory according to the following guidelines:

- 5 percent increase for items that are currently priced under \$50
- 10 percent increase for items that are currently priced between \$50 and \$100
- 15 percent increase for items that are currently priced over \$100

You want to increase the prices by using the fewest possible resources. Which statement group should you execute?

A: *UPDATE Inventory
SET Price =
CASE
WHEN Price < 50
THEN Price * 1.05
WHEN Price BETWEEN 50 AND 100
THEN Price * 1.10
ELSE Price * 1.15
END*

73: You have a database to keep track of sales information. Many of your queries calculate the total sales amount for a particular salesperson. You must write a nested procedure that will pass a parameter back to the calling procedure. The parameter must contain the total sales amount from the table that is defined as follows:

```
CREATE TABLE Sales Information
(SalesInformationID      int IDENTITY (1, 1)      NOT NULL
PRIMARY KEY NONCLUSTERED,
SalesPersonID           int                      NOT NULL,
RegionID                int                      NOT NULL,
ReceiptID               int                      NOT NULL,
```


77: Department managers in your company want to use Microsoft Excel pivot tables to analyze data from your SQL Server database. You need to extract data from tables in the database to an Excel spreadsheet so that managers can copy the spreadsheet and build pivot tables. The data in the database changes frequently, and you want to automate the process of updating the Excel spreadsheet.

You plan to use a SQL Server Agent scheduled job to automate the extraction of the data to the spreadsheet. What should the scheduled job execute?

A: a Data Transformation Services export package to populate the spreadsheet.

78: You have a database that contains information about publications for sale. You want to write a full-text search query that will search through all columns in one table enabled for full-text querying. The table includes a column named titles, a column named price, and a column named notes. The titles and notes columns are full-text enabled.

You want to find all publications that deal with French gourmet cooking. Which CONTAINS statement should you use?

A: WHERE CONTAINS (, ' "French gourmet"')*

79: Your database includes a table named sales. You monitor the disk I/O on your sales table, and you suspect that the table indexes are fragmented. The sales table has a clustered index named c_sales on the primary key and two nonclustered indexes named nc_sales1 and nc_sales2. You want to rebuild the indexes on the sales table by using a method that consumes the fewest resources. How should you rebuild the indexes?

A: dbcc dbreindex(sales)

80: You need to create a 6-GB online transaction processing (OLTP) database. Your SQL Server computer has two disk controllers, and each controller has four 6-GB hard disk drives. Each hard disk drive is configured as a separate NTFS partition. Microsoft Windows NT, the Windows NT swap file, and SQL Server are all installed on drive C. The remaining drives, which are labeled as drives D through J, are empty. How should you create the OLTP database?

A: Create the data portion of the database as six separate files on drives D through I, with one file on each

drive.

Create the transaction log as a single file on drive J.

81: You work for a telemarketing company. The Company's telemarketing application is a purchased application that allows minimal enhancements. The application dials a telephone number and displays the last name of the customer from the Customers table. The telemarketers want to see the first name of the customer as quickly as possible.

You write the following stored procedure:

```
CREATE PROCEDURE GetCustomerFirstName
  @LastName varchar (50)
AS
  SELECT LastName, FirstName FROM Customers WHERE LastName = @LastName
```

You create a nonclustered index on the LastName column. You set the SHOWPLAN_TEXT option to ON and execute the stored procedure. The output is as follows:

I--Bookmark Lookup (BOOKMARK: ([Bmk1000]), OBJECT: ([Bank].[dbo].[Customer]))

**I--Index Seek(OBJECT: ((Bank].[dbo].[Customer].[LastNameIDX1]),
SEEK: ([Customer].[LastName]= [@LastName]) ORDERED)**

What can you do to make the query more efficient?

A: Add FirstName to the nonclustered index on the Customers table.

82: Your users report that your database application takes an excessive amount of time to complete an operation. Over time, with the addition of new rows and changes to existing rows, the situation has worsened. You suspect that the tables are not optimally indexed. You plan to use the SQL Server Profiler Create Trace Wizard to find out the cause of the problem. What should you use the Create Trace Wizard to do? (Choose two.)

*A: Find the worst performing queries
Identify scans of large tables.*

83: You have a table with a clustered primary key. The table is used frequently for both queries and data modifications. As part of a review of data storage and disk utilization, you run the DBCC SHOWCONTIG statement. The statement provides the following output:

- Pages Scanned: 158
- Extents Scanned: 21
- Extent Switches.: 20
- Avg. Pages per Extent: 7.5
- Scan Density [Best Count:Actual Count]...: 95.24% [20:21]
- Extent Scan Fragmentation: 4.76%
- Avg. Bytes Free per Page: 408.4
-Avg. Page Density (full) : 94.95%

What does this output tell you about how the data is stored? (Choose all that apply.)

*A: The table is not externally fragmented
The number of extent switches is excessive.
The IAM page does not reflect the actual extent usage.*

84: In the last year, your users have inserted or updated over 200,000 rows in a table named Sales. The Sales table has a clustered primary key, Response times were good, but users report that response times have become worse when queries are run against the Sales table.

You run the DBCC SHOWCONTIG statement on the sales table and receive the following output:

- Pages Scanned: 1657
- Extents Scanned: 210
- Extent Switches: 1528
- Avg. Pages per Extent: 7.9
- Scan Density [Best Count:Actual Count]: 13.60% [208:1529]
- Logical Scan Fragmentation : 91.43%
- Extent Scan Fragmentation: 1.43%
- Avg. Bytes free per page: 2843.5
- Avg. Page density (full): 64.87%

What should you do to improve the response times for queries?

A: Run the DBCC DBREINDEX statement on the sales table.

85: You add new functionality to an existing database application. After the upgrade, users of the application report slower performance. The new functionality executes multiple stored procedures and dynamic SQL statements. You want to be able to identify specific queries that are encountering excessively long execution times. What should you do?

A: Create a SQL Server Profiler trace that uses the minimum execution time and application filters.

86: You have an accounting application that allows users to enter information into a table named staging. When data entry is complete, a batch job uses the rows in the staging table to Update a table named production. Each user's rows in the staging table are identified by the user's SQL Server process ID number in a column named spid. The code for the batch job that updates the production table is:

```
declare @count int begin tran
select @count = count(*) from production p join staging s on p.account = s.account
where s.spid = @@spid
update p set amount = s.amount
from production p join staging s on p.account = s.account
where s.spid = @@spid
if @@rowcount <> @count rollback tran else commit tran
```

You find out that there have been locking problems when two users run the batch job at the same time. What should you do to solve the locking problem?

A: Include the table hint WITH ROWLOCK, UPDLOCK when counting the rows in the production table.

87: You are implementing a logical data model. All of the tables in your logical data model are normalized to at least third normal form. There are no surrogate primary keys in any of the tables. Some table relationships involve up to eight levels of parent, child, grandchild, and so forth. In the model the primary key of each descendent table inherits the primary key of all ancestor tables.

You want to accomplish the following goals:

- Allow tables at any level in the hierarchy to be joined to any other table in the hierarchy.
- Ensure that tables are joined on a single column.
- Limit the index length of primary keys to 8 bytes or less.
- Ensure that key columns are always compared on a binary basis regardless of which options were selected during the installation of SQL Server.

You take the following actions:

- Implement the data model as is.
- Create indexes on all foreign keys.

Which result or results do these actions produce? (Choose all that apply.)

A: Tables are joined on a single column.

Key columns are always compared on a binary basis regardless of which options were selected during the installation of SQL Server.