CertifyExpress

70-229

Designing and Implementing Databases with MS SQL Server 2000 Enterprise Ed.

CertifyExpress

Exam 70-229

Designing and Implementing Databases with Microsoft SQL Server 2000 Enterprise Edition

Abstract

This Study Guide intends to provide you with information to prepare for the Microsoft Sql Server 2000 70-229 Exam.

Before you start

This study guide provides you with information on the many different aspects of "SQL Server 2000 Database Design". Before you proceed with this subject, please read through the study material for the following and make sure you are 100% comfortable with the SQL Server architecture:

- SQL Server 7 Admin 70-028
- SQL Server 7 Database Design 70-029

The reason you need to be familiar with SQL 7 is that the new version has many things similar to the version 7. Especially for the database design concepts, they are about the same.

Version 7 and Version 2000 have too many things in common. In fact, experience on 70-028 and 70-029 will be extremely helpful. This study guide will have its focus on the new features of SQL Server 2000.

Do NOT rely solely on this study notes for the exam. By all means read more than one book on the subject and make sure you understand the material well enough so that you could be ready for the questions. There is no quick way to succeed for this topic. Ideally you must work things out and gain experience before even trying to sign up for the exam.

Your Study Track for SQL Server 2000 DB Design

1, Make sure you are comfortable with the concept of Relational Database System. SQL Server is a relational database system. Also know about SQL Structured Query Language, which is the language you will use to control many of the database operations.

For tutorials on RDBMS concepts as well as the SQL language, please visit the following sites:

http://www.spnc.demon.co.uk/ora_sql/sqlmain.htm

http://w3.one.net/~jhoffman/sqltut.htm

http://freeskills.com

http://www.ourfriends.net/pages/rdbms_b.htm

2, Make sure you know the basic concept of the Client Server Computing Model. The client presents and manipulates data on the desktop computer, while the server runs at the back end to store and retrieve protected data. SQL Server mostly works in a multi tier set up, but not as the front end though.

To gain a better understanding of Client Server Model, please visit:

http://webopedia.internet.com/TERM/c/client_server_architecture.html

http://www.uth.tmc.edu/~atonnese/clinserv.html

http://www.faqs.org/faqs/client-server-faq/

3, As of this writing many of the SQL Server 2000 resources are still in BETA testing stage. Due to the lack of SQL 2000 material in the market, you may want to start with those of SQL 7 first. Version 7 and Version 2000 have too many things in common. This study guide will have its focus on the new features of version 2000 of SQL Server.

Information for Version 7 of SQL Server DB Design should have covered the following identical information:

- Developing a Logical Data Model
- Create and alter databases
- Create and alter database objects
- Alter database objects to support replication
- Troubleshoot failed object creation
- Retrieving and Modifying Data
- Programming Business Logic
- Manage data manipulation
- Tuning and Optimizing Data Access
- Designing a Database Security Plan
- Create and manage application roles

The scenario types of questions in the Transcender SQL 7 Practice tests (www.transcender.com) are extremely helpful. For SQL Server DB Design, pay attention to all the questions in the package. Know how to read the diagrams too.

Additional Suggested Readings from bookstore:

McDba Sql Server 7.0 Database Design Study Guide : (Exam 70-29) (Test Yourself (Berkeley, Calif.).)
by Inc. Syngress Media(Editor). Paperback (May 25, 1999)

McSe : Sql Server 7 Database Design (The Training Guide Series)
by David Besch, et al. Textbook Binding (May 1999)

MCSE Database Design on SQL Server 7 Exam Prep
by Christopher A. Leonard, et al. Paperback (November 1999)

MCSE SQL 7 Database Design and Administration Practice Tests Exam Cram
by Geoffrey Alexander, Joseph, Jr. Alexander. Paperback (October 27, 1999)

Version 7 and Version 2000 have too many things in common. Due to the lack of SQL 2000 material in the market, you may want to start with those of SQL 7 first.

4, Know the permissions and security measures in Windows NT and Win2K. Here is a list of recommended readings for this topic:

Microsoft Windows 2000 Security Handbook by Jeff Schmidt, et al (Paperback)

Configuring Windows 2000 Server Security
by Thomas W. Shinder(Contributor), et al. Paperback (November 1999)

NT 4 Network Security
by Matthew Strebe, et al. Paperback (March 15, 1999)

5, Learn the new features offered by SQL Server 2000. Please visit the official SQL Server site for a list of new features:

http://www.microsoft.com/sql/

The best way to learn is to actually play with SQL Server 2000. You may obtain an evaluation edition from

http://www.microsoft.com/sql/productinfo/evaluate.htm

You should also download the set of reference materials for SQL Server 2000 from the following location:

http://www.microsoft.com/sql/productinfo/evaluate.htm#Evaluation%20Materials

6, Know XML, as SQL Server 2000 has integrated into it the support of XML. Short for Extensible Markup Language, XML is a specification developed by the W3C. Being the pared-down version of SGML, it is designed especially to allow designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

The relational database engine of SQL Server 2000 can return data as XML documents. Also, we can use XML to insert, update, and delete values in the database.

To learn more about XML, visit the following sites:

http://webdevelopment.developersnetwork.com/xml/

http://www.w3.org/XML/

http://www.csclub.uwaterloo.ca/u/relander/XML/

General Design Concept

There are mainly two kinds of database systems: OLTP and DSS. With OLTP you want to make sure update is fast and table structure is optimized. You tend to normalize the tables. In a 1NF, no table has columns that define similar attributes and no column contains multiple values in a single row. In a 2NF, each column that is not part of the primary key should depend on all columns that make a composite Primary key and not only depend on a subset of the composite Primary key. In a 3NF, columns that are not covered by the Primary Key should not depend on each other. You may not want to go to 4NF and 5NF, as they may involve too complex relationships which are costly to process.

With DSS you need fast retrieval speed, and redundant data is unavoidable. We classify data as redundant when data is irrelevant to a specific project; data has multiple copies stored in the database; data is basically derived from existing data.

In any case, make sure you know how to read ER Diagram and know how to distinguish between one to one, one to many and many to many relationships. Also be sure to know what is the best column for creating primary key and foreign key.

For the SQL language, make sure you fully understand how SQL is used in TSQL context. The order of SQL command is:

Select > From > Where > Group > Having > Order by > Compute By

With the Where clause, aggregate functions cannot be used. With the Group clause, aliases cannot be used, and that all columns in the Select list must occur in the Group clause, although you can have additional columns. With the Having clause, all columns must be listed in the group by clause or used in aggregate functions. With the Order By clause, all columns in the Compute by clause must appear in the order by clause. With the Compute By clause, an ORDER BY clause must also be used - all columns presented in the compute by clause must be listed in order by, and that it cannot contain aliases. The main difference between COMPUTE and GROUP BY is that GROUP BY produces a single result set, while COMPUTE produces multiple result sets.

For complex query, Inner Joins uses a comparison operator to match rows from two columns based on the values in the common columns to return rows that match. In contrast, Outer Joins returns all rows from at least one of the tables or views specified in the From clause. With Left Outer Joins, you may return all rows from the left table regardless, and that the right table will return NULL where there is no match. Right Outer Join is the other way around. Full Outer Joins returns all rows from both tables, and that NULL values will fill the non-matched fields. With Cross Joins, all rows in the left table are matched with all rows in the right table to produce a Cartesian product which is most likely meaningless.

Sometimes you may prefer to use the UNION clause to combine the result sets from two or more SELECT statements into one result set. To make this work, the result sets must have the same number of columns with compatible data types.

When we talk about UNION, we should also know that SQL Server 2000 supports updateable UNION views over local or remote linked servers, known as distributed partitioned views. This allows federations of servers to cooperate to provide a single view of partitioned data and provide unlimited scalability. To define distributed partitioned views, you need to first create an image of the database on each of the participating servers, then define the member tables that are to be horizontally partitioned, then define each member server as linked server, and finally define the distributed partitioned views in all of the member databases.

New Data Types

SQL Server 2000 introduces the 64-bit integer (bigint), variant (sql_variant), and table data types. The bigint data type is an integer that supports data from -263 (-9,223,372,036,854,775,808) through 263-1 (9,223,372,036,854,775,807) with a storage size of 8 bytes. Note that functions will return bigint only if the parameter expression is a bigint data type, and that SQL Server will not automatically promote other integer data types to bigint. Sql_variant data type stores values of various supported data types except text, ntext, image, timestamp, and sql_ variant. It operates like the variant data type in Visual Basic in that it allows a single column, parameter, or variable to store data values of different data types. The new table data type is a local variable for temporarily storing a rowset. It can be used in place of temporary tables stored in the tempdb database. Keep in mind that table data type is managed in memory only, it provides performance benefits over other disks based alternatives.

To support the new bigint data type, we have two new built-in functions: BIG_COUNT and ROWCOUNT_ BIG. They are similar to the COUNT function and @@ROWCOUNT variable, except that they return an integer value of type bigint.

To working with a sql_variant data type, you should always cast the variant to the required base data element with the CAST operator:

```
SET @xyz_chr = CAST(@xyz_var AS VARCHAR(12))
```

A new function called SQL_VARIANT_PROPERTY is provided for returning the base data type and other information about a sql_variant instance:

```
SQL_VARIANT_PROPERTY(expression, property)
```

In the above example, the property parameter contains the name of the sql_variant property for which information is to be provided. This can take a value of BaseType, Precision, Scale, TotalBytes, Collation, and MaxLength.

One of the common use of the variant datatype is for internet application that collects information from a Web user. When someone submit a value via a web form,instead of saving the form value in the database as a character variable together with all the associated metadata information, you can simply save the value as a sql_variant. As long as you remember to cast the variable before persisting it to the database, it will work fine.

An extended property is a name and value pair with the value being a sql_variant that can contain max 7500 bytes of data. To implement, you may use the three new system stored procedures for adding, updating, and dropping extended properties. They are sp_ setextendedproperty, sp_dropextendedproperty, and sp_updateextendedproperty. You can use the user-defined function fn_listextendedproperty to retrieve the extended properties if needed.

The table data type can be used as an alternative to the temporary tables stored in the tempdb database. It is managed in memory, and so it is more efficient.

You define such a data type like this:

```
DECLARE @local_variable TABLE <table_definition>
```

Together with a user defined function that returns a table data type, you can declare an internal table variable and return that variable as the return value. We refer this as rowset functions can be used where table or view expressions are allowed in TSQL queries: table returned by a user-defined function can be referenced in the from clause. User-defined functions that return a table is an alternative to views, given the fact that views are limited to a single select statement, while user-defined functions can contain multiple statements for more powerful logic in views.

User-defined Functions and other new Functions

User-defined functions are similar to stored procedures. They accept zero or more input parameters, and can return a scalar data type such as int, decimal, varchar, or sql_variant as well as the table data type. However, you must specify the RETURNS value and to terminate with the RETURN statement.

To have a user-defined function returns a scalar data type, you need to define the return type and specify the value in the return statement:

```
CREATE FUNCTION dbo.MyProductsTable ()
RETURNS TABLE
AS
RETURN      SELECT TOP 5
            ProductName AS MyProducts, UnitPrice
            FROM Products
            ORDER BY Products.UnitPrice
```

When the return type is a local table variable, this allows you to use the local table variable for inserts and other operations prior to the return.

A special type of table exists that returns a user-defined function (in-line function). It has a return type of table without the table definition. It actually returns the resultset of a single select statement from which the table definition is derived.

One reason of using function over stored procedure is that it allows you to include the function in the select statement:

```
SELECT Products.*, Supp.*
FROM MyProductsTable() AS Products
INNER JOIN Supp ON Products.SuppID = Supp.SuppID
```

You need to understand the determinism of a function, as all functions are either deterministic or nondeterministic. Deterministic functions always return the same result for a given set of parameters, while nondeterministic functions do not.

Some of the new functions:

GetUTCDate:

- returns the date and time in UTC format
- value derived from the current local time and the time zone setting of the operating system
- helpful for multi-site applications

SCOPE_IDENTITY:

- for the @@IDENTITY global variable
- returns the last value inserted into an identity column within the same scope - a module of work contained in a stored procedure, trigger, function, or batch

IDENT_CURRENT:

- for the @@IDENTITY global variable
- returns the last identity value generated for a specified table in any session and any scope. You may use the CONTEXT_ INFO variable to associate up to

128 bytes of binary information with the current session or connection to be referenced in multiple batches, stored procedures, triggers, or user-defined functions

OPENDATASOURCE:

- provides ad hoc connection information as part of object name without using a linked server name
- used mainly for infrequently accessed remote data

Triggers

Trigger is a special class of stored procedures defined to execute automatically when an update, insert, or delete statement is issued against a table. The new trigger type supported is the INSTEAD OF trigger, which has the ability to be specified on views in place of the triggering action, and the AFTER trigger, which is now the default trigger executed after the statement that triggered it completes for table (and only table).

For an AFTER trigger, if the statement fails with an error, it will not execute. You may specify multiple AFTER triggers for each triggering action, and you may specify the order of trigger execution using a new system stored procedure called sp_settriggerorder. Note that all the other AFTER triggers fire in an undefined order, and that you have no way to exercise any control.

Unlike AFTER trigger which is for table only, INSTEAD OF triggers can be specified on both tables and views. However, you can only define one INSTEAD OF trigger for each triggering action. You may use INSTEAD OF triggers to perform before trigger logic. Additionally, INSTEAD OF triggers can be used to provide update capabilities to UNION views by dispersing data to the appropriate tables in the union view.

Integrity Constraints

There are many different types of integrity. Domain Integrity refers to the requirement that data values in each column must be valid. To enforce, use Primary and Foreign Keys, Default definitions and Check constraints. With Entity Integrity, each row in a given table must be unique in the entity. To enforce, you use primary key. With Referential Integrity, relationships between tables are maintained through Primary Key and Foreign Key. In SQL Server 2000, this referential integrity concept has been extended to support cascading delete and update operations.

With the new NO ACTION constraint, you can prevent data on a referenced table from being deleted or updated if corresponding records exist on the referring table. You can also prevent data on the referring table from being inserted or updated if no corresponding entry exists on the referenced table.

```
CONSTRAINT FK_myOrder_Details FOREIGN KEY
    (OrderID) REFERENCES dbo.myOrders(OrderID)
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
```

You may also allow update and delete operations to be cascaded from the referenced table to the referred table. For example, if you change a product code, it may make more sense to cascade this new code to all the related order detail records. You do this with:

```
ON UPDATE CASCADE
```

Performance

Generally, for OLTP, disk I/O is always the bottleneck. To optimize I/O performance, you should place tables and indexes across as many different physical disks as possible, and placing the transaction log on its own disk. Also, for tables that are likely to participate in joins, you should place them on different disks so that they can be scanned in parallel.

Know when to use clustered index and when to use non-clustered index. Since clustered index actually rearrange the actual order of the records, there can be only one per table. Generally, clustered index should be kept as narrow as possible to reduce disk space consumption and reduce I/O burden caused by data modification in the table. In contrast, Non-clustered indexes use the clustered index keys as pointers to data. You should create clustered index on the Primary key column and create non-clustered indexes on columns often used in WHERE clauses. Always use the column that has more unique values first in a composite index.

SQL Server Query Analyzer is a GUI tool tightly integrated with the Index Tuning Wizard and SQL Server Profiler that enables you to write queries, execute multiple queries simultaneously, view results, analyze the query plan, and receive assistance to improve the query performance. Database Maintenance Plan Wizard helps setting up the core maintenance tasks necessary to ensure your database performs well, that it is regularly backed up and is checked for inconsistencies.

New in SQL Server 2000 is indexed view. In SQL Server 2000, a view that has a unique clustered index is referred to as an indexed view. You may actually create a unique clustered index as well as non-clustered indexes on a view to improve data access performance on complex queries. However, keep in mind that not all queries will benefit from indexed views. Most importantly, if the indexed views are not used, there is no benefit.

You should consider using indexed view in schema that involves any subset of tables referenced in the query, any subset of the conditions in the query for that subset of tables, grouping columns or aggregate functions.

Decision support queries usually reference large numbers of rows and aggregate large amounts of information into concise aggregates. By using indexed view you will force the view's resultset to be stored in the database, leading to substantially better response times.

With the NOEXPAND option you can force the query optimizer to treat the view like an ordinary table with a clustered index. Or you can explicitly exclude indexed views from consideration by using the EXPAND VIEWS option with the query. The SCHEMABINDING option is used to prevent the base referenced tables from being changed without view adjustment.

For the best result, indexes on tables and indexed views should be designed concurrently to avoid redundant recommendations that incur high storage and maintenance overhead. Also, when designing the indexed view, follow these guidelines if possible:

- Design indexed views that can be used by multiple queries.
- Keep the index compact by using the fewest number of columns and bytes as possible.
- Estimate the size of the resulting indexed view in advance, as a large size may not provide any significant performance gains at all.
- Use multiple smaller indexed views when possible.

You may use Index Tuning Wizard for creating indexed view, as it will make recommendation for you. For maintenance, SQL Server automatically maintains indexed views similar to any other index. However, if the indexed view references several tables, updating any of them may require updating the indexed view, which is expensive relatively. You need to take this into account before considering the use of indexed view, if the view will reference multiple sources.

To define an index on a view, use the create index statement as follow:

```
CREATE UNIQUE CLUSTERED INDEX myindex ON OrderTotals (OrderID)
```

Nonclustered indexes are supported only on view with previously defined unique clustered index. Indexes defined on computed columns are allowed if the expression defined for the column only references columns from the table containing the computed column and is deterministic, meaning calculated values do not change subsequent invocations. Using indexes on computed columns is recommended when you are trying to create covering indexes. For views using the GROUP BY aggregation, the COUNT_ BIG(*) statement is mandatory. In addition, all grouping columns must appear in the view's select list.

Keep in mind that index cannot be created on a computed column if the column expression references nondeterministic functions. Also, clustered index cannot be created on a view if the view references nondeterministic functions.

Collations

With SQL Server 7.0, during install you must specify a default code page and sort order. All databases will then be locked into that particular code page and sort order. In SQL Server 2000, collation acts as a collection of the sort order for Unicode data types, the sort order for non-Unicode character data types, and the code page that is used to store non-Unicode character data types. You can specify collations at the database or column level, in addition to the default collation specified during installation. In addition to support collations at design time, you may specify collations for individual statements to create queries on tables supporting multilingual data specific to the language of the data.

Take a look at this example code fragment for column level collation:

```
CREATE TABLE myBody (
   myName    NVARCHAR(128) COLLATE SQL_Latin1_General_CP1_CI_AI,
   myDesc   NVARCHAR(256),
   myType   VARCHAR(8),
   myLang   VARCHAR(16),
)
```

Take a look at this example code fragment for database level collation:

```
CREATE DATABASE myBody
 COLLATE SQL_Latin1_General_CP1_CI_AI
```

XML Support

In terms of XML Data Retrieval, results of SELECT statements can be returned as XML document fragments through using the FOR XML clause:

```
FOR XML {mode} [,XMLDATA][, ELEMENTS][, BINARY Base64]
```

With RAW mode you transform each row in the SQL query into an XML element with the generic row identifier. With AUTO mode you can return query results in a simple, nested XML tree. Within AUTO mode you can specify the ELEMENTS option to have columns returned as sub-elements. The option BINARY Base64 allows you to have binary data to be returned in base64 encoding. The XMLDATA option you can specify that XML-Data schema information should also be returned together.

With EXPLICIT mode you can specify the shape of the XML tree to have the power of manually ensuring that the generated XML is well formed. The EXPLICIT mode query will produce a universal table containing all the information about the resulting XML tree, and the table will be vertically partitioned into groups that become XML elements in the result. The Tag column will store the tag number of the current element. The parent column will store the tag number of the parent. You use these metadata tags together to describe a parent and child hierarchy in the XML tree.

OpenXML is a rowset function used to expose data from the XML document as a relational rowset. You may use it as a table reference and allow yourself to use the data in XML documents for inserts or updates into database tables. However, to use the OpenXML function, you must first create an internal instance of an XML document with sp_xml_preparedocument. Once the internal document has been used, you should remove it using sp_xml_removedocument to free up the memory.

A+ Certification, MCSE Training, CCNA Certification, CISSP, Network+ Training at Testking