Database Concepts (VII)

Query Processing

Chaokun Wang

School of Software, Tsinghua University

chaokun@tsinghua.edu.cn

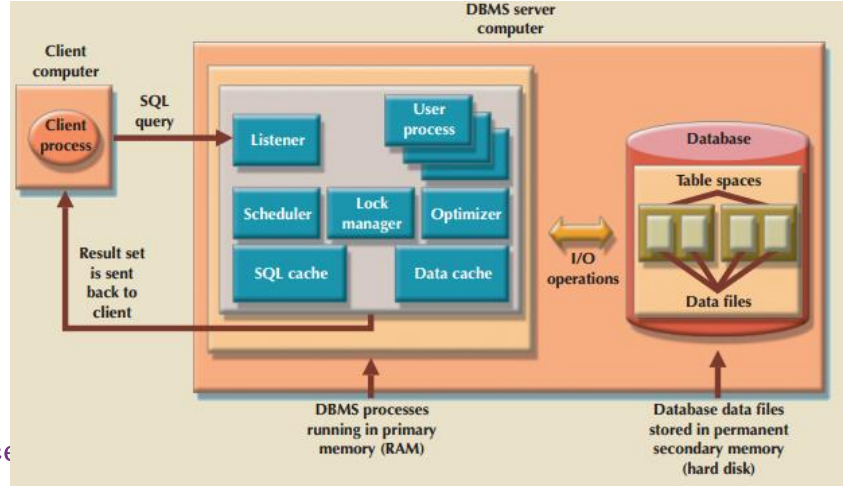
May 31, 2021

Outline

- Database Performance Tuning*
- Query Optimization*
- Distributed Database Management Systems

Database Performance-Tuning Concepts

- One of the main functions of a database system is to provide timely answers
  - End users interact with the DBMS through the use of queries to generate information, using the following sequence:
    1. End-user (client-end) application generates a query
    2. Query is sent to the DBMS (server end)
    3. DBMS (server end) executes the query
    4. DBMS sends the resulting data set to the end-user (client-end) application



Database Performance-Tuning Concepts

- Goal of database performance is to execute queries as fast as possible 核心就是要快
  - Database performance tuning: set of activities and procedures that reduce response time of database system
- Fine-tuning the performance of a system requires a holistic approach
  - All factors must operate at optimum level with minimal bottlenecks

Database Performance-Tuning Concepts

GENERAL GUIDELINES FOR BETTER SYSTEM PERFORMANCE

	SYSTEM RESOURCES	CLIENT	SERVER
Hardware	CPU	The fastest possible Dual-core CPU or higher "Virtualized Client desktop technologies could also be used."	The fastest possible Multiple processors (quad-core technology or higher) Cluster of networked computers "Virtualized server technology could be used"
	RAM	The maximum possible to avoid OS memory to disk swapping	The maximum possible to avoid OS memory to disk swapping
	Storage	Fast SATA/EIDE hard disk with sufficient free hard disk space Solid state drives (SSDs) for faster speed	Multiple high-speed, high-capacity disks Fast disk interface (SAS / SCSI / Firewire / Fibre Channel RAID configuration optimized for throughput Solid state drives (SSDs) for faster speed Separate disks for OS, DBMS, and data spaces
	Network	High-speed connection	High-speed connection
Software	Operating system (OS)	64-bit OS for larger address spaces Fine-tuned for best client application performance	64-bit OS for larger address spaces Fine-tuned for best server application performance
	Network	Fine-tuned for best throughput	Fine-tuned for best throughput
	Application	Optimize SQL in client application	Optimize DBMS server for best performance

Performance Tuning: Client and Server

客户端：查询语句怎么样有效合理

- Client side: SQL performance tuning
  - Generate SQL query that returns correct answer in least amount of time
    - Using minimum amount of resources at server

服务器端：考虑利用较小资源就能完成查询处理

- Server side: DBMS performance tuning
  - DBMS environment is configured to respond to clients' requests as fast as possible
    - While making optimum use of existing resources

DBMS Architecture

- All data in a database are stored in data files
  - Data files automatically expand in predefined increments known as extends
- Data files are grouped in file groups or table spaces
  - Logical grouping of several data files that store data with similar characteristics
- Data cache or buffer cache: shared, reserved memory area
  - Stores the most recently accessed data blocks in RAM

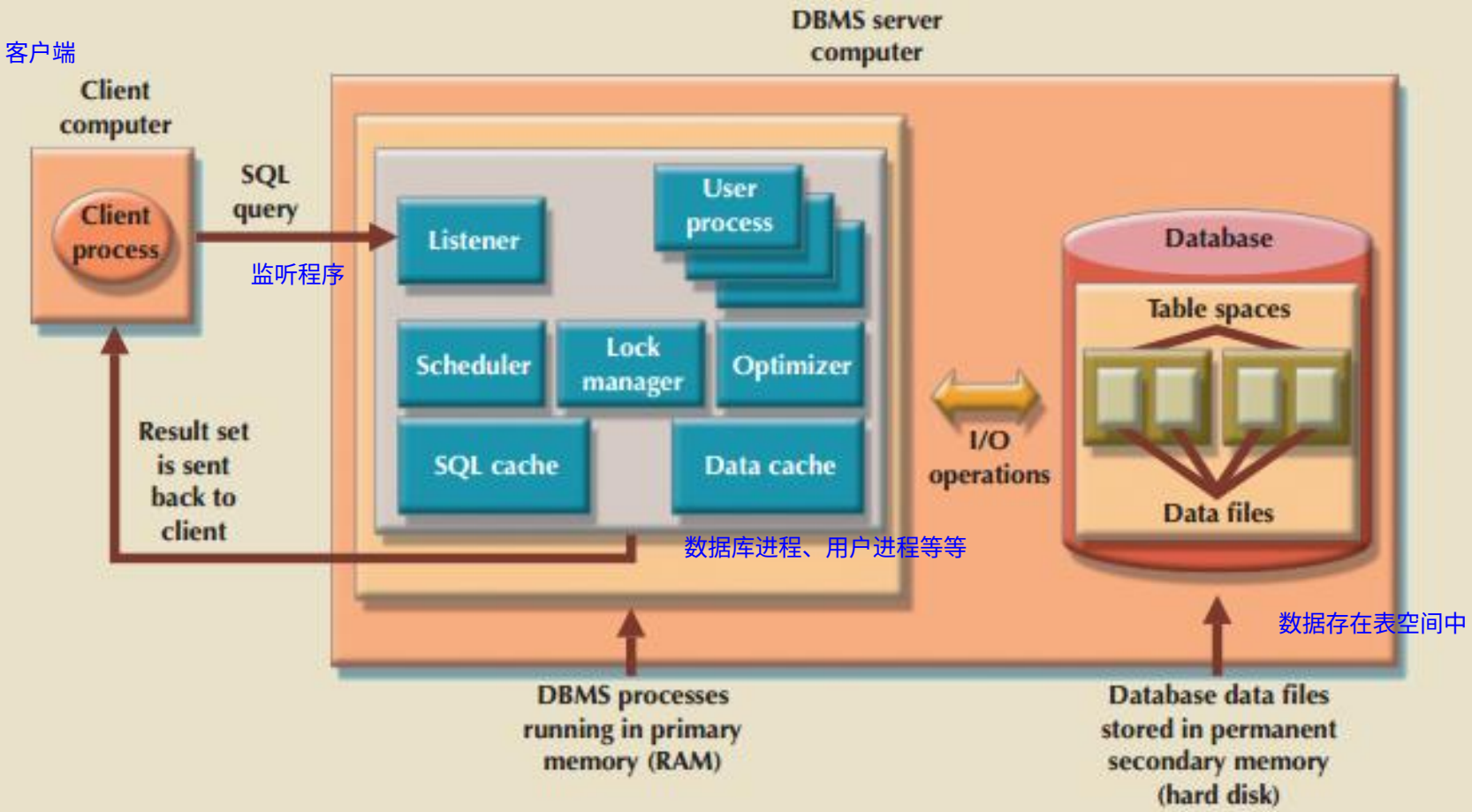
DBMS Architecture

- SQL cache or procedure cache: shared, reserved memory
  - Stores most recently executed SQL statements or PL/SQL procedures 等价的查询计划等
- DBMS retrieves data from permanent storage and places them in RAM
  - Data is retrieved from the data files and placed in the data cache
  - Input/output request: low-level data access operation that reads or writes data to and from computer devices
  - Data cache is faster than working with data files
  - Majority of performance-tuning activities focus on minimizing I/O operations

DBMS Architecture

简单认为客户端和服务器不在一台机器上

FIGURE 11.1 BASIC DBMS ARCHITECTURE



Database Query Optimization Modes

- Algorithms proposed for query optimization are based on selection of:
  - Optimum order to achieve the fastest execution time
  - Sites to be accessed to minimize communication costs 最小化通信开销
- Evaluated based on:
  - Operation mode
  - Timing of its optimization

Database Query Optimization Modes

- Classification of operation modes
  - Automatic query optimization: DBMS finds the most cost-effective access path without user intervention
  - Manual query optimization: requires that optimization be selected and scheduled by the end-user or programmer

自动的查询优化 -> 最希望看到的

人工手动的查询优化，此时需要用户/程序员介入

得看你的系统是否支持

Database Query Optimization Modes

- Classification based on timing of optimization
  - Static query optimization: best optimization strategy is selected when the query is compiled by the DBMS
    - Takes place at compilation time
  - Dynamic query optimization: access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database
    - Takes place at execution time

Database Query Optimization Modes

- Classification based on type of information used to optimize the query
  - Statistically based query optimization algorithm: statistics are used by the DBMS to determine the best access strategy
  - Statistical information is generated by DBMS through:
    - Dynamic statistical generation mode
    - Manual statistical generation mode
  - Rule-based query optimization algorithm: based on a set of user-defined rules to determine the best query access strategy

Database Statistics

- Measurements about database objects; provide a snapshot of database characteristics
  - Number of processors used
  - Processor speed
  - Temporary space available

Database Statistics

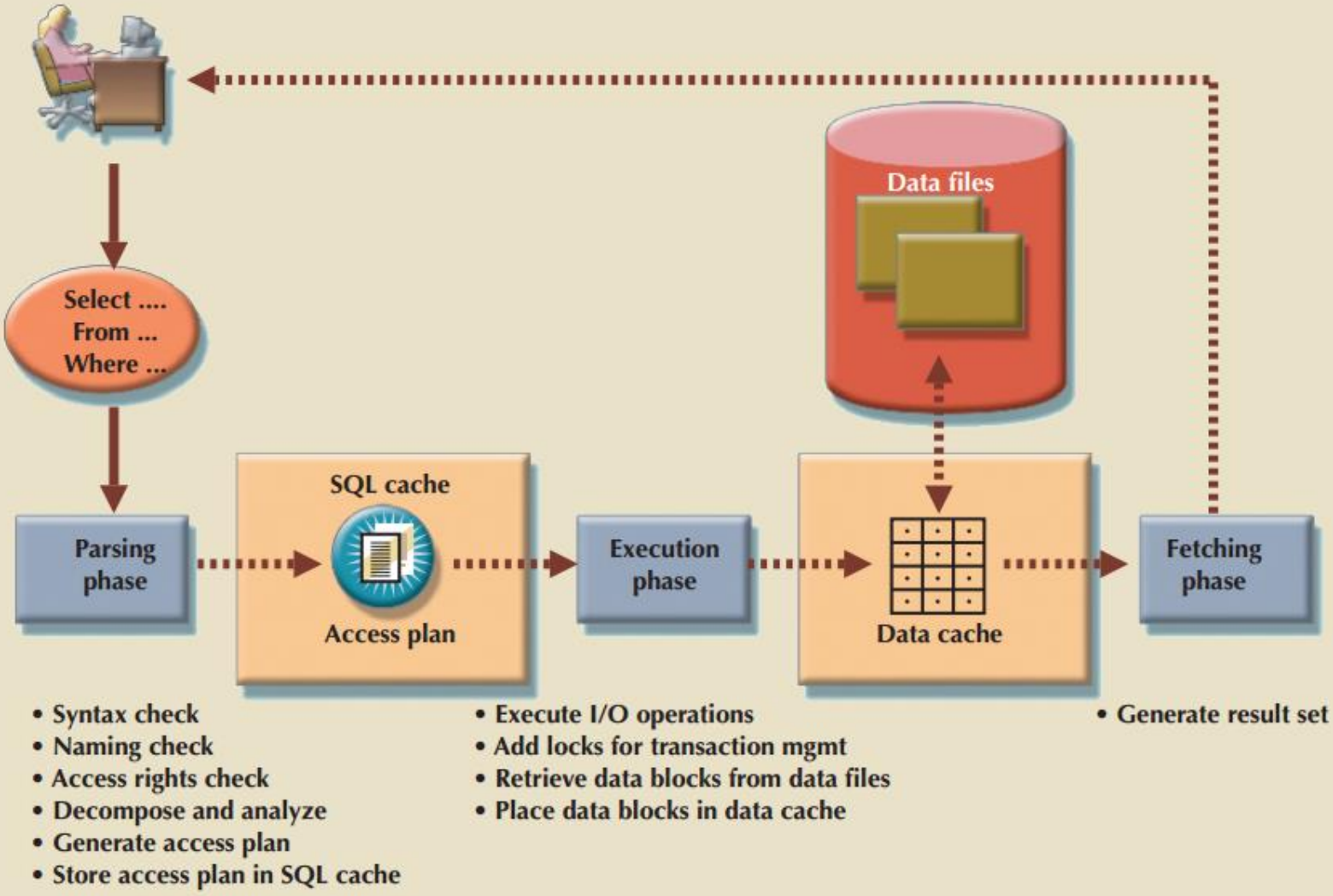
SAMPLE DATABASE STATISTICS MEASUREMENTS	
DATABASE OBJECT	**SAMPLE MEASUREMENTS**
Tables	Number of rows, number of disk blocks used, row length, number of columns in each row, number of distinct values in each column, maximum value in each column, minimum value in each column, and columns that have indexes
Indexes	Number and name of columns in the index key, number of key values in the index, number of distinct key values in the index key, histogram of key values in an index, and number of disk pages used by the index
Environment Resources	Logical and physical disk block size, location and size of data files, and number of extends per data file

Query Processing

- Parsing 应用程序→解析器→重写器→优化器→执行器→应用程序
  - DBMS parses the SQL query and chooses the most efficient access/execution plan
- Execution
  - DBMS executes the SQL query using the chosen execution plan
- Fetching
  - DBMS fetches the data and sends the result set back to the client

Query Processing

FIGURE 11.2 QUERY PROCESSING



SQL Parsing Phase

- Query is broken down into smaller units
  - Original SQL query transformed into slightly different version of original SQL code which is fully equivalent and more efficient
- Query optimizer: analyzes SQL query
  - Finds most efficient way to access data
- Access plans: result of parsing a SQL statement; contains a series of steps the DBMS will use to execute the query and return the result set in the most efficient way
  - Access plan exists for query in SQL cache: DBMS reuses it
  - No access plan: optimizer evaluates various plans and chooses one to be placed in SQL cache for use

Query Processing

了解就行

SAMPLE DBMS ACCESS PLAN I/O OPERATIONS	
OPERATION	**DESCRIPTION**
Table scan (full) 全表扫描	Reads the entire table sequentially, from the first row to the last, one row at a time (slowest)
Table access (row ID) 查某一行	Reads a table row directly, using the row ID value (fastest)
Index scan (range) 基于索引查	Reads the index first to obtain the row IDs and then accesses the table rows directly (faster than a full table scan)
Index access (unique)	Used when a table has a unique index in a column
Nested loop 嵌套循环	Reads and compares a set of values to another set of values, using a nested loop style (slow)
Merge	Merges two data sets (slow)
Sort	Sorts a data set (slow)

SQL Execution Phase

- All I/O operations indicated in the access plan are executed
  - Locks are acquired
  - Data are retrieved and placed in data cache
  - Transaction management commands are processed

SQL Fetching Phase

- Rows of resulting query result set are returned to client
  - DBMS may use temporary table space to store temporary data
  - Database server coordinates the movement of the result set rows from the server cache to the client cache

Query Processing Bottlenecks

- Delay introduced in the processing of an I/O operation that slows the system
  - Caused by the:
    - CPU
    - RAM
    - Hard disk
    - Network
    - Application code

Outline

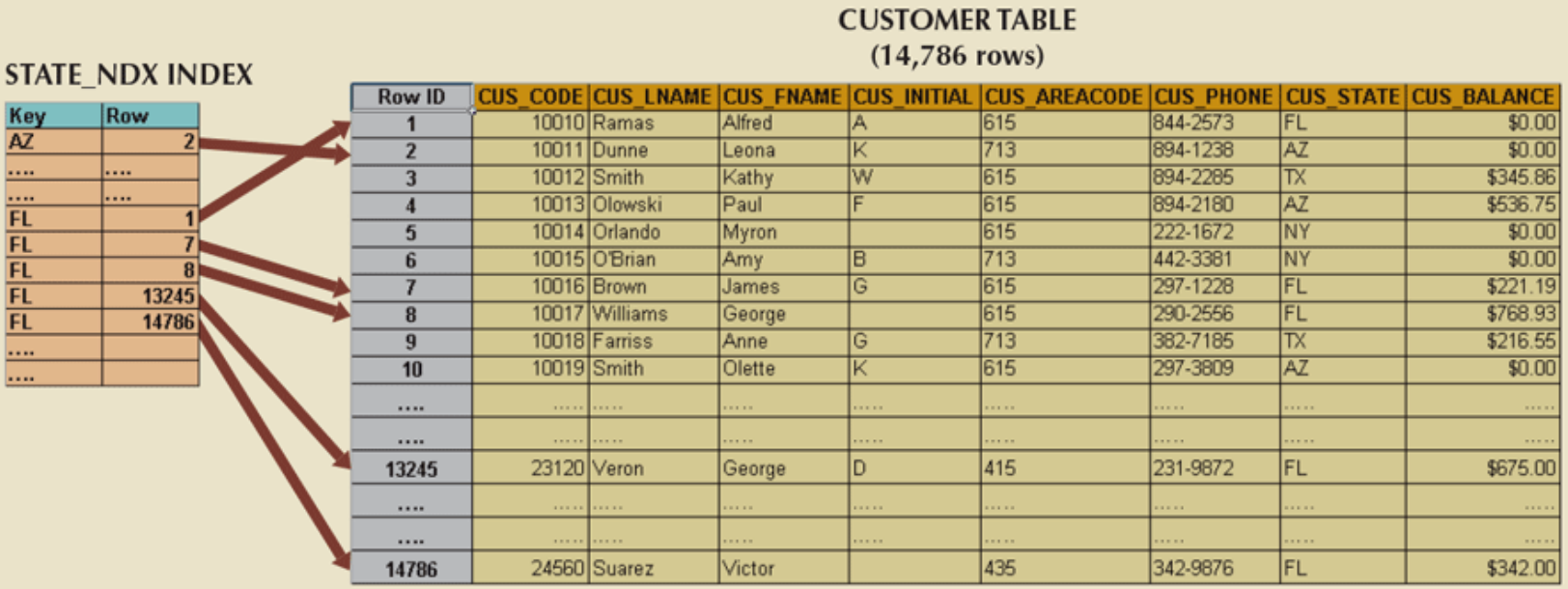
- Database Performance Tuning
- Query Optimization
- Distributed Database Management Systems

Indexes and Query Optimization

- Indexes
  - Help speed up data access
  - Facilitate searching, sorting, using aggregate functions, and join operations
  - Ordered set of values that contain the index key and pointers
  - More efficient than a full table scan; index data is preordered and the amount of data is usually much smaller

Indexes and Query Optimization

FIGURE 11.3 INDEX REPRESENTATION FOR THE CUSTOMER TABLE



STATE_NDX INDEX

Key	Row
AZ	2
....	....
....	....
FL	1
FL	7
FL	8
FL	13245
FL	14786
....	
....	

CUSTOMER TABLE
(14,786 rows)

Row ID	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_STATE	CUS_BALANCE
1	10010	Ramas	Alfred	A	615	844-2573	FL	$0.00
2	10011	Dunne	Leona	K	713	894-1238	AZ	$0.00
3	10012	Smith	Kathy	W	615	894-2285	TX	$345.86
4	10013	Olowski	Paul	F	615	894-2180	AZ	$536.75
5	10014	Orlando	Myron		615	222-1672	NY	$0.00
6	10015	O'Brian	Amy	B	713	442-3381	NY	$0.00
7	10016	Brown	James	G	615	297-1228	FL	$221.19
8	10017	Williams	George		615	290-2556	FL	$768.93
9	10018	Farriss	Anne	G	713	382-7185	TX	$216.55
10	10019	Smith	Olette	K	615	297-3809	AZ	$0.00
....	.....	.....	.....	.....	.....	.....	.....	.....
....	.....	.....	.....	.....	.....	.....	.....	.....
13245	23120	Veron	George	D	415	231-9872	FL	$675.00
....	.....	.....	.....	.....	.....	.....	.....	.....
....	.....	.....	.....	.....	.....	.....	.....	.....
14786	24560	Suarez	Victor		435	342-9876	FL	$342.00

Indexes and Query Optimization

- Data sparsity: number of different values a column could have
  - High or low
- Data structures used to implement indexes
  - Hash indexes
  - B-tree indexes
  - Bitmap indexes
- DBMS determines the best type of index to use

B-Tree index is used in columns with high data sparsity—that is, columns with many different values relative to the total number of rows.

Bitmap index is used in columns with low data sparsity—that is, columns with few different values relative to the total number of rows.

重复性较高时用bitmap

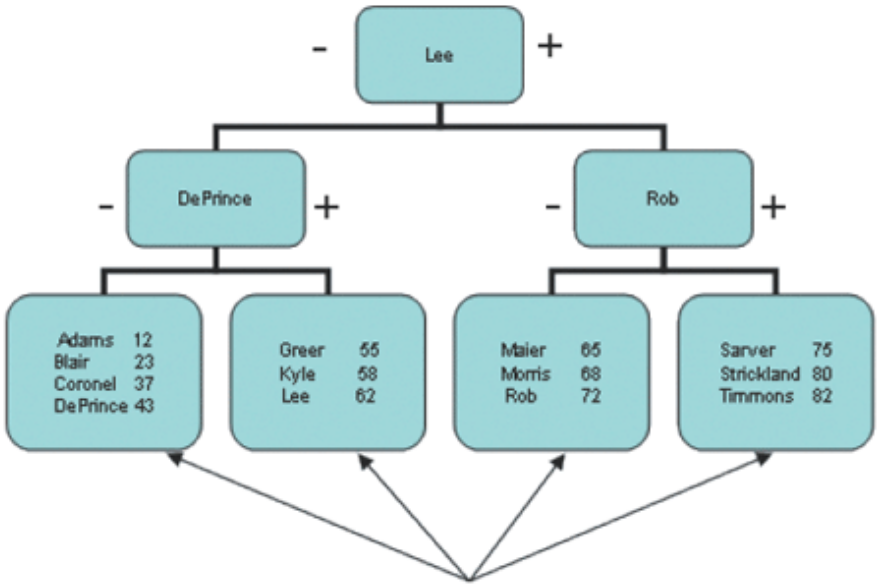
CUSTOMER TABLE

CUS_ID	CUS_LNAME	CUS_FNAME	CUS_PHONE	REGION_CODE
12	Adams	Charlie	4533	NW
23	Blair	Robert	5426	SE
37	Coronel	Carlos	2358	SW
43	DePrince	Albert	6543	NE
55	Greer	Tim	2764	SE
58	Kyle	Ruben	2453	SW
62	Lee	John	7895	NE
65	Maier	Jerry	7689	NW
68	Morris	Steve	4568	NW
72	Rob	Pete	8123	NE
75	Sarver	Lee	8193	SE
80	Strickland	Tomas	3129	SW
82	Timmons	Douglas	3499	NE

B-tree Index

On CUS_LNAME

希望数据本身是高稀疏的



Leaf objects contain index: key and pointers to rows in table. Access to any row using the index will take the same number of I/O accesses. In this example, it would take four I/O accesses to access any given table row using the index: One for each index tree level (root, branch, leaf object) plus access to data row using the pointer.

Bitmap Index

On REGION_CODE

Region →

NE	NW	SE	SW		
Bit 1	Bit 2	Bit 3	Bit 4	Bit ...	Bit ...
0	1	0	0		
0	0	1	0		
0	0	0	1		
1	0	0	0		
0	0	1	0		
0	0	0	1		
1	0	0	0		
0	1	0	0		
0	1	0	0		
1	0	0	0		
0	0	1	0		
0	0	0	1		
1	0	0	0		

← One byte

In the bitmap index, each bit represents one region code. In the first row, bit number two is turned on, thus indicating that the first row region code value is NW.

REGION_CODE = 'NW'

Each byte in the bitmap index represents one row of the table data. Bitmap indexes are very efficient with searches. For example, to find all customers in the NW region, the DBMS will return all rows with bit number two turned on.

Optimizer Choices

- Rule-based optimizer: uses preset rules and points to determine the best approach to execute a query
  - Rules assign a “fixed cost” to each SQL operation
- Cost-based optimizer: uses algorithms based on statistics about objects being accessed to determine the best approach to execute a query
  - Optimizer process adds up the processing cost, I/O costs, and resource costs (RAM and temporary space) to determine the total cost of a given execution plan

list all products provided by a vendor based in Florida

COMPARING ACCESS PLANS AND I/O COSTS

PLAN	STEP	OPERATION	I/O OPERATIONS	I/O COST	RESULTING SET ROWS	TOTAL I/O COST
A	A1	Cartesian product (PRODUCT, VENDOR)	7,000 + 300	7,300	2,100,000	7,300
	A2	Select rows in A1 with matching vendor codes	2,100,000	2,100,000	7,000	2,107,300
	A3	Select rows in A2 with V_STATE = 'FL'	7,000	7,000	1,000	**2,114,300**
B	B1	Select rows in VENDOR with V_STATE = 'FL'	300	300	10	300
	B2	Cartesian Product (PRODUCT, B1)	7,000 + 10	7,010	70,000	7,310
	B3	Select rows in B2 with matching vendor codes	70,000	70,000	1,000	**77,310**

- The PRODUCT table has 7,000 rows.
- The VENDOR table has 300 rows.
- Ten vendors are located in Florida.
- One thousand products come from vendors in Florida.

```
SELECT   P_CODE, P_DESCRIPT, P_PRICE, V_NAME, V_STATE
FROM     PRODUCT, VENDOR
WHERE    PRODUCT.V_CODE = VENDOR.V_CODE
         AND VENDOR.V_STATE = 'FL';
```

Using Hints to Affect Optimizer Choices

- Optimizer might not choose the best execution plan
  - Makes decisions based on existing statistics; might be old
  - Might choose less-efficient decisions
- Optimizer hints: special instructions for the optimizer
  - Embedded in the SQL command text

Using Hints to Affect Optimizer Choices

OPTIMIZER HINTS

HINT	USAGE
ALL_ROWS	Instructs the optimizer to minimize the overall execution time—that is, to minimize the time needed to return all rows in the query result set. This hint is generally used for batch mode processes. For example: SELECT /*+ ALL_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
FIRST_ROWS	Instructs the optimizer to minimize the time needed to process the first set of rows—that is, to minimize the time needed to return only the first set of rows in the query result set. This hint is generally used for interactive mode processes. For example: SELECT /*+ FIRST_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
INDEX(name)	Forces the optimizer to use the P_QOH_NDX index to process this query. For example: SELECT /*+ INDEX(P_QOH_NDX) */ * FROM PRODUCT WHERE P_QOH < 10

SQL Performance Tuning

- Evaluated from client perspective
  - Most current relational DBMSs perform automatic query optimization at the server end
  - Most SQL performance optimization techniques are DBMS-specific and thus rarely portable
- Majority of performance problems are related to poorly written SQL code
  - A carefully written query almost always outperforms a poorly written one

Index Selectivity

- Measure of the likelihood that an index will be used in query processing
  - Indexes are used when a subset of rows from a large table is to be selected based on a given condition
    - Cannot always be used to improve performance
- Function-based index: based on a specific SQL function or expression
    - An index on EMP_SALARY + EMP_COMMISSION

Conditional Expressions

- Expressed within WHERE or HAVING clauses of a SQL statement
  - Restricts the output of a query to only rows matching conditional criteria

CONDITIONAL CRITERIA		
OPERAND1	CONDITIONAL OPERATOR	OPERAND2
P_PRICE	>	10.00
V_STATE	=	FL
V_CONTACT	LIKE	Smith%
P_QOH	>	P_MIN * 1.10

Conditional Expressions

- Expressed within WHERE or HAVING clauses of a SQL statement
  - Restricts the output of a query to only rows matching conditional criteria
- Guidelines to write efficient conditional expressions in SQL code
  - Use simple columns or literals as operands
    - P_PRICE > 10.00 is faster than P_QOH > P_MIN * 1.10
  - Numeric field comparisons are faster than character, date, and NULL comparisons
  - Equality comparisons are faster than inequality comparisons
  - Transform conditional expressions to use literals
    - P_PRICE − 10 = 7

      -> P_PRICE = 17

Conditional Expressions

AND: 把最容易拒绝的放前面去
OR： 把最容易对的放前面去

- Guidelines to write efficient conditional expressions in SQL code (cont'd)
  - Write equality conditions first when using multiple conditional expressions
    - P_QOH < P_MIN AND P_MIN = P_REORDER AND P_QOH = 10
      -> P_QOH = 10 AND P_MIN = P_REORDER AND P_MIN > 10
  - When using multiple AND conditions, write the condition most likely to be false first
  - When using multiple OR conditions, put the condition most likely to be true first
  - Avoid the use of NOT logical operator

Query Formulation

- Step to formulate a query
  - Pinpoint what columns and computations are required
  - Identify source tables
  - Decide how to join tables
  - Establish action criteria are needed
  - Determine the order in which to display the output

DBMS Performance Tuning

- Managing DBMS processes in primary memory and the structures in physical storage
  - DBMS performance tuning at server end focuses on setting parameters used for:
    - Data cache
    - SQL cache
    - Sort cache
    - Optimizer mode
- In-memory database: store large portions of the database in primary storage
  - These systems are becoming popular
    - Increasing performance demands of modern database applications
    - Diminishing costs
    - Technology advances of components

DBMS Performance Tuning

- Recommendations for physical storage of databases
  - Utilize I/O accelerators
  - Use RAID (Redundant Array of Independent Disks) to provide a balance between performance improvement and fault tolerance
  - Minimize disk contention
  - Put high-usage tables in their own table spaces
  - Assign separate data files in separate storage volumes for indexes, system, and high-usage tables
  - Take advantage of the various table storage organizations in the database
  - Partition tables based on usage
  - Apply denormalized tables where appropriate
  - Store computed and aggregate attributes in tables

SQL Commands

EXPLAIN

EXPLAIN — show the execution plan of a statement

Synopsis

```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement

where option can be one of:

    ANALYZE [ boolean ]
    VERBOSE [ boolean ]
    COSTS [ boolean ]
    SETTINGS [ boolean ]
    BUFFERS [ boolean ]
    WAL [ boolean ]
    TIMING [ boolean ]
    SUMMARY [ boolean ]
    FORMAT { TEXT | XML | JSON | YAML }
```

```
EXPLAIN SELECT * FROM foo;

                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
(1 row)
```

```
EXPLAIN SELECT * FROM foo WHERE i = 4;

                          QUERY PLAN
--------------------------------------------------------------
 Index Scan using fi on foo  (cost=0.00..5.98 rows=1 width=4)
   Index Cond: (i = 4)
(2 rows)
```

```
PREPARE query(int, int) AS SELECT sum(bar) FROM test
    WHERE id > $1 AND id < $2
    GROUP BY foo;

EXPLAIN ANALYZE EXECUTE query(100, 200);

                                                       QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=9.54..9.54 rows=1 width=8) (actual time=0.156..0.161 rows=11 loops=1)
   Group Key: foo
   ->  Index Scan using test_pkey on test  (cost=0.29..9.29 rows=50 width=8) (actual time=0.039..0.091 rows=99 loops=1)
         Index Cond: ((id > $1) AND (id < $2))
 Planning time: 0.197 ms
 Execution time: 0.225 ms
(6 rows)
```

Outline

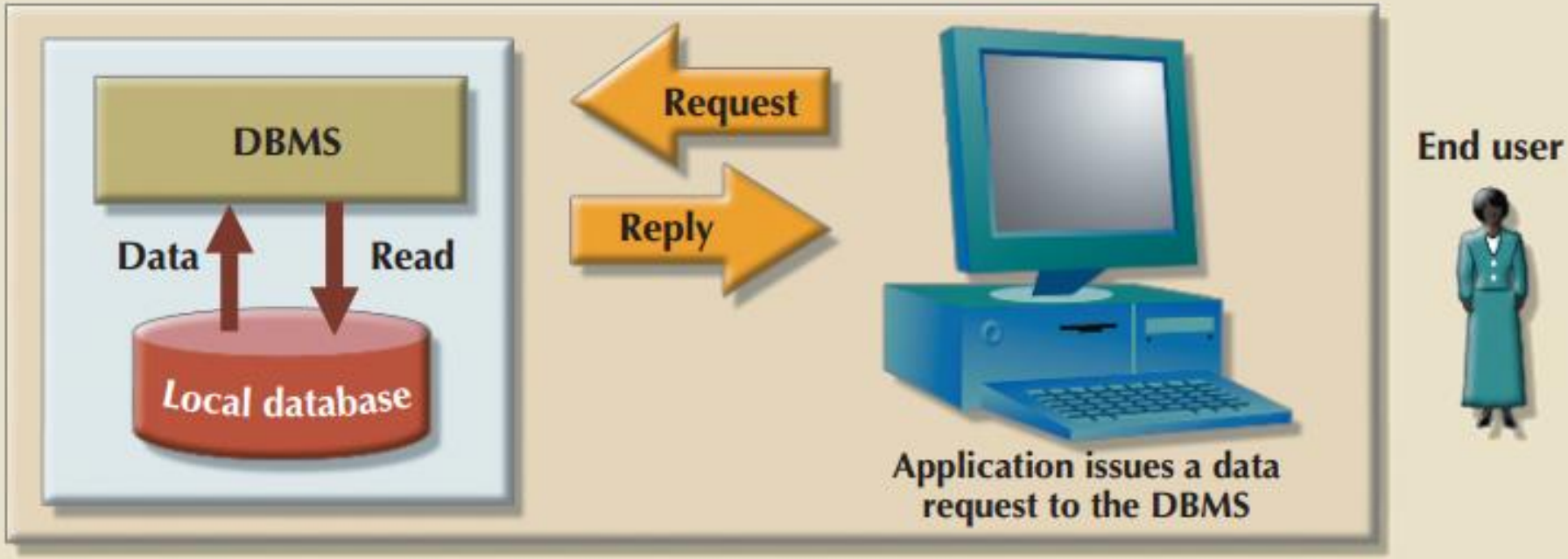
- Database Performance Tuning
- Query Optimization
- Distributed Database Management Systems

The Evolution of Distributed DBMS

- A distributed database management system (DDBMS)
  - Governs storage and processing of logically related data over interconnected computer systems
  - Distributes data and processing functions among several sites
- Centralized database management system
  - Required corporate data be stored in a single central site
  - Provided data access through dumb terminals
  - Filled structured information needs of corporations; fell short when quickly moving events required faster response times and equally quick access to information

The Evolution of Distributed DBMS

FIGURE 12.1 CENTRALIZED DATABASE MANAGEMENT SYSTEM

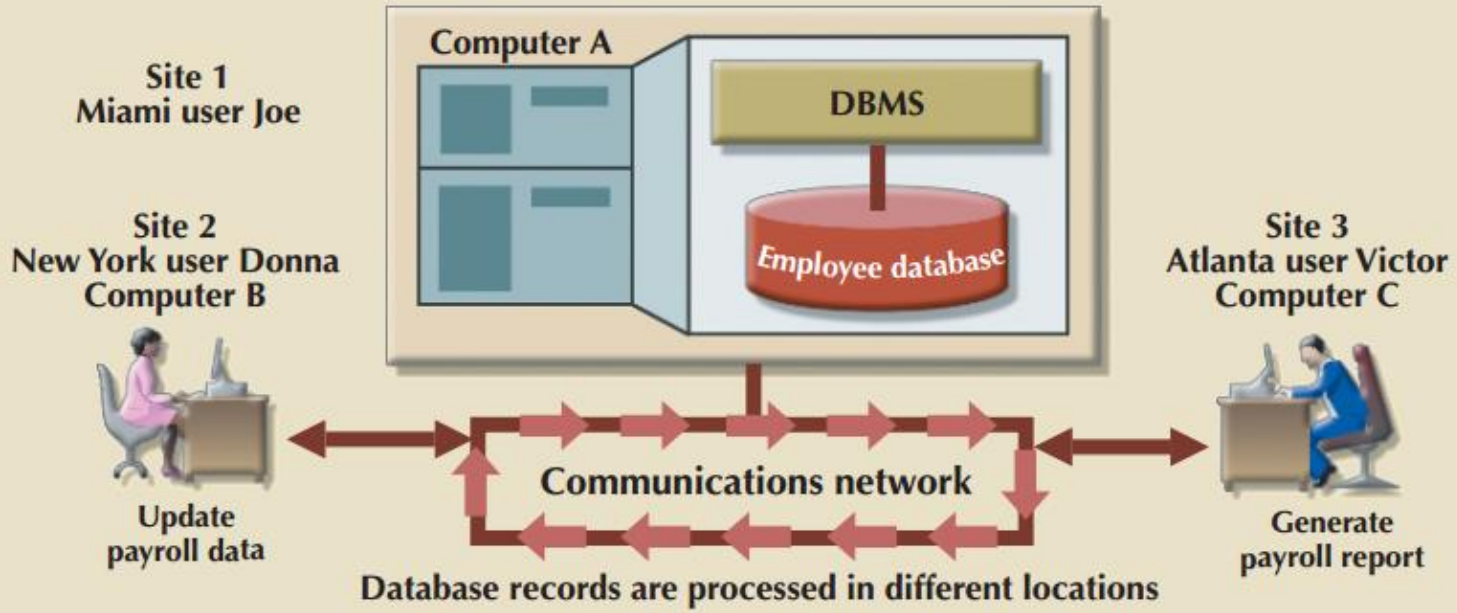


Distributed Processing and Distributed Databases

- Distributed processing: database's logical processing is shared among two or more physically independent sites via network
  - Distributed database: stores logically related database over two or more physically independent sites via a computer network
  - Database fragments: database composed of many parts in distributed database system

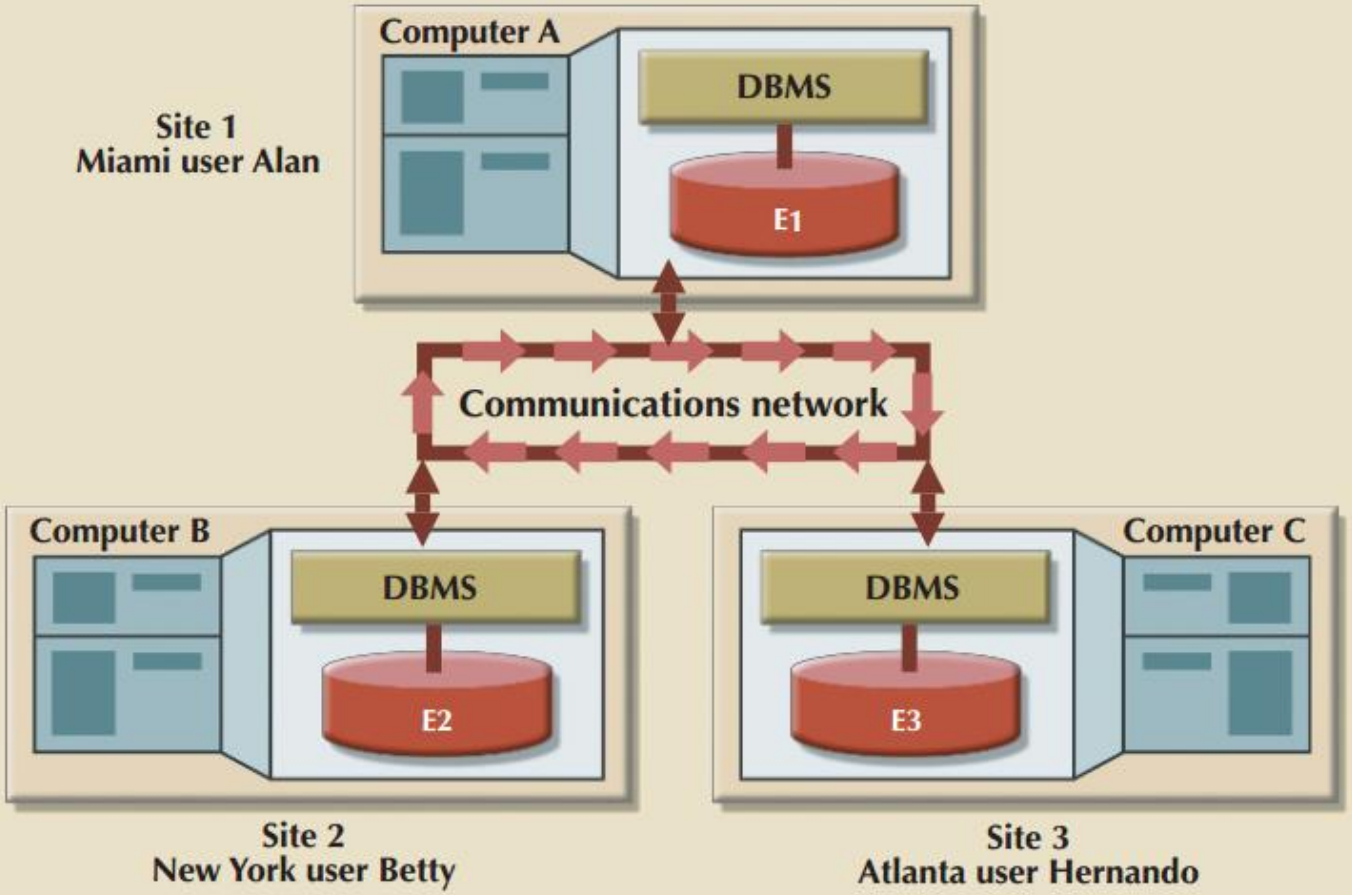
Distributed Processing and Distributed Databases

FIGURE 12.2 DISTRIBUTED PROCESSING ENVIRONMENT



Distributed Processing and Distributed Databases

FIGURE 12.3 DISTRIBUTED DATABASE ENVIRONMENT



Characteristics of Distributed Management Systems

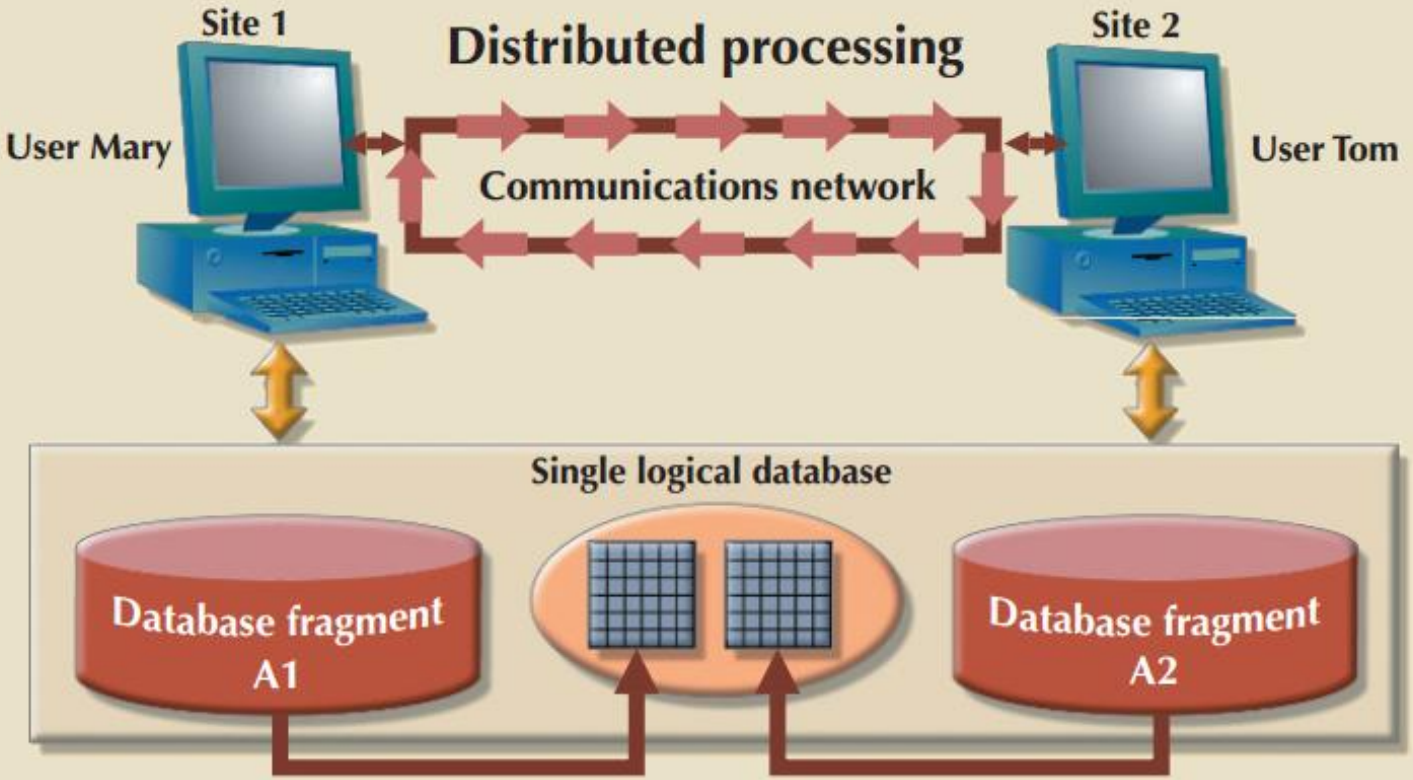
- A DBMS must have several functions to be classified as distributed
  - Application interface
  - Validation
  - Transformation
  - Query optimization
  - Mapping
  - I/O interface
  - Formatting
  - Security
  - Backup and recovery
  - DB administration
  - Concurrency control
  - Transaction management

Characteristics of Distributed Management Systems

- Functions of fully distributed DBMS
  - Receive the request of an application or end user
  - Validate, analyze, and decompose the request
  - Map request's logical-to-physical data components
  - Decompose request into several I/O operations
  - Search, locate, read and validate data
  - Ensure database consistency, security, and integrity
  - Validate data for conditions specified by request
  - Present data in required format
  - Handle all necessary functions transparently to user

Characteristics of Distributed Management Systems

FIGURE 12.4 A FULLY DISTRIBUTED DATABASE MANAGEMENT SYSTEM



DDBMS Components

- The DDBMS must include at least the following components:
  - Computer workstations or remote devices
  - Network hardware and software components
  - Communications media
  - Transaction processor (TP)
  - Data processor (DP) or data manager (DM)