QUERY PROCESSING

Engr. Laraib Siddiqui

Database performance-tuning concepts

- Goal of database performance is to execute queries as fast as possible
- **Database performance tuning**
  - Set of activities and procedures designed to reduce response time of database system
- All factors must operate at optimum level with minimal bottlenecks
- Good database performance starts with good database design

Key Terms

Query Processing

The activities involved in parsing, validating, optimizing, and executing a query.

Query Optimization

The activity of choosing an efficient execution strategy for processing a query.

Comparison of different processing strategies

Find all Managers who work at a London branch.

SELECT *

FROM Staff s, Branch b

WHERE s.branchNo = b.branchNo **AND**

(s.position = 'Manager' **AND** b.city = 'London');

Three equivalent relational algebra queries corresponding to this SQL statement are:

$$\sigma_{(\text{position}=\text{'Manager'}) \wedge (\text{city}=\text{'London'}) \wedge (\text{Staff.branchNo}=\text{Branch.branchNo})}(\text{Staff} \times \text{Branch})$$

$$\sigma_{(\text{position}=\text{'Manager'}) \wedge (\text{city}=\text{'London'})}(\text{Staff} \bowtie_{\text{Staff.branchNo}=\text{Branch.branchNo}} \text{Branch})$$

$$(\sigma_{\text{position}=\text{'Manager'}}(\text{Staff})) \bowtie_{\text{Staff.branchNo}=\text{Branch.branchNo'}} (\sigma_{\text{city}=\text{'London'}}(\text{Branch}))$$

Comparison of different processing strategies

Assume that there are 1000 tuples in Staff, 50 tuples in Branch, 50 Managers (one for each branch), and 5 London branches.

$$\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}(Staff \times Branch)$$

The first query calculates the Cartesian product of Staff and Branch, which requires (1000 + 50) disk accesses to read the relations, and creates a relation with (1000 * 50) tuples. We then have to read each of these tuples again to test them against the selection predicate at a cost of another (1000 * 50) disk accesses, giving a total cost of: (1000 + 50) + 2*(1000 * 50) = 101 050 disk accesses

$$\sigma_{(position='Manager') \wedge (city='London')}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch)$$

The second query joins Staff and Branch on the branch number branchNo, which again requires (1000 + 50) disk accesses to read each of the relations. We know that the join of the two relations has 1000 tuples, one for each member of staff (a member of staff can only work at one branch). Consequently, the Selection operation requires 1000 disk accesses to read the result of the join, giving a total cost of: 2*1000 + (1000 + 50) = 3050 disk accesses

Comparison of different processing strategies

$$(\sigma_{\text{position}=\text{'Manager'}}(\text{Staff})) \bowtie_{\text{Staff.branchNo}=\text{Branch.branchNo}} (\sigma_{\text{city}=\text{'London'}}(\text{Branch}))$$

The final query first reads each Staff tuple to determine the Manager tuples, which requires 1000 disk accesses and produces a relation with 50 tuples. The second Selection operation reads each Branch tuple to determine the London branches, which requires 50 disk accesses and produces a relation with 5 tuples. The final operation is the join of the reduced Staff and Branch relations, which requires (50 + 5) disk accesses, giving a total cost of: 1000 + 2*50 + 5 + (50 + 5) = 1160 disk accesses

DBMS Architecture

All data in database are stored in **data files**

Data files
- Can automatically expand in predefined increments known as **extends**
- Grouped in file groups or table spaces

Table space or **file group**
- Logical grouping of several data files that store data with similar characteristics

Data cache or **buffer cache**
- Shared, reserved memory area that stores the most recently accessed data blocks in RAM

SQL cache or **procedure cache**
- Shared, reserved memory area that stores the most recently executed SQL statements or PL/SQL procedures, including triggers and functions

DBMS retrieves data from permanent storage and places it in RAM

DBMS Architecture

Input/output (I/O) request: low-level data access operation to and from computer devices

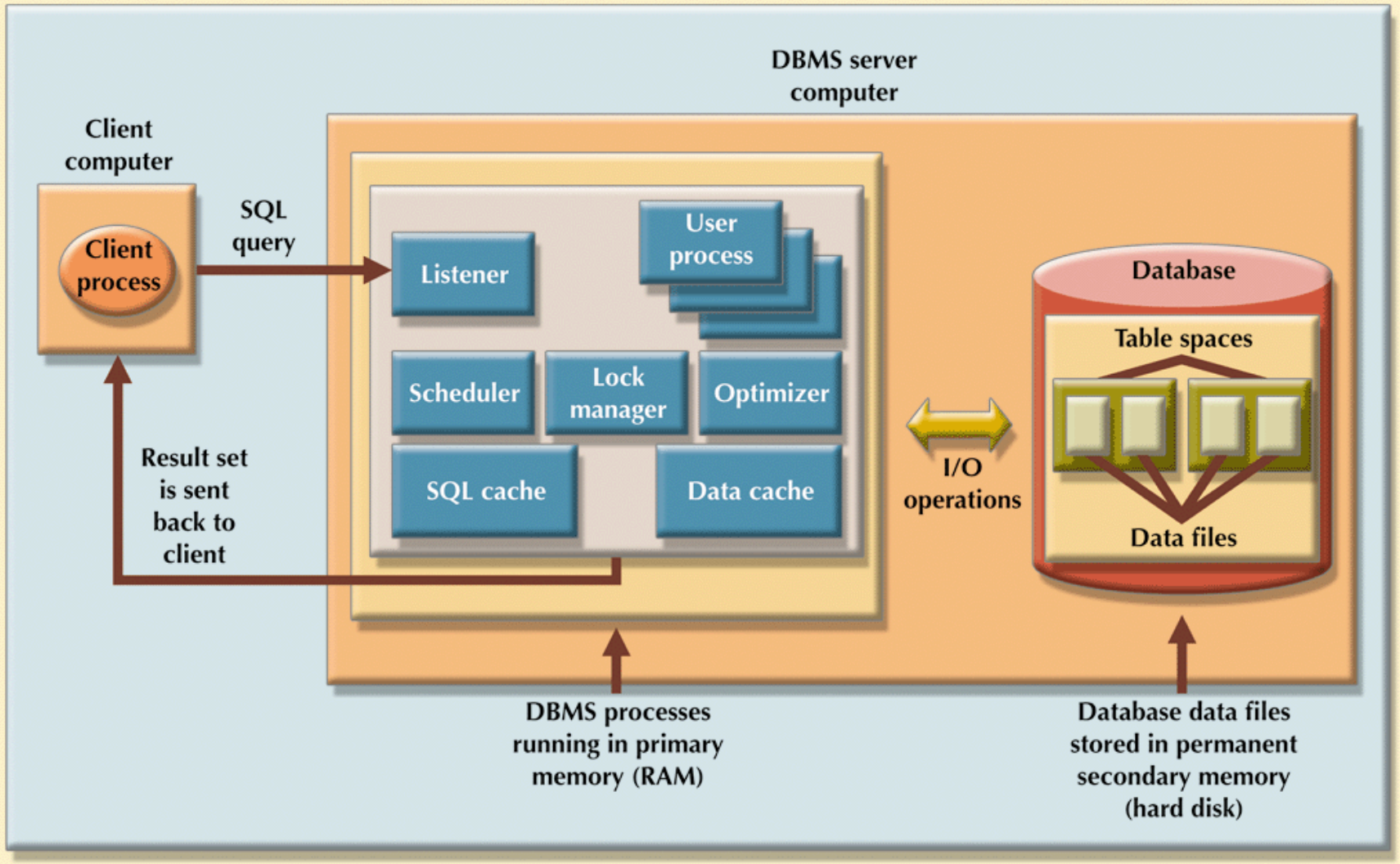
Data cache is faster than data in data files because the DBMS does not wait for hard disk to retrieve data

Majority of performance-tuning activities focus on minimizing the number of I/O operations

Typical DBMS processes:

Listener, user, scheduler, lock manager, optimizer

FIGURE 11.1 Basic DBMS architecture



SOURCE: Course Technology/Cengage Learning

DBMS Processes

Listener: Takes clients' requests and handles the processing of the SQL requests to other DBMS processes. Once received, the request is passes to appropriate user process

User: Created to manage each client session and handles all requests submitted to the server

Scheduler: Organizes the concurrent execution of SQL requests

Lock Manager: Manages all locks placed on database objects

Optimizer: Analyzes SQL queries and finds the most efficient way to access the data

Database statistics

- Number of measurements about database objects and available resources:
  - Number of processors used
  - Processor speed
  - Temporary space available
- Make critical decisions about improving query processing efficiency
- Can be gathered manually by DBA or automatically by DBMS

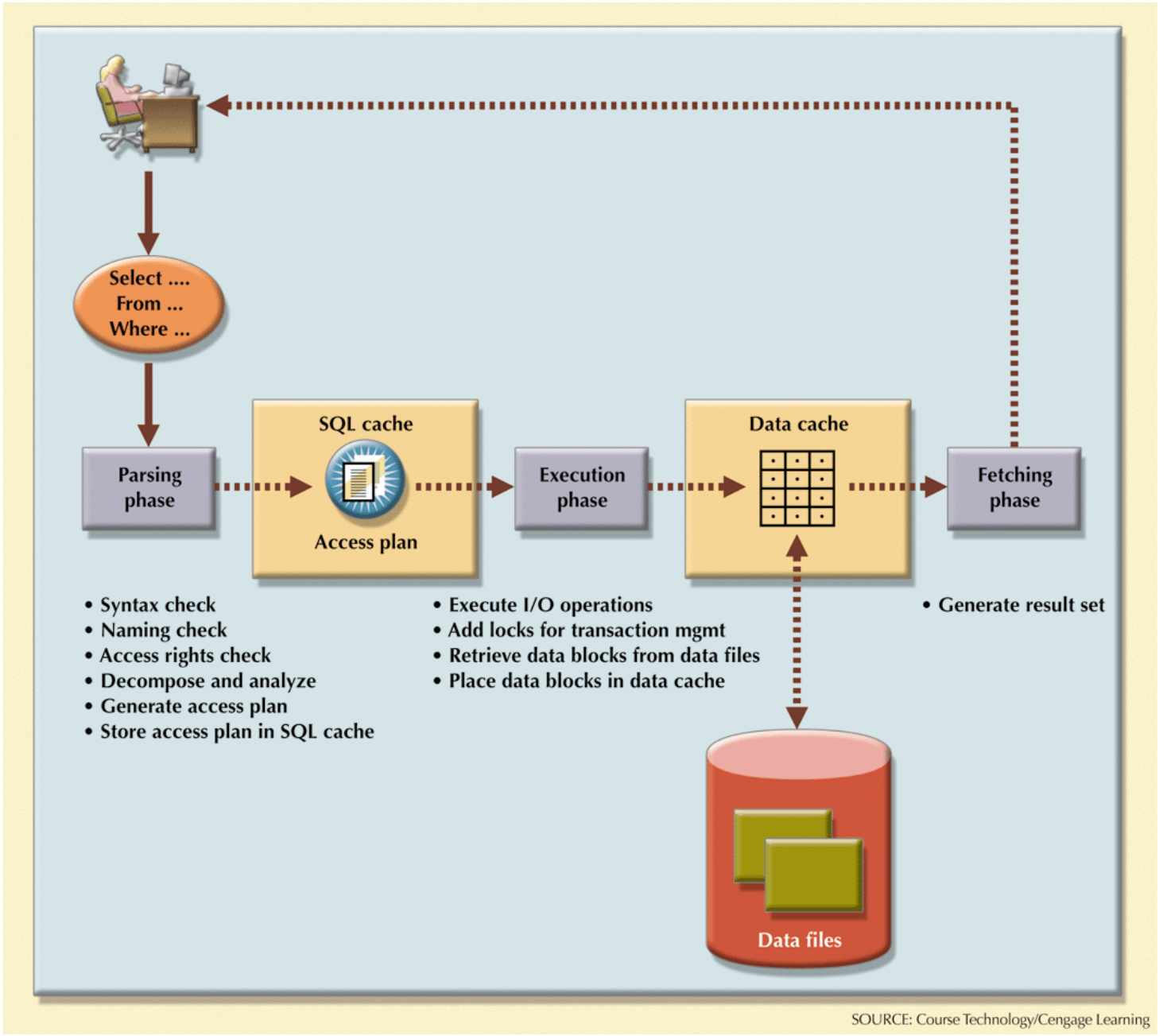
TABLE 11.2

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 Process

- DBMS processes queries in three phases
  - Parsing
    - DBMS parses the SQL query and chooses the most efficient access/execution plan
  - Execution
    - DBMS executes the SQL query using chosen execution plan
  - Fetching
    - DBMS fetches the data and sends the result back to the client



SOURCE: Course Technology/Cengage Learning

SQL parsing phase

- Break down query into smaller units
- Transform original SQL query into slightly different version of original SQL code
  - Fully equivalent
    - Optimized query results are always the same as original query
  - More efficient
    - Optimized query will almost always execute faster than original query

SQL parsing phase

- **Query optimizer:** Analyzes SQL query and finds most efficient way to access data
  - Validated for syntax compliance
  - Validated against data dictionary
    - Tables and column names are correct
    - User has proper access rights
  - Analyzed and decomposed into components
  - Optimized into a fully equivalent but more efficient SQL query
  - Prepared for execution by determining the most efficient execution

SQL parsing phase

- **Access plans**: Result of parsing a SQL statement
  - DBMS-Specific
  - Translate client's SQL query into a series of complex I/O operations
  - Required to read the data from the physical data files and generate result set
- DBMS checks if access plan already exists for query in SQL cache
- DBMS reuses the access plan to save time
- If it doesn't, the optimizer evaluates various plans
  - Chosen plan placed in SQL cache

TABLE 11.3

Sample DBMS Access Plan I/O Operations

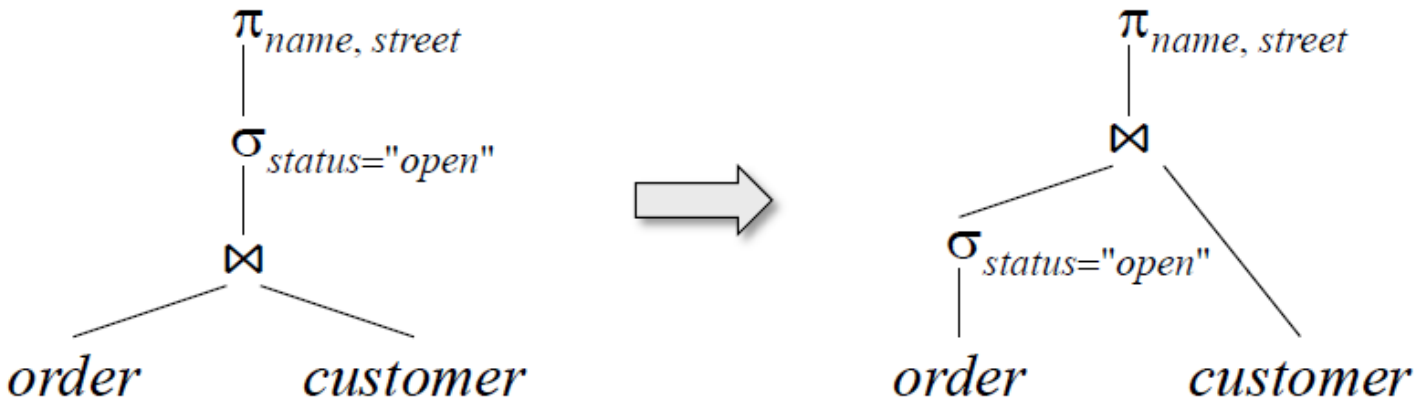
OPERATION	DESCRIPTION
Table Scan (Full)	Reads the entire table sequentially, from the first row to the last row, one row at a time (slowest)
Table Access (Row ID)	Reads a table row directly, using the row ID value (fastest)
Index Scan (Range)	Reads the index first to obtain the row IDs and then accesses 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	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 access plan are executed
  - Locks acquired
  - Data retrieved and placed in data cache
  - Transaction management commands processed

Query Optimization

- There are alternative ways for evaluating a given query
  - different equivalent expressions (query expression trees)



SQL Fetching Phase

- All rows that match the specified condition(s) are retrieved, sorted, grouped, and/or aggregated and results are returned to the client
- DBMS might use temporary table space to store temporary data

Query processing bottlenecks

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

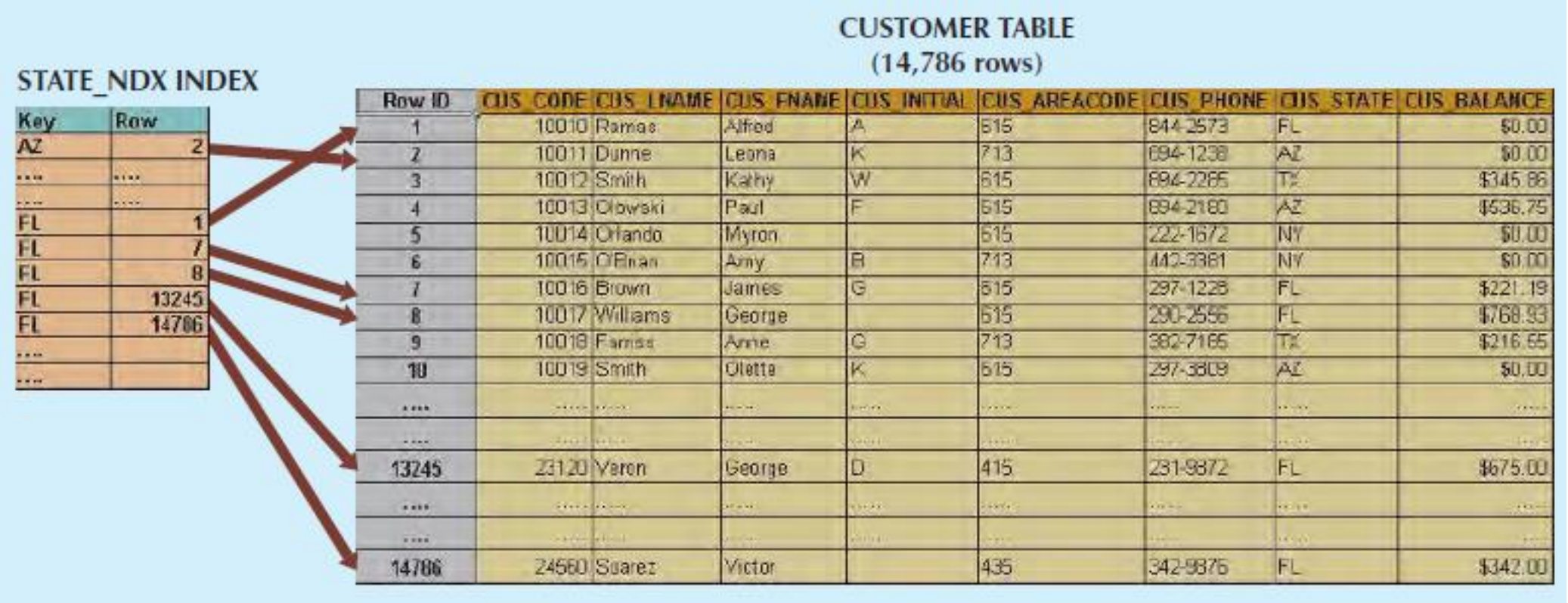
Indexes and Query Optimization

- Indexes
  - Crucial in speeding up data access
  - Facilitate searching, sorting, and using aggregate functions as well as join operations
  - Ordered set of values that contains index key and pointers
- More efficient to use index to access table than to scan all rows in table sequentially

Indexes and Query Optimization Cont.

Data sparsity: number of different values a column could possibly have

- Indexes implemented using:
- Hash indexes
  - good for simple and fast lookup operations based on equality conditions
- B-tree indexes
  - used mainly in tables in which column values repeat a relative smaller number of times.
- Bitmap indexes
  - used mostly in data warehouse applications in tables with a large number of rows in which a small number of column values repeat many times
- DBMSs determine best type of index to use



The CUSTOMER table has two columns that are used extensively for query purposes: CUS_LNAME, which represents a customer's last name, and REGION_CODE, which can have one of four values (NE, NW, SW, and, SE). Based on this information, you could conclude that:

-CUS_LNAME column contains many different values that repeat a relatively small number of times (compared to the total number of rows in the table), a B-tree index will be used.

-REGION_CODE column contains only a few different values that repeat a relatively large number of times (compared to the total number of rows in the table), a bitmap index will be used.

Example

Assume that you want to list all products provided by a vendor based in Florida. To acquire that information, you could write the following query:

```
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';
```

Furthermore, let's assume that the database statistics indicate that:

- 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.

TABLE 11.4

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**