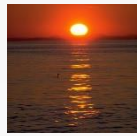
# Chapter 30: MicroSoft SQL Server

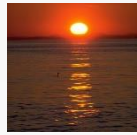
# Database System Concepts, 6th Ed.





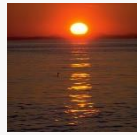
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



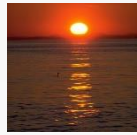
# History

- Microsoft SQL Server is a relational database-management system that scales from laptops and desktops to enterprise servers, with a compatible version
- SQL Server was originally developed in the 1980s at Sybase for UNIX systems
- Later it was ported to Windows NT systems by Microsoft
- Since 1994, Microsoft has shipped SQL Server releases developed independently of Sybase
- The latest available release is SQL Server 2005



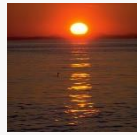
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# MS SQL Server: Design Tools [1/2]

- **Database Development and Visual Database Tools**
  - **SQL Server Management Studio**
    - Provides access to visual database tools
    - Provides three mechanisms to aid in database design
      - Database Designer
      - Table Designer
      - View Designer
  - Fig. 29.1 shows a view opened from the Management Studio



# MS SQL Server: Design Tools [2/2]

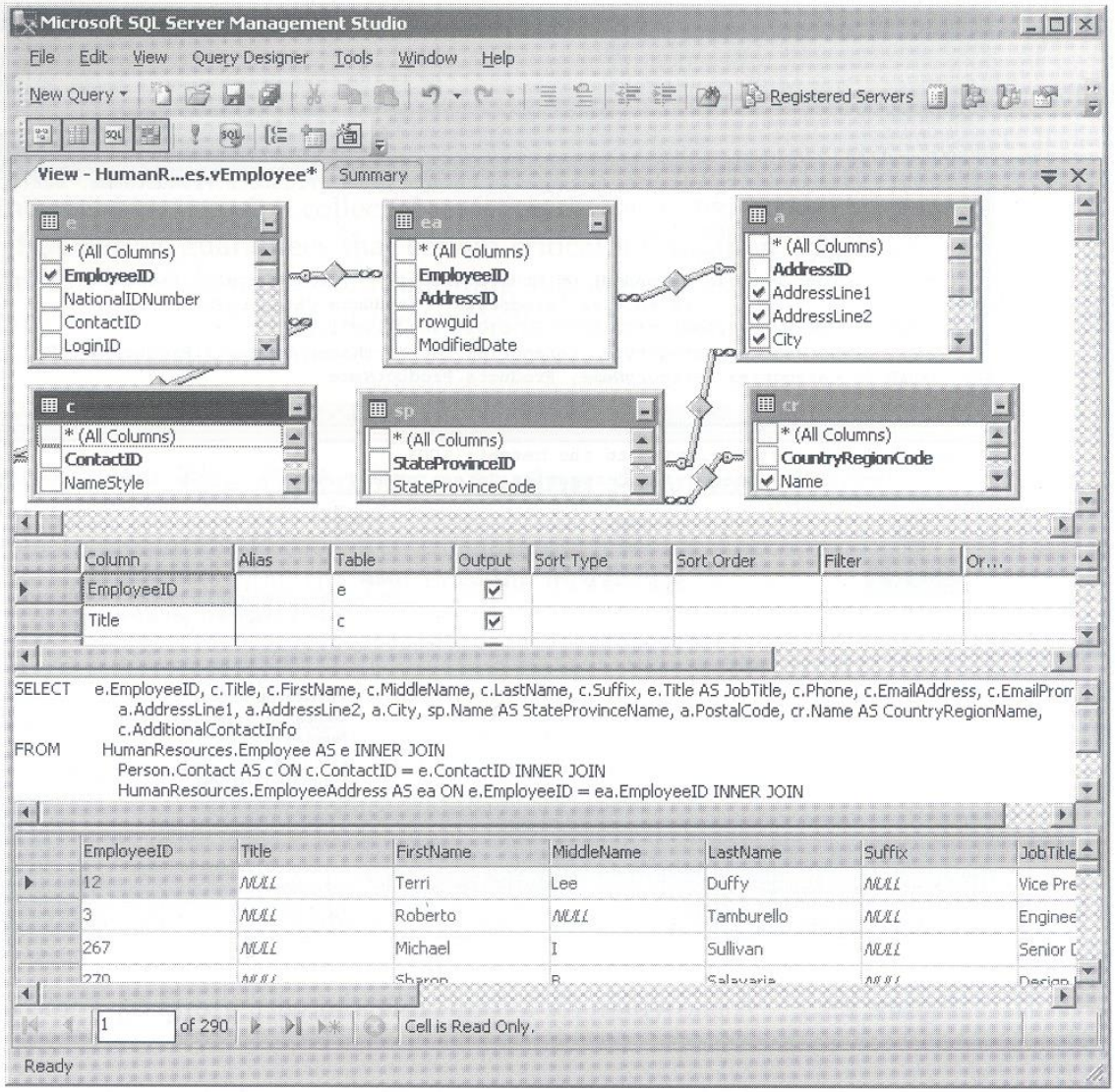
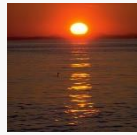



**Figure 29.1** The View Designer opened for the HumanResources.vEmployee view.

# MS SQL Server: Query Tool [1/2]

- **Query Editor**
  - Provides a simple graphical user interface for running SQL queries and viewing the results
  - Provides a graphical representation of **showplan**, the steps chosen by the optimizer for query execution
  - Is integrated with Management Studio Object Explorer
  - Can be used to
    - Analyze queries
    - Format SQL queries
    - Use templates for stored procedures, functions, and basic SQL statememts
  - Fig. 29.2 shows the Management Studio with the Query Editor displaying the graphical execution plan for a query



# MS SQL Server: Query Tool [2/2]

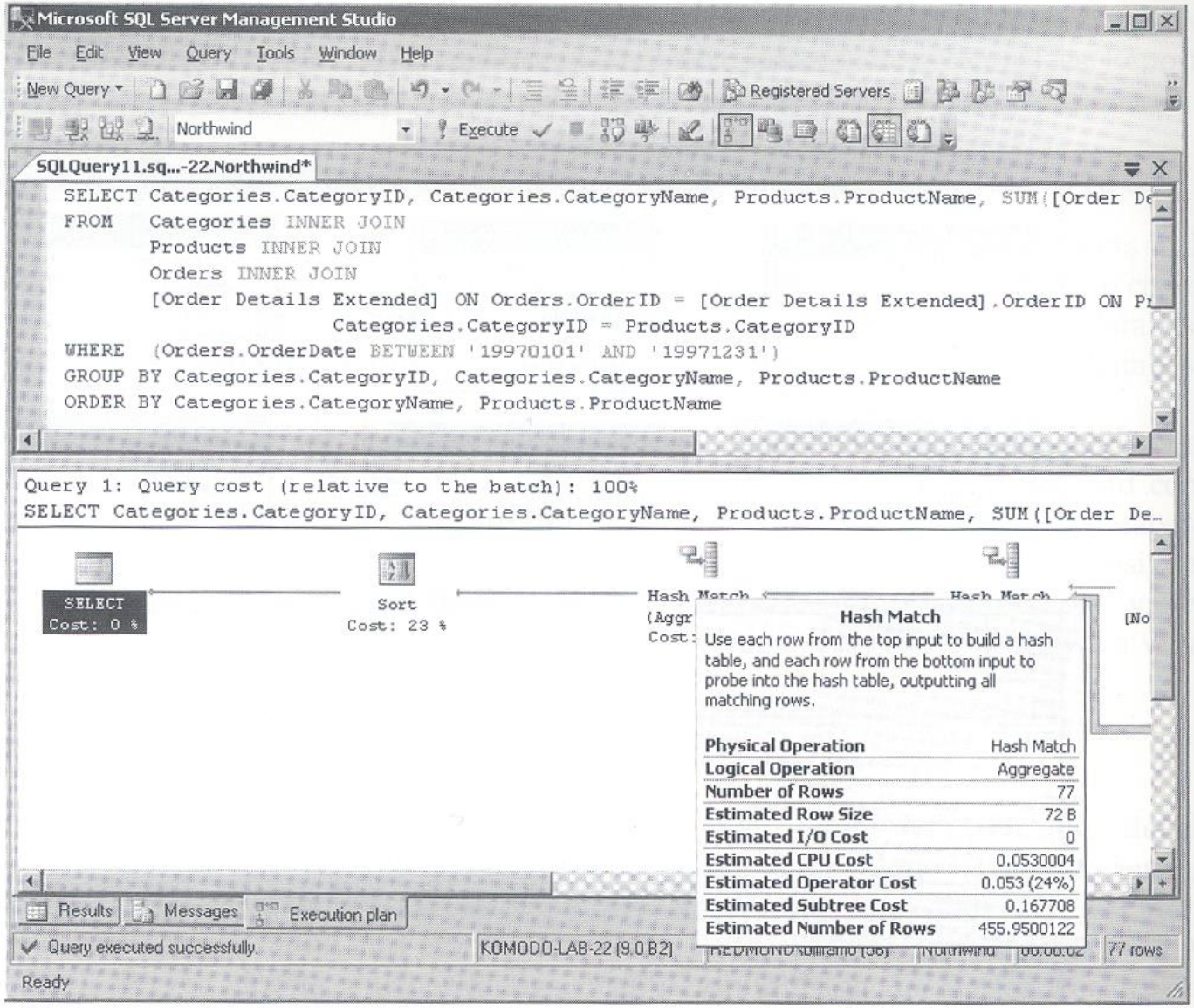
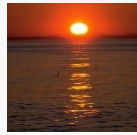



**Figure 29.2** A showplan for a four-table join with **group by** aggregation.

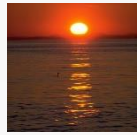
# MS SQL Server: Tunning Tools [1/2]

- **SQL Profiler**
  - A graphical utility to monitor and record database activity of the SQL Server Database Engine and Analysis Services
  - Can display all server activity in real time
  - Can display any SQL statement or stored procedure sent to any instance of SQL server as well as performance data
  - Allows drilling down even deeper into SQL Server to monitor every statement and operationClient-side trace facility
    - A user can choose to save the captured data to a file or table
    - Once trace data are saved, SQL Profiler can read the saved data for display or analysis purpose
  - Server-side trace facility
    - It manages queues of events generated by event procedures
    - A consumer thread reads events from the queues and filters them before sending them to the process that requested them



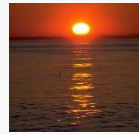
# MS SQL Server: Tunning Tools [2/2]

- **The Database Tuning Advisor (DTA)**
  - Is a powerful tool for designing the best possible indices and indexed views
    - based on observed query and update workloads
  - Can tune across multiple databases
  - Bases its recommendations on a workload that can be a file of captured trace events, a file of SQL statements, or an XML input file
  - Can look at the data access patterns for all users, for applications, for all tables, and make balance recommendations



# MS SQL Server: Management Tools [1/2]

- **SQL Server Management Studio**
  - Supports centralized management of all aspects of multiple installations of many tools
  - Allows a DBA to create, modify, and copy SQL Server database schemas and objects
  - Can manage hundreds of servers simultaneously
  - Provides wizards to guide the DBA through the process of setting up and maintaining an installation of SQL Server
  - Fig. 29.3 show the interface and how a script for database backup can be created directly from its dialogs



# MS SQL Server: Management Tools [2/2]

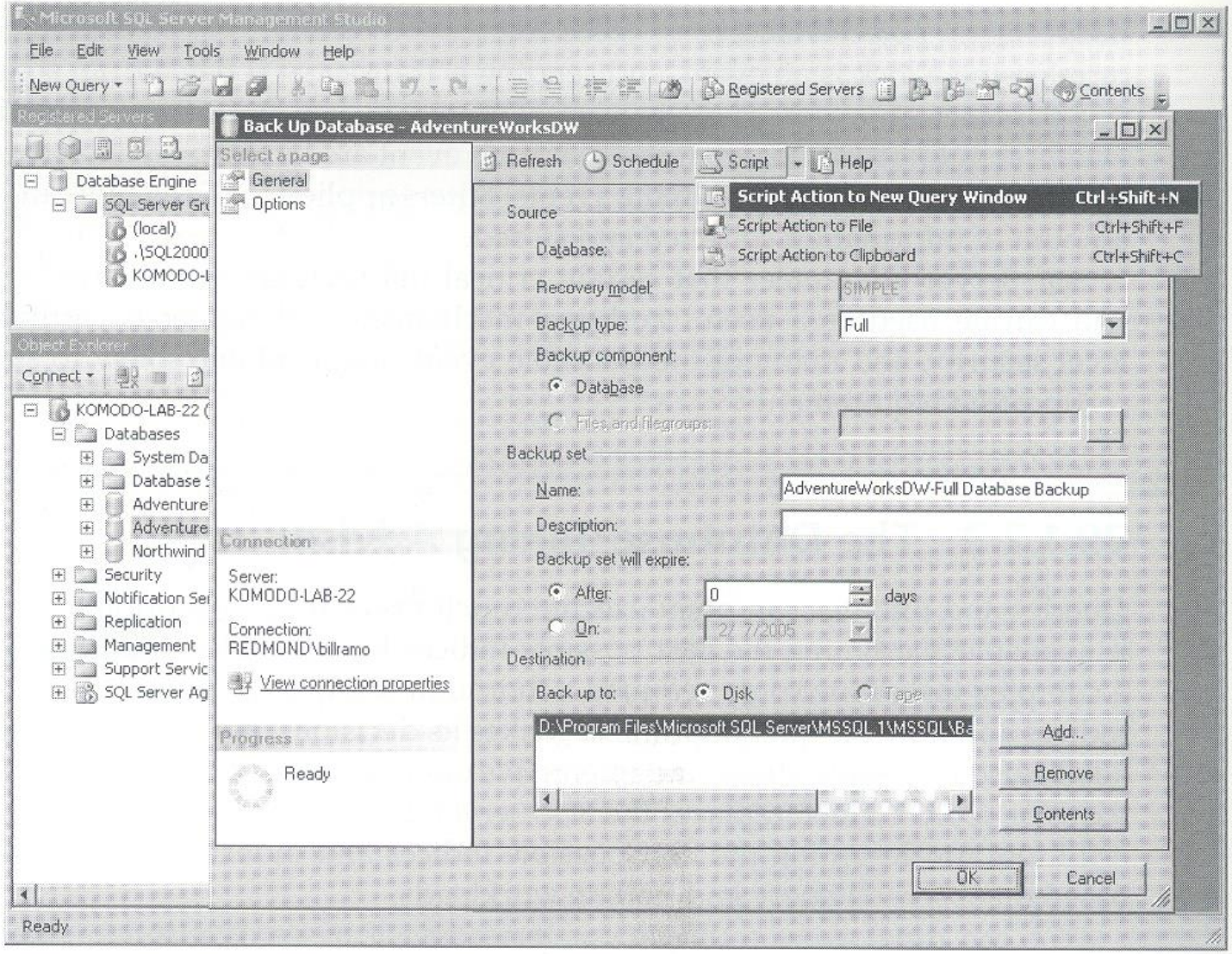
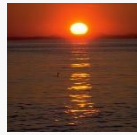



**Figure 29.3** The SQL Server Management Studio interface.

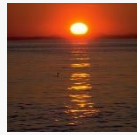
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



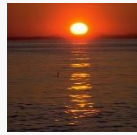
# MS Transact- SQL Variations and Extensions

- **MS Transact-SQL**
  - Database language supported by SQL Server
  - A complete database programming language including
    - Data definition and data manipulation statements
    - Iterative and conditional statements
    - Variables, procedures, and functions
  - Supports most of the **mandatory** DDL query and data modification statements and constructs in the SQL-2003 standard
  - Also supports many **optional** features in the SQL-2003 standard



# MS SQL Data Types

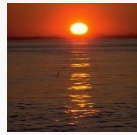
- **Data Types**
  - SQL Server supports all the mandatory scalar data types in the SQL-2003 standard except date and time
  - Primitive types unique to SQL Server
    - Large character and binary string types of variable size up to 2^31-1 bytes
      - **text/ntext/ image, varchar/nvarchar/varbinary(max)**
    - An XML type used to store XML data inside a table column
    - Sql-variant is a scalar data type that con contain values of any SQL scalar type
    - A **table** type enables a variable to hold a set of rows
    - A **cursor** type enables references to a cursor objects



# MS SQL Special Operators [1/2]

- SQL Server supports the special relational operators in addition to **inner join** and **outer join**
  - **Pivot**
    - An operator that transforms the shape of its input result set from two columns that represent name-value pairs into multiple columns, one for each name from input
    - The following query returns the *SalesQty* for each of the three months as separate columns

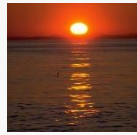
      **select ***
      **from** *MonthlySales* **pivot**( **sum**(*SalesQty*) **for** *month* **in** ('Jan', 'Feb', 'Mar')) *T*
    - The **pivot** operator also performs an implicit aggregation on all the other columns and an explicit aggregation on the pivot-column



# MS SQL Special Operators [2/2]

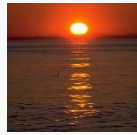
- **Apply**
  - A binary operator that takes two table-valued inputs
    - The right input is typically a table-valued function invocation that takes as arguments
    - One or more columns from the left input
  - Can be used to evaluate its right input for each row of its left input and perform **union all** of the rows
  - Two flavors
    - **cross apply**
    - **outer apply**
  - The following query call a function for the Manager of each department from the **Departments** table

    **select** *
    **from** *Departments D* **cross apply** *FindReports(D.ManagerID)*



# MS SQL Routines [1/3]

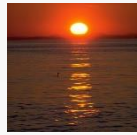
- **Routines**
  - Users can write routines that run inside the server process as scalar or table functions, stored procedures, and triggers using Transact-SQL or a .NET language
  - All routines are defined by using the corresponding **create [function, procedure, trigger]** DDL statement
  - **Indexed Views**
    - Indexed (materialized) views can substantially enhance the performance of complex decision support queries that retrieve larger numbers of base table rows
    - SQL Server supports creating a clustered index on a view and any number of nonclustered indices



# MS SQL Routines [2/3]

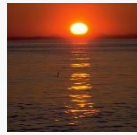
- **Routines (cont.)**
  - **Updatable Views and Triggers**
    - Views can be the target of **update, delete**, or **insert** statements
    - Updates to partitioned views can be propagated to multiple base tables

      **update** *titleview*
      **set** *price* = *price* * 1.10
      **where** *pub_id* ='0736'
    - Views can be updated by using the **instead** trigger
    - **Instead** triggers for **insert, update**, or **delete** operations can be defined on a view



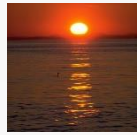
# MS SQL Routines [3/3]

- **Routines (cont.)**
  - **Updatable Views and Triggers (cont.)**
    - Triggers are Transact-SQL or .NET procedures that are automatically executed when either a DML (**update, insert** or **delete**) or DDL statement is issued against a base table or view
    - Two general kinds of triggers
      - **After** triggers
        - execute after the triggering statement and subsequent declarative constraints are enforced
        - can be defined only on base table
      - **Instead** triggers
        - execute instead of the triggering action
        - can be defined on base tables or views



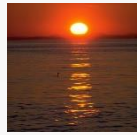
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



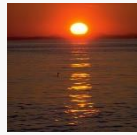
# MS-SQL Storage [1/3]

- **FileGroups**
  - In order to manage space effectively in a database, the set of data files in a database is divided into filegroups
  - Each filegroup contains one or more operating system files
  - Primary filegroup
    - Contains all the metadata for the database in system tables
    - May also store user data
  - User-defined filegroups
    - A user can explicitly control the placement of individual tables, indices, or the large-object columns of a table by placing them in a filegroup
    - Placing these tables and indices in different filegroups allows the user to control the use of hardware resources



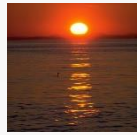
# MS-SQL Storage [2/3]

- **Space Management within Filegroups**
  - Main purpose
    - Effective space management
  - All data files are divided into fixed-size 8 KB units called **pages**
  - The allocation system is responsible for allocating these pages to tables and indices
  - Goal of allocation system
    - To minimize the amount of space wasted while keeping the amount of fragmentation in the database to a minimum to ensure good scan performance
  - In order to achieve this goal
    - Allocation manager usually allocates and deallocates all the pages in units of eight contiguous pages called **extents**
    - Allocation system manages these extents through various bitmaps



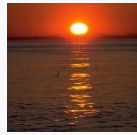
# MS-SQL Storage [3/3]

- **Tables**
  - SQL Server supports heap and clustered organizations for tables
  - In a heap-organized table
    - The location of row of table is determined entirely by the system
    - Rows of a heap have a fixed identifier known as the row (RID)
  - In a clustered index organization
    - Rows of the table are stored in a B+ -tree sorted by the clustering key of the index
    - The clustered index key also serves as the unique identifier for each row



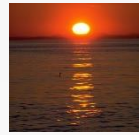
# MS-SQL Index

- **Indices**
  - SQL Server also supports secondary (nonclustered) B+-tree indices
  - Nonclustered indices over a table with a clustered index contain the key columns of the clustered index
  - SQL Server supports the addition of computed columns to a table
    - A computed column is a column whose value is an expression, usually based on the value of other columns in that row



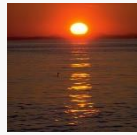
# MS-SQL Partitions

- **Partitions**
  - SQL Server supports range partitioning on tables and nonclustered indicies
  - A partitioned index is made up of multiple B+-trees, one per partition
  - Partitioning a large index
    - Allows an administrator more flexibility in managing the storage for the index
    - Can improve some query performance because the partitions act as a coarse-grained index



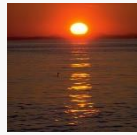
# MS-SQL On-line Index Build

- **On-line Index Build**
  - Building new indices and rebuilding existing indices on a table can be performed online
  - Three phases in creating a new index
    - Simply creating an empty B+-tree for the new index with the catalog
    - Scanning the table to retrieve the index columns for each row, sorting the rows and inserting them into the new B+-tree
    - Updating the catalog



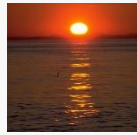
# MS-SQL Scans

- **Scans and Read-ahead**
  - Each of scan modes has a read-ahead mechanism, which tries to keep the scan ahead of the needs of the query execution in order to
    - Reduce seek and latency overheads
    - Utilize disk idle time
  - Read-ahead algorithm uses the knowledge from the query-execution plan in order to drive the read-ahead and make sure that only data that are actually needed by the query are read



# Chapter 30: Microsoft SQL Server

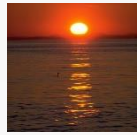
- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# MS-SQL Query Optimization Overview [1/2]

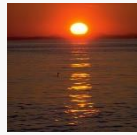
Query-compilation process consists of four steps

- **Parsing/binding**
  - The parser resolves table and column names using catalogs
  - SQL  Server utilize a plan cache to avoid repeated optimization
  - If no cached plan is available, an initial operator is generated
- **Simplification/normalization**
  - The optimizer applies simplification rules on the operator tree to obtain a normal, simplified form
  - The optimizer determines and loads statistics for cardinality estimation



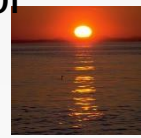
# MS-SQL Query Optimization Overview [2/2]

- **Overview of Optimization Process (cont.)**
  - **Cost-based optimization**
    - The optimizer
      - Applies exploration and implementation rules to generate alternatives
      - Estimates execution cost
      - Chooses the plan with cheapest anticipated cost
    - Implementation rules introduce execution alternatives such as merge join and hash join
  - **Plan preparation**
    - The optimizer creates query-execution structures for the selected plan



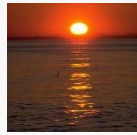
# MS-SQL Query Simplification [1/2]

- **Query Simplification**
  - A common scenario
    - Elimination of **union** branches that retrieve data from tables with different constraints
  - A number of simplification rules are *context dependent*
    - The substitution is valid only in the context of utilization of the subexpressions
    - Three examples
      - An outer join can be simplified into an inner join if a later filter operation will discard nonmatching rules that were padded with **null**
      - Elimination of joins on foreign keys need not be performed if there are no later uses of columns from the referenced table
      - The context of duplicate insensitivity specifies that delivering one or multiple copies of a row does not affect the query result



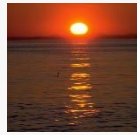
# MS-SQL Query Simplification [2/2]

- **Query Simplification (cont.)**
  - For grouping and aggregation, the *GbAgg* operator is used
    - It creates groups and optionally applies an aggregate function on each group
    - Duplicate removal is simply a *GbAgg* with no aggregate functions to compute
  - Subqueries are normalized by
    - Removing correlated query specifications
    - Using some join variant instead



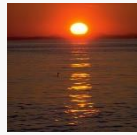
# MS-SQL Cost-based Optimization [1/2]

- **Reordering and Cost-Based Optimization**
  - Transformations are fully integrated into the cost-based generation and selection of execution plans
  - Correlated execution is considered during plan exploration, the simplest case being index-lookup join
    - SQL Server models correlated execution as **apply**, which operates on a table *T* and a parameterized relational expression *E(t)*
  - **Apply** executes E for each row of *T*, which provides parameter values



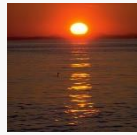
# MS-SQL Cost-based Optimization [2/2]

- **Reordering and Cost-Based Optimization (cont.)**
  - Materialized view utilization is also considered during cost-based optimization
    - View matching interacts with operator reordering in that utilization may not be apparent until some other reordering has taken place
    - When a view is found to match some subexpression, the table that contains the view result is added as an alternative for the corresponding expression



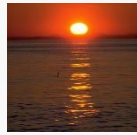
# MS-SQL Update Query Plans [1/2]

- **Update Plans**
  - Update plans optimize maintenance of indices, verify constraints, apply cascading actions, and maintain materialized views
    - For index maintenance
      - Update plans apply modification per index, sorting rows and applying the update operation in key order
      - This minimizes random I/O, especially when the number of rows to update is large



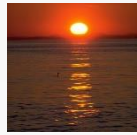
# MS-SQL Update Query Plans [2/2]

- **Update Plans (cont.)**
  - Halloween problem is addressed by using cost-based choices
    - Suppose a salary index is read in ascending order, and salaries are being raised by 10 percent.
    - As a result of the update, rows will move forward in the index and will be found and updated again, leading to an infinite loop
    - One way to address this problem is to separate processing into two phases
      - First read all rows that will be updated and make a copy of them in some temporary place
      - Then read from this place and apply all updates
    - Another alternative
      - Read from a different index where rows will not move as a result of the update



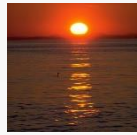
# MS-SQL Data Analysis for Query Processing

- **Data Analysis at Optimization Time**
  - Techniques to perform gathering of statistics as part of an ongoing optimization
  - The computation of result size is based on statistics for columns used in a given expression
  - These statistics consist of
    - Max-diff histograms on the column values
    - A number of counters that capture densities and row sizes
  - DBA may explicitly create statistics by using extended SQL syntax



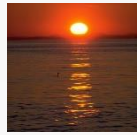
# MS-SQL Query Processing Heurostics

- **Partial Search and Heuristics**
  - SQL Server uses multiple optimization stages, each of which uses query transformations to explore successively larger regions of the search space
  - Each stage needs to balance opposing plan generation techniques
    - **Exhaustive generation of alternatives**
      - The optimizer uses complete, local, nonredundant transformations
    - **Heuristic generation of candidates**
      - Desirable transformations are incomplete, global, and redundant



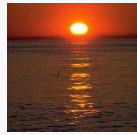
# MS-SQL Query Execution [1/3]

- **Query Execution**
  - Hash-based processing
    - Hash operations support basic aggregation and join, with a number of optimizations, extensions, and dynamic tuning for data skew
    - **Flow-distinct** operation
      - Is a variant of hash distinct, where rows are output early, as soon as a new distinct value is found
      - Is effective for queries that use **distinct** and request only a few rows



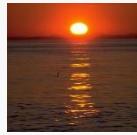
# MS-SQL Query Execution [2/3]

- **Query Execution (cont.)**
  - Hash-based processing (cont.)
    - *Asynchronous prefetching* allows issuing multiple index-lookup requests to the storage engine
      - A nonblocking index-lookup request is made for a row *t* of *T*, then *t* is placed in a prefetch queue
      - Rows are taken out of the queue and used by **apply** to execute *E(t)*
      - Execution of *E(t)* does not require that data be already in the buffer pool, but having outstanding prefetch operations maximizes H/W utilization and increase performance



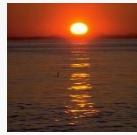
# MS-SQL Query Execution [3/3]

- **Query Execution (cont.)**
  - Index plans are made up of the pieces described earlier
  - For example, we consider
    - The use of index join to resolve predicate conjunctions (or index union, for disjunctions), in a cost-based way
    - Joining indices for the sole purpose of assembling a row with the set of columns needed on a query, which is sometimes faster than scanning a base table
  - Taking record ids from a secondary index and locating the corresponding row in a base table is effectively equivalent to performing index lookup join



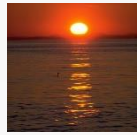
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# MS-SQL Transaction Model

- **Transactions**
  - All statements are atomic and application can specify various levels of isolation for each statement
  - Concurrency control based on locking is the default
  - SQL Server supports two snapshot-based isolation levels
    - **Snapshot**
      - Specifies that data read by any statement in a transaction will be the transactionally consistent version of the data that existed at the start of the transaction
    - **Read committed snapshot**
      - Specifies that each statement executed within a transaction sees a transactionally consistent snapshot of the data as it existed at the start of the statement



# MS-SQL Locking [1/2]

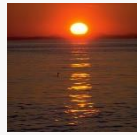
- Locking is the primary mechanism used to enforce the semantics of the isolation levels
- SQL Server provides multigranularity locking that allows different types of resources to be locked by a transaction
  - To minimize the cost of locking, SQL Server locks resources automatically at a granularity appropriate to the task
  - Locking at a smaller granularity, such as rows, increases concurrency, but has a higher overhead
- Fundamental lock modes
  - Shared (S)
  - Update (U)
  - Exclusive (X)
  - Intent locks (for multigranularity locking)
- Fig. 29.4 shows that resources are listed in order of increasing granularity

| *Resource* | *Description* |
|---|---|
| RID | Row identifier; used to lock a single row within a table |
| Key | Row lock within an index; protects key ranges in serializable transactions |
| Page | 8-kilobyte table or index page |
| Extent | Contiguous group of eight data pages or index pages |
| Table | Entire table, including all data and indices |
| DB | Database |

**Figure 29.4** Lockable resources.

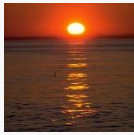
# MS-SQL Locking [2/2]

- **Dynamic Locking**
  - Fine-granularity locking
    - Can improves concurrency at cost of extra CPU cycles and memory to acquire and hold many locks
  - Coarser-granularity locking
    - Provides better performance with no (or minimal) loss of concurrency
  - Locking granularity is optimized automatically for optimal performance and concurrency for each index used in a query
  - The memory dedicated to the lock manager is adjusted dynamically on the basis of feedback from other parts of the system
- **Deadlock Detection**
  - SQL Server automatically detects deadlocks involving both locks and other resources
- **Row Versioning for Snapshot Isolation**
  - Two snapshot-based isolation levels use row versioning to achieve isolation for queries while not blocking the queries behind updates and vice versa



# MS-SQL Recovery and Availability [1/2]

- **Crash Recovery**
  - Logically, the log is a potentially infinite stream of log records identified by log sequence numbers (LSNs)
  - Physically, a portion of the stream is stored in log files
  - SQL Server dynamically adjusts the checkpoint frequency to reduce recovery time to within the recovery interval
  - Checkpoints
    - Flush all dirty pages from the buffer pool
    - Adjust to the capabilities of the I/O system and its current workload to effectively eliminate any impact on running transactions
  - Recovering multiple databases in parallel after a crash
    - The first phase: an analysis pass on the log, which builds a dirty page table and active transaction list
    - The next phase**:** a redo pass starting form the last checkpoint and redoing all operations
    - The final phase: an undo phase where incomplete transactions are rolled back

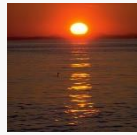


# MS-SQL Recovery and Availability [2/2]

- **Media Recovery**
  - SQL Server's backup and restore capabilities allow recovery from many failures
    - Loss or corruption of disk media, user errors, and permanent loss of a server
  - Backups can be taken on databases, files, filegroups, and the transaction log
  - All backups are fuzzy and completely online
  - Backup and restore operations are highly optimized and limited only by the speed of the media onto which the backup is targeted
- **Database Mirroring**
  - Database mirroring involves immediately reproducing every update to a database (principal database) onto a separate, complete copy of the database (mirror database)
  - The system automatically failover to the mirror in the event of a disaster or even just maintenance

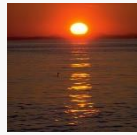
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



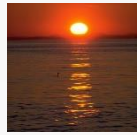
# MS-SQL Thread Pooling

- **Thread Pooling on the Server**
  - In order to minimize the context switching on the server, the SQL Server process maintains a pool of threads that execute client request
  - As requests arrive from the client, they are assigned a thread on which to execute
  - The thread executes the SQL statements and sends the results back to it
  - Thread pool is used to assign threads for internal background tasks
    - **Lazywriter**
    - **Checkpoint**
    - **Deadlock monitor**



# MS-SQL Memory Management [1/2]

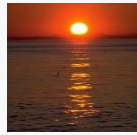
- **Memory Management**
  - **Buffer pool**
    - The biggest consumer
    - Maintains a cache of the most recently used database pages
  - **Dynamic memory allocation**
    - Is allocated dynamically to execute requests submitted by the user
  - **Plan and execution cache**
    - Stores the compiled plans for various queries that have been previously executed by users in the system
  - **Large memory grants**
    - For query operators that consume large amounts of memory



# MS-SQL Memory Management [2/2]

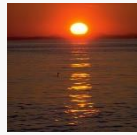
- **Memory Management (cont.)**
  - The memory manager is responsible for dynamically partitioning and redistributing the memory between the various consumers
  - It distributes the memory accordance with an analysis of the relative cost benefit of memory for any particular use
  - It interacts with OS to decide dynamically how much memory it should consume out of the total amount of memory in the system

- **Security**
  - SQL Server provides comprehensive security mechanisms and policies
  - Features that provide to help users secure the system properly
    - “Off-by-default”
    - “Best-practice analyzer”



# Chapter 30: Microsoft SQL Server

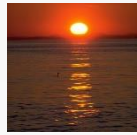
- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# MS-SQL Data Access

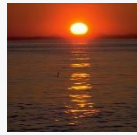
SQL Server supports APIs for building data-intensive applications

- **ODBC**
  - MS's implementation of the standard SQL:1999 call-level interface
- **OLE-DB**
  - A low-level, systems-oriented API designed for programmers building database components
- **ADO.NET**
  - A newer API for applications written in .NET languages
- **DB-Lib**
  - DB-Library for C API
- **HTTP/SOAP**
  - To invoke SQL Server queries and procedures



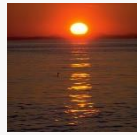
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# MS-SQL Distributed Heterogeneous Query Processing [1/3]

- SQL Server distributed heterogeneous query capability
  - Allows transactional queries
  - Updates against a variety of relational and nonrelational sources
- Two methods for referencing heterogeneous OLE-DB data sources in Transact-SQL statement
  - Linked-server-names method
  - Ad-hoc-connector-names method



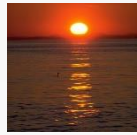
# MS-SQL Distributed Heterogeneous Query Processing [3/3]

- Linked-server-names method
  - Uses system-stored procedures to associate a server name with an OLE-DB data source
  - Objects in these linked servers can be referenced in Transact-SQL statements using the four-part name convention

    Ex) **select ***
    **from** *DeptSQLSrvr.Northwind.dbo.Employees*
  - Once a linked server is defined, it data can be accessed using four-part name, <linked-server>,<catalog>,<schema>,<object>

    Ex) **exec** sp_addlinkedserver OraSvr,'Oracle 7.3','MSDAORA','OracleServer‘
  - A query against this linked server is expressed as

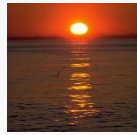
    Ex) **select ***
    **from** *OraSvr.CORP.ADMIN.SALES*



# MS-SQL Distributed Heterogeneous Query Processing [3/3]

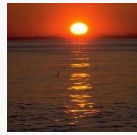
- Ad-hoc-connector-names method
  - Is used for infrequent references to a data source
  - Uses built-in, parameterized table-valued function called **openrowset** and **openquery**
  - The following query combines information stored in an Oracle server and a MS Index Server

```
Ex) select e.dept, f.DocAuthor, f.FileName
    from OraSvr.Corp.Admin.Employee e,
    openquery(EmpFiles,
                ' select DocAuthor, FileName
                from scope("c:\EmpDocs")
                where contains(' "Data" near() "Access" ')>0') as f
    where e.name = f.DocAuthor
    order by e.dept, f.DocAuthor
```



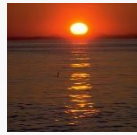
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# MS-SQL Replication

- **Replication Model**
  - SQL Server
    - Introduced the *Publish-Subscribe* metaphor to database replication
    - Extends this publishing industry metaphor throughout its replication administration and monitoring tools
  - **Publisher**
    - A server that makes data available for replication to other servers
  - **Subscribers**
    - Servers that receive replicated data from a publisher
  - **Distributor**
    - A server that plays different roles, depending on the replication options chosen

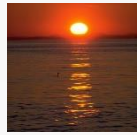


# MS-SQL Replication Options

- **Snapshot replication**
  - Copies and distributes data and database objects exactly as they appear at a moment in time
  - Is best suited for smaller sizes of data and when updates typically affect enough of the data that replicating a complete refresh of the data is efficient
- **Transactional replication**
  - The publisher propagates an initial snapshot of data to subscribers, then forwards incremental data modifications to subscribers
  - **Log reader agent** reads the transactions from the database transaction log, applies an optional filter, and stores them in the distribution database
  - **Distribution agent** forwards the changes to each subscriber
- **Merger replication**
  - Allows each replica in the enterprise to work with total autonomy whether online or offline
  - The system tracks metadata on the changes to published objects at publishers and subscribers in each replicated database
  - The replication agent merges those data modifications together during synchronization

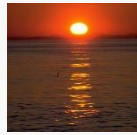
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



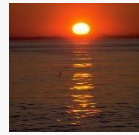
# MS-SQL Server Programming in .NET

- **Basic .NET Concepts**
  - In the .NET framework, a programmer writes program code in a high-level programming language that implements a class defining its structure and methods
  - The compilation of the program produces
    - A file, called *Assembly* containing the compiled code in *Microsoft Intermediate Language (MSIL)*
    - *Manifest* containing all references to dependent assemblies
  - The .NET framework supports a out-of-band mechanism called *custom attributes*
  - *Managed code* refers to MSIL executed in the CLR



# MS-SQL CLR Hosting [1/2]

- **SQL CLR Hosting**
  - .NET Common Language Runtime (CLR) is a run-time environment with a strongly typed intermediate language that executes multiple modern programming languages
    - Such as C#, Visual Basic, C++, COBOL, and J++
  - SQL Server and CLR are two different runtimes with different internal models
    - SQL Server supports a cooperative non-preemptive threading model
    - CLR supports a preemptive threading model



# MS-SQL CLR Hosting [2/2]

- **SQL CLR Hosting (cont.)**
  - Fig. 29.5 shows that CLR calls low-level primitives implemented by SQL Server

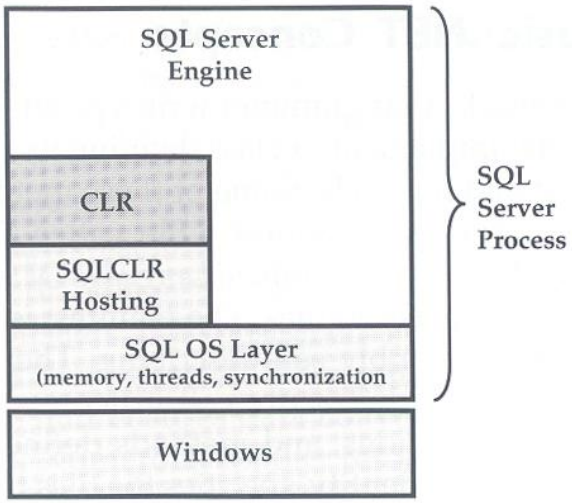
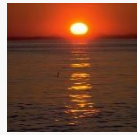



**Figure 29.5** Integration of CLR with SQL Server operating system services.

  - This approach provides the following scalability and reliability benefits
    - **Common threading, scheduling, and synchronization**
    - **Common memory management**

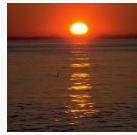
# MS-SQL Extensibility Contracts [1/2]

- **Routines**
  - We classify scalar functions, procedures, and triggers generically as routines
  - Routines, implemented as static class method, can specify the following properties
    - **IsPrecise**
    - **UserDataAccess**
    - **SystemDataAccess**
    - **IsDeterministic**
    - **IsSystemVerified**
    - **HasExternalAccess**



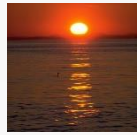
# MS-SQL Extensibility Contracts [2/2]

- **Table Functions**
- **Types**
  - Classes implementing user-defined types are annotated with a *SqlUserDefinedType*() attributes
    - **Format**
    - **MaxByteSize**
    - **IsFixedLength**
    - **IsByteOrdered**
    - **Nullability**
    - **Type conversions**
- **Aggreagates**
  - The query optimizer uses the following properties to derive alternative plans for the aggregate computation
    - **IsInvariantToDuplicates**
    - **IsInvariantToNulls**
    - **IsInvariantToOrder**



# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



# XML Support in SQL Server 2005

- Fig. 29.6 provides a high-level architectural diagram of SQL Server's native XML support in the database

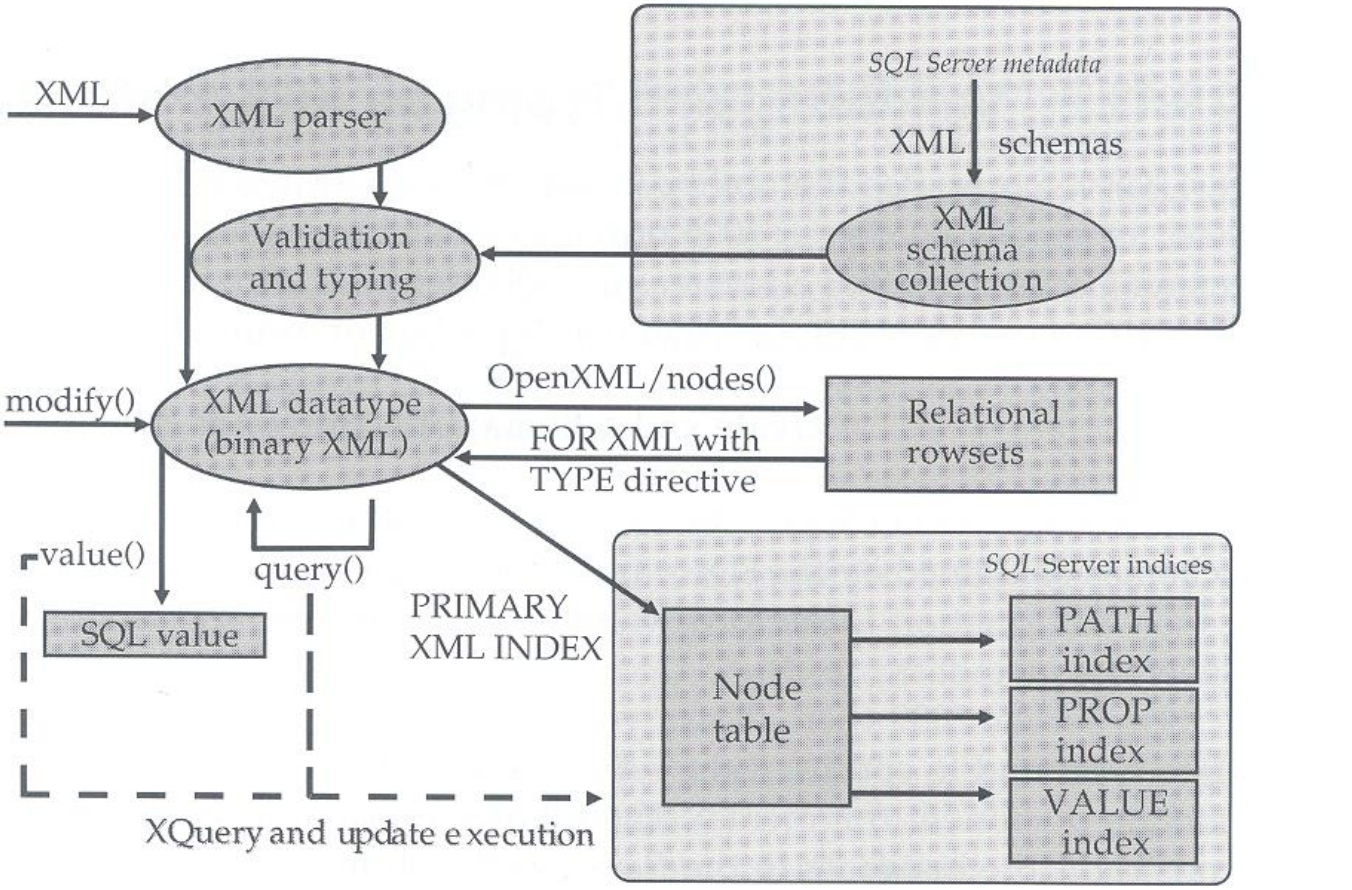
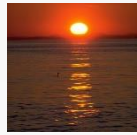



**Figure 29.6** Architectural overview of the native XML support in SQL Server.

# XML Storage in SQL Server 2005

- **Natively Storing and Organizing XML**
  - **Xml** data type can store XML documents and content fragments
  - The following SQL statement creates a table where one of the columns is of type XML

    Ex) **create table** *TripReports*(*id* **int**,
    *tripdate* **datetime**,
    *Report* **xml**)
  - A database system can choose to store an instance of type **xml** in any of a variety of formats, such as binary large object (**blob**)

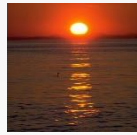


# XML Data Typing in SQL Server 2005

- **Validating and Typing and XML Data Type**
  - SQL Server 2005 provides a database metadata concept called an XML schema collection
  - The following expression creates a SQL Server schema collection with name S1

    Ex) **create xml schema collection** *S1* **as** *@s*
  - The following example shows a table definition that constraints the instances in the *Report* XML column to a well-formed document

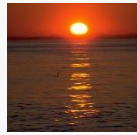
    Ex) **create table** *TripReport*(*id* **int**,
    *tripdate* **datetime**,
    *Report* **xml**(**document** *ReportSchema*))



# XML XQuery in SQL Server 2005 [1/5]

- **Querying and Updating the XML Data Type**
  - SQL Server 2005 provides XQuery-based query and modification capabilities on the XML data type
  - The following example shows a simple XQuery expression that summarizes a complex Customer element in a trip report document

Ex) **select** *Report.query*('
declare namespace c="urn:example/customer";
for $cust in /c:doc/c:customer
where $cust/c:notes//c:saleslead
return
<customer_id="$cust/@id">{
$cust/c:name,
$cust/c:notes//c:saleslead
}</customer>')
**from** *TripReports*



# XML XQuery in SQL Server 2005 [2/5]

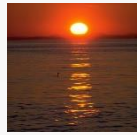
## Querying and Updating the XML Data Type

The following example shows a simple XQuery expression that counts the sales-lead elements in each XML data type instance and returns it as a SQL integer value

Ex) **select** *Report.value*(
'declare namespace c="urn:example/customer";
count(/c:doc/c:customer//c:saleslead)','int')
**from** *TripReports*

- The following expression retrieves every row of the *TripReports* table, where the document contains at least one customer with a sales lead

Ex) **select** *Report*
**from** *TripReports*
**where** 1=*Report.exist*('declare namespace c="urn:example/customer";
/c:doc/c:customer//c:saleslead')

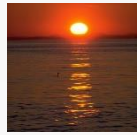


# XML XQuery in SQL Server 2005 [3/5]

## Querying and Updating the XML Data Type (cont.)

- The following example extracts for every customer order in the XML column a row that contains the XML representation of its customer, the order id, and the id of the document that contains the customer

```
Ex) select N1.customer.query('.') as Customer,
        N1.customer.value(
            'declare namespace c="urn:example/customer";
            c:name[1]','nvarchar(20)') as CustomerName,
        N2."order".value('@id','int') as OrderID,
        N1.customer.value('../@id','nvarchar(5)') as DocID
    from TripReports cross apply
        TripReports.Report.nodes(
            'declare namespace c="urn:example/customer";
            /c:doc/c:customer') as N1(customer)
        cross apply N1.customer.nodes(
            'declare namespace c="urn:example/customer";
            ./c:order') as N2("order")
```



# XML XQuery in SQL Server 2005 [4/5]

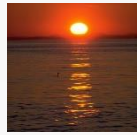
- **Querying and Updating the XML Data Type (cont.)**
  - The following example deletes all customer saleslead elements of years previous to the year given by an SQL variable or parameter with the name @year

Ex) **update** *TripReports*

**set** *Report.modify*(

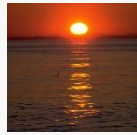
'declare namespace c="urn:example/customer";

delete /c:doc/c:customer//c:saleslead[@year<sql:variable("@year")]')



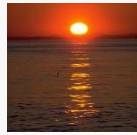
# XML XQuery in SQL Server 2005 [5/5]

- **Extension of XQuery Expressions**
  - In order to execute XQuery expressions, the XML data type is internally transformed into a so-called node table
  - The internal node table basically uses a row to represent a node
  - All XQuery and update expressions are then translated into an algebraic operator tree
  - SQL Server 2005 provides three secondary XML indices so that query execution can take further advantage of index structures
    - *Path* index
    - *Properties* index
    - *Value* index



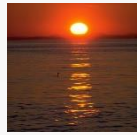
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



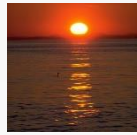
# SQL Server Service Broker

- Service Broker helps developers create loosely coupled distributed applications by providing support for queued, reliable messaging in SQL Server
- The basic unit of communication is the *conversation*
- A persistent, reliable, full-duplex stream of messages
- Each conversation is part of a *conversation group*
- A *service* is a named endpoint for a conversation
- A *contract* defines the message types that are allowable for a conversation



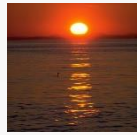
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



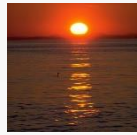
# MS-SQL DW and BI

- The data warehouse and business intelligence (DW & BI) component of SQL Server contains three components
  - SQL Server Integration Services (SSIS)
  - SQL Server Analysis Services (SSAS)
  - SQL Server Reporting Services (SSRS)
- The different components of the analysis server can integrate and leverage each other's capability



# MS- SQL Server Integration Services (SSIS)

- MS SQL Server 2005 Integration Services (SSIS) is an enterprise data transformation and data integration solution
  - You can use SSIS to perform the following tasks
    - Merge data from heterogeneous data stores
    - Refresh data in data warehouse and data marts
    - Cleanse data before loading it into destinations
    - Bulk-load data into OLTP and OLAP databases
    - Send notifications
    - Build business intelligence into a data transformation process
    - Automate administrative functions
  - Fig. 29.7 shows an example of how various transformations can be combined to cleanse and load book sales information



# Example of Enterprise Data Transformation in SSIS

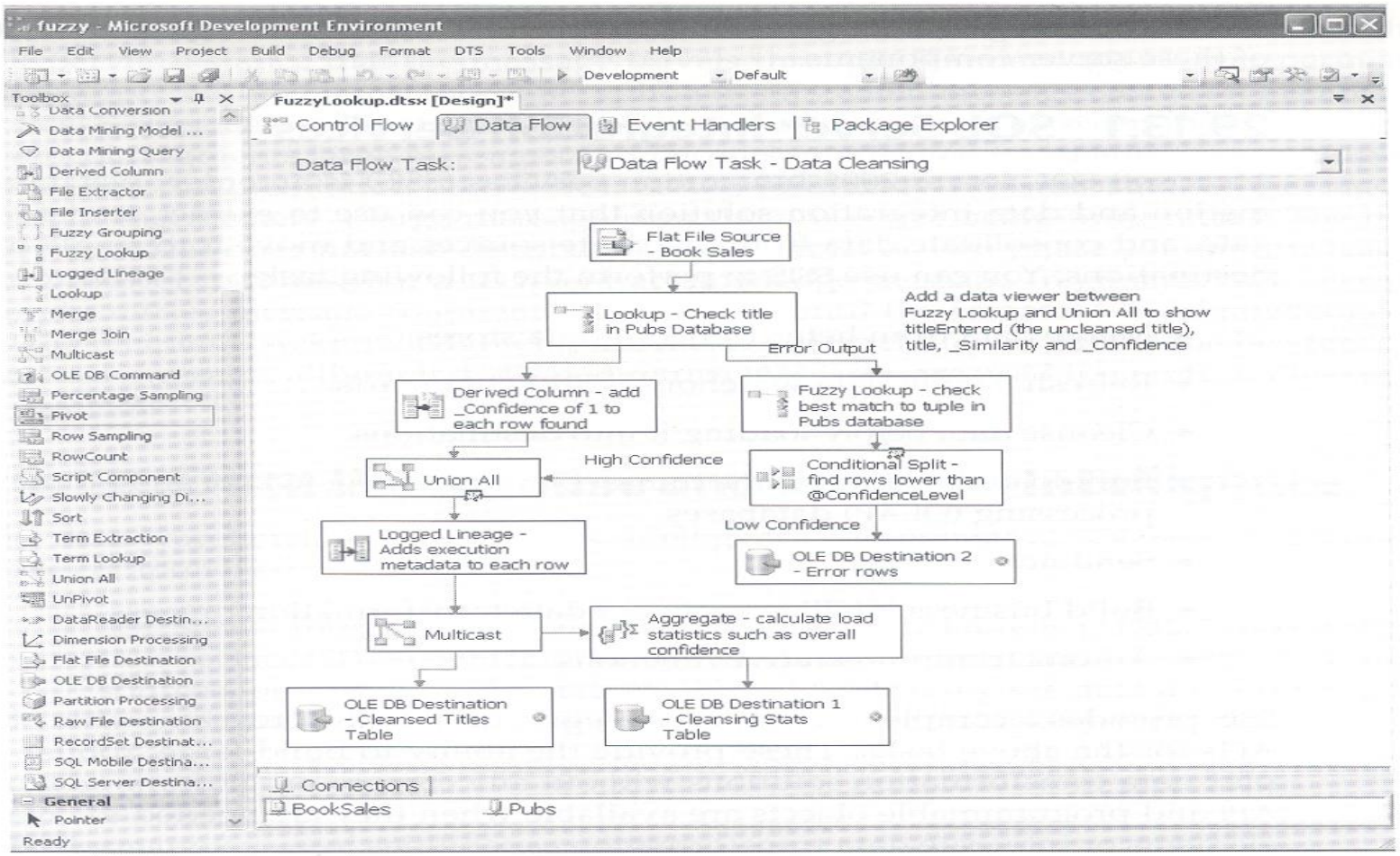
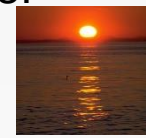



**Figure 29.7** Loading of data by using fuzzy lookup.

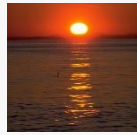
# MS- SQL Server Analysis Services (SSAS)

- **SSAS OLAP**
  - Analysis Server 2005 introduces a Unified Dimensional Model (UDM)
  - The role of UDM is to bridge the gap between traditional relational reporting and OLAP ad-hoc analysis
  - UDM provides a rich environment for defining powerful yet exhaustive business logic, rules, and semantic definition
- **SASS Data Mining**
  - SQL Server 2005 provides various mining techniques, with a rich graphical interfaces to view mining results
  - Mining algorithms supported include
    - Association rules
    - Classification and prediction techniques
    - Time series forecasting
    - Clustering techniques
  - SQL Server also supports the *Data-Mining Extensions* (DMX) extensions for SQL



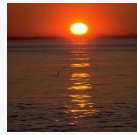
# SQL Server Reporting Services (SSRS)

- Reporting Services is a new server-based reporting platform that can be used to create and manage tabular, matrix, graphical, and free-form reports
- The reports can be viewed and managed over a Web-based connection



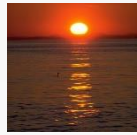
# Chapter 30: Microsoft SQL Server

- 30.0 History
- 30.1 Management, Design, and Querying Tools
- 30.2 SQL Variations and Extensions
- 30.3 Storage and Indexing
- 30.4 Query Processing and Optimization
- 30.5 Concurrency and Recovery
- 30.6 System Architecture
- 30.7 Data Access
- 30.8 Distributed Heterogeneous Query Processing
- 30.9 Replication
- 30.10 Server Programming in .NET
- 30.11 XML Support in SQL Server 2005
- 30.12 SQL Server Service Broker
- 30.13 Data Warehouse and Business Intelligence
- Summary & Bibliographical Notes



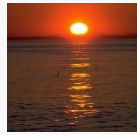
# Summary [1/2]

- Microsoft SQL Server is a complete data management package including relational database server, full text indexing and search, XML data import and export, distributed and heterogeneous data integration, analysis server and client for OLAP and data mining, replication among heterogeneous data stores, a programmable data transformation engine, and more
- Thus, SQL Server serves as a foundation of Microsoft's suite of enterprise server products
- SQL Server provides a suite of tools for managing all aspects of SQL Server development, querying, tuning, testing, and administration
- SQL Server allows application developers to write server-side business logic using Transact-SQL or a .NET programming language



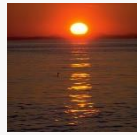
# Summary [2/2]

- SQL Server provides a container for physical structures such as tables and indices and for logical structures such as constraints and views
- SQL Server's transaction, logging, locking, and recovery subsystems realize the ACID properties expected of a database system
- SQL Server distributed heterogeneous query capability allows transactional queries and updates against various relational and nonrelational sources
- SQL Server supports the hosting of the .NET Common Language Runtime (CLR)
- SQL Server 2005 provides XML supprt, SQL Service Broker, and data warehouse and business intelligence



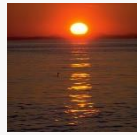
# Bibliographical Notes [1/4]

- Detailed information about using a C2 certified system with SQL Server is available at www.microsoft.com/Download/Release.asp?ReleaseID=25503
- SQL Server's optimization framework is based on the Cascades optimizer prototype, which Graefe[1995] proposed
- Simmen et al.[1996] discuss the scheme for reproducing grouping columns
- Galindo-Legria and Joshi[2001] present the variety of execution strategies that SQL Server considers during cost-based optimization
- Additional information on the self-tuning aspect of SQL Server are discussed by Chaudhuri et al.[1999]
- Chaudhuri and Shim[1994] and Yan and Larson[1995] discuss partial aggregation



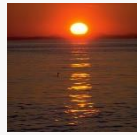
# Bibliographical Notes [2/4]

- Chatziantoniou and Ross [1997] and Galindo-Legria and Joshi[2001] proposed the alternative used by SQL Server for SQL queries requiring a self-join
- Pellenkft et al.[1997] discuss the optimization scheme for generating complete search space using a set of transformations that are complete, local, and nonredundant
- Graefe et al.[1998] offer discussion concerning hash operations that support basic aggregation and join
- Graefe et al. [1998] present the idea of joining indices for the sole purpose of assembling a row with the set of columns needed on a query



# Bibliographical Notes [3/4]

- Blakeley[1996] and Blakeley and Pizzo[2001] offer discussions concerning communication with the storage engine through OLE-DB
- Blakeley et al. [2005] detail the implementation of the distributed and heterogeneous query capabilities of SQL Server
- Acheson et al. [2004] provide details on integration of the .NET CLR inside the SQL Server process



# Bibliographical Notes [4/4]

- SQL:2003 standard is defined in SQL/XML [2004]
- Rys [2001] provides an overview of the extensions to the **for xml** aggregation
- For information on XML capabilities that can be used on the client side or inside CLR, refer to the collection of white papers at MSDN:XML Developer Center
- The XQuery 1.0/XPath 2.0 data model is defined in Walsh et al. [2004]
- Rys [2003] provides an overview of implementation techniques for XQuery in the context of relational databases
- The OrdPath numbering scheme is described in O'Neil et al. [2004]
- Pal et al. [2004] and Baras et al. [2005] provide more information on XML indexing and XQuery algebraization and optimization in SQL Server 2005