This Cram Sheet contains the distilled, key facts about the SQL Server 2000 Design exam. Review this information immediately before entering the test room, paying special attention to those areas in which you feel you need the most review. Remember, you can transfer any of these facts that you've crammed into your short-term memory onto a blank piece of paper before beginning the exam.

Data Modeling

1. A *primary key* is composed of one or more attributes (columns) that uniquely identify an instance of the entity (a row in the table). In a relational database, every entity (table) should have a primary key.
2. Primary keys must adhere to certain rules:
   * The value must be unique for each instance of the entity.
   * The value must not be null.
   * The value cannot change or become null during the life of the instance.
3. A foreign key is an attribute or composite attribute that completes the relationship between two entities. This key, then, relates one entity to its parent entity. Foreign keys can relate to a primary key or to an alternate key in another entity. If a foreign key points to a primary key, then the foreign key must be non-null. Otherwise, it can be null. Foreign keys are used to maintain data integrity (referential integrity).
4. *Normalization* is a refinement process that occurs after you have identified all of the data objects. Normalization creates a set of relational entities or tables that are free of duplicate data and that can be modified correctly and consistently.
5. An entity is in *first normal form (1NF)* if there are no repeating attributes or groups of attributes and if the columns present atomic information.
6. A relational table is in *second normal form (2NF)* if it is in 1NF *and* every non-key attribute is fully dependent on the primary key. Second normal form has no redundant data.
7. *Third normal form (3NF)*-the most common form-removes data that is not dependent on the primary key. An entity is in third normal form if it is in 2NF *and* if no non-key attributes are transitively dependent on their primary keys.
8. The *referential integrity* rule states that every foreign key value must match a primary key value in an associated table. Referential integrity ensures that data stays consistent between the two tables after you use an **INSERT**, **DELETE**, or **UPDATE** command.

Note: Although normalizing tables is effective for queries, to improve inserts, updates, and deletes (online transaction processing), you should, in some cases, denormalize and use fewer indexes on tables.

Database Objects

9. Common SQL Server data types include **char**, **int**, **varchar**, **datetime**, **bit**, and **text**.
10. The **int** data type is a 4-byte value that can store numbers between Ð2,147,483,648 and 2,147,483,648.
11. More data types are available; some are related to those just listed. For example, SQL Server 2000 has a new data type called **bigint**, which can store 8 bytes. The new **sql_variant** data type can contain **char**, **nchar**, **smallint**, and **float** values; however, it cannot contain larger data types such as **image** or **text**.
12. The latest possible date that can stored in the **datetime** data type is 12/31/9999, and the earliest date that can be stored is 1/1/1753.
13. If you do not want existing data to be confirmed for a **CHECK** or **FOREIGN KEY** constraint, you should use the **WITH NOCHECK** argument in the **ALTER TABLE** command.
14. SQL Server 2000 provides a few new system-supplied data types:
    * **bigint**-An 8-byte integer with a value range of plus or minus 2^63.
    * **sql_variant**-Stores the values of various SQL-Server-supported data types, except for **text**, **ntext**, **timestamp**, and **image**. The **sql_variant** data type is similar to the **variant** type in Visual Basic in that it can represent many data types in different situations.

Note: When Microsoft lists the exclusions for these data types, **sql_variant** itself is listed as an excluded data type.

    * **table**-Stores a result set for later processing. The **table** data type cannot be used in column definitions. It can be used only with local variables or return values.
15. You can inhibit **NULL** values in columns by using the **NOT NULL** constraint.
16. Because a **CHECK** constraint can refer to other columns in the table, use a **CHECK** constraint rather than a trigger whenever possible. A **CHECK** constraint occurs before the **INSERT** or **UPDATE** statement, whereas a trigger happens afterward.
17. The **PRIMARY KEY** constraint identifies each row with this designation as being unique. Remember:
    * Only one **PRIMARY KEY** constraint can be defined per table.
    * A **PRIMARY KEY** constraint can be applied to one or more columns. If a single column does not identify a record uniquely, then choose a set of columns and designate them, combined, as the primary key.
    * A unique index is created automatically. It is clustered by default.
    * The index created cannot be dropped. You would have to drop the **PRIMARY KEY** constraint.
18. SQL Server enforces **UNIQUE** constraints by creating nonclustered indexes that are unique. **UNIQUE** constraints cannot assign values.
19. The order of constraints is:
    a. The **UNIQUE** or the **PRIMARY KEY** constraint
    b. The **FOREIGN KEY** constraint
20. The **CASCADE** clause propagates a deletion to the referencing tables, rather than generating an error or requiring triggers or stored procedures to handle the propagation.
21. **CHECK** constraints are logical expressions that can prevent database users from entering invalid data into particular columns. All data in a table is validated with any **CHECK** constraints that you add.
22. By implementing the **Immediate-Updating Subscribers** option, you can use transactional replication to modify data on computers subscribing to that service.

23. When column values in a database are valid, this validity is called *domain integrity*.
24. The **IDENTITY** property can be used only once in a table.
25. The **IDENTITY** property data type must be **integer**, **numeric**, or **decimal**. The **numeric** and **decimal** data types must have a scale of zero.
26. A column with the **IDENTITY** property cannot be **NULL** or have a **DEFAULT** constraint.
27. A column with the **IDENTITY** property cannot be updated.
28. Uniqueness with the **IDENTITY** property is not guaranteed; use the **UNIQUE INDEX** constraint.
29. When a column in a table has the **IDENTITY** property, you can execute the **SET IDENTITY_INSERT ON** statement to enable particular values to be inserted into that column.
30. The **RULE** objects are used primarily for backward compatibility. **CHECK** constraints are now the standard for attribute integrity.
31. As with **CHECK** constraints, by implementing **DEFAULT** definitions, you can effectively maintain domain integrity.

Retrieve and Modify Data

32. SQL Server 2000, unlike its predecessors, has two limits for nondeterministic functions:
    * You cannot create an index on a computed column whose expression includes a nondeterministic function.
    * You cannot create a clustered index on a view if the view refers to any nondeterministic functions.

Implementing Indexes

33. A table can contain only one clustered index. Always try to use nonclustered indexes on columns where few records are returned.
34. Nonclustered indexes work best on columns that contain a range of values such as **datetime** values. Nonclustered indexes use the table's clustered index keys as pointers to the actual data.
35. You can use the **DROP_EXISTING** option with the **CREATE INDEX** statement to drop and re-create all indexes.
36. Generally, create an index only if the data to be stored in the corresponding indexed column(s) will often be queried or used to identify information stored in specific rows.
37. You *always* read the execution plan from right to left, top to bottom.

Views

38. If you create a view that joins data from two databases, make sure you have permissions granted for both of them.
39. If you want to be able to update data in a view without causing records to disappear, you can specify the **WITH CHECK OPTION** argument in the **CREATE VIEW** statement.
40. The rules for the partitioning column are as follows:
    * Each underlying base table must have a partitioning column defined with one (and only one) **CHECK** constraint so that any given value of the partitioning column maps to a single table. The **CHECK** constraints can use only these operators: **BETWEEN**, **AND**, **OR**, <, <=, >, >=, =.
    * A partitioning column cannot allow nulls, must be part of the primary key, and cannot be made up of computed columns.
    * A partitioning column must be in the same ordinal location in the select list of each **SELECT** statement in the view.
41. **INSERT**, **UPDATE**, and **DELETE** statements run faster when there are fewer indexes on a table.

Stored Procedures

42. When stored procedures refer to database objects owned by a user other than the owner of the stored procedure, **EXECUTE** permissions must be checked for all objects, including the stored procedure.
43. Objects within a stored procedure are resolved at runtime and therefore should be qualified within the stored procedure.
44. **SET QUOTED_IDENTIFIER** and **SET ANSI_NULLS** are determined at database creation time. All other settings are determined at database execution time.
45. Executing a stored procedure without required parameters that do not have defaults will generate an error.
46. You must include the **OUTPUT** keyword both in the **CREATE** statement and in the **EXECUTE** statement.
47. If objects referenced by a stored procedure change (such as when you're adding indexes or importing a large amount of data), you should force a recompile of the stored procedure.

Triggers

48. The sequence of events is: **INSTEAD OF** trigger, constraints, **AFTER** triggers. Any failure rolls back the modification.
49. **INSTEAD OF** triggers are not allowed on an updatable view with the **WITH CHECK OPTION** clause. You must alter the view to remove this clause if you want to use an **INSTEAD OF** trigger.
50. Cascading referential integrity constraints are new to SQL Server 2000. The **ON DELETE CASCADE** statement deletes child table records when their corresponding parent records are deleted, not the other way around. If you delete from the **Customers** table a record with a **CustomerID** of 99, all records with a **CustomerID** (foreign key) of 99 in the **Orders** table will also be deleted.

Transactions, Cursors, and Locks

51. SQL Server 2000 autocommits transactions by default. Each individual SQL statement is a separate transaction that is committed automatically upon completion.
52. You can use the **SET** command as follows to change a transaction's isolation level:

```
SET TRANSACTION ISOLATION LEVEL
{READ COMMITTED | READ UNCOMMITTED |
REPEATABLE READ | SERIALIZABLE}
```

53. The **READ UNCOMMITTED** isolation level (isolation level 0) is the least restrictive isolation level used by SQL Server. **READ UNCOMMITTED** uses dirty reads:

Your transaction can read another transaction's uncommitted changes. Data values could be changed, causing records to appear or disappear in the result set before the end of the transaction. This option has the same effect as setting **NOLOCK** on all tables in all **SELECT** statements in a transaction.

54. At the **READ COMMITTED** level, SQL Server issues shared locks while reading data and respects exclusive locks. This is the SQL Server default isolation level. You cannot read uncommitted data in this state, and any outstanding exclusive locks block you.

55. A transaction executing at the **REPEATABLE READ** isolation level holds all locks for the duration of the transaction. This isolation level guarantees that other transactions cannot change any records read in the transaction. **REPEATABLE READ** protects a range of data from changes by other transactions, but it doesn't prohibit new rows (phantoms) from being inserted into the protected result set.

56. Serializable isolation is the most restrictive isolation level because it provides a simulated single-user environment within a multi-user database. The **SERIALIZABLE** level holds all its locks for a transaction's duration, just as **REPEATABLE READ**, and it implements key-range locks to prohibit other transactions from inserting new records (phantoms) into that range. Although you can ensure better data consistency by using a more restrictive isolation level, it comes at the cost of concurrency because locks are held longer, increasing the possibility of blocking other operations.

57. You can use table-level locking hints to override a transaction isolation level.

58. The SQL Server syntax for declaring a cursor is as follows:

```
DECLARE MyCursorName CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
FOR SELECT * FROM MyTableName
[ FOR UPDATE [ OF myColumnName] ]
```

59. To avoid locking issues, keep your transactions as short as possible.

60. SQL Server stores information about all unresolved locks in the **Syslocksinfo** system table.

Remote Data Access and XML

61. The **FOR XML** clause has the following syntax:

```
SELECT statement
FOR XML { RAW | AUTO | EXPLICIT }
[ , XMLDATA ]
[ , ELEMENTS ]
[ , BINARY BASE64 ]
```

62. The **FOR XML** clause is *not* valid in:

    * Subselections, which include nested **SELECT** statements, **SELECT INTO** statements, and assignments
    * **COMPUTE BY** or **FOR BROWSE** clauses
    * **GROUP BY** clauses with aggregate functions
    * **SELECT** statements used in view definitions
    * User-defined functions that return a rowset
    * Stored procedures called within an **INSERT** statement

Tuning and Optimizing Data Access

63. *Physical Disk: Avg Disk Queue Length* - A high number on this counter means that disk requests are delayed because the disk cannot keep up with requests. If this number is consistently high, consider adjusting the disk configuration or reorganizing its data.

64. *Physical Disk: % Disk Time* - A number greater than 90 percent on any disk means that the disk is too busy; consider adjusting the disk configuration or reorganizing its data.

65. *Processor: % Processor Time* - A percentage of more than 90 is too high. Upgrade your processor, or terminate applications other than SQL Server to ensure better throughput.

66. **UPDATE STATISTICS** causes a shared table lock to be placed on the table from which statistical information is being retrieved. This will prevent **UPDATE**, **DELETE**, and **INSERT** statements from being executed. **SELECT** statements will execute normally.