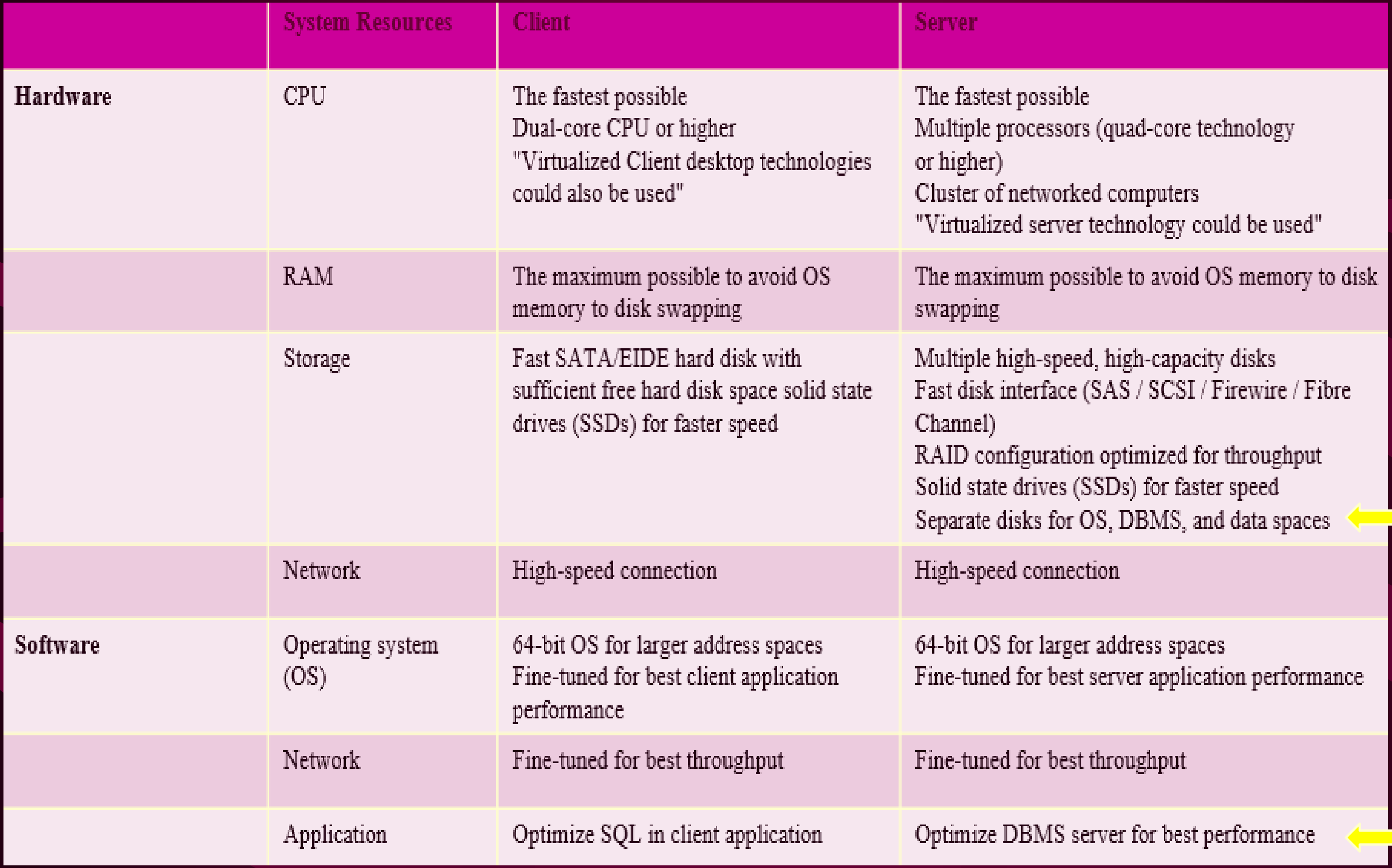
Performance Tuning and Optimization

Database Performance

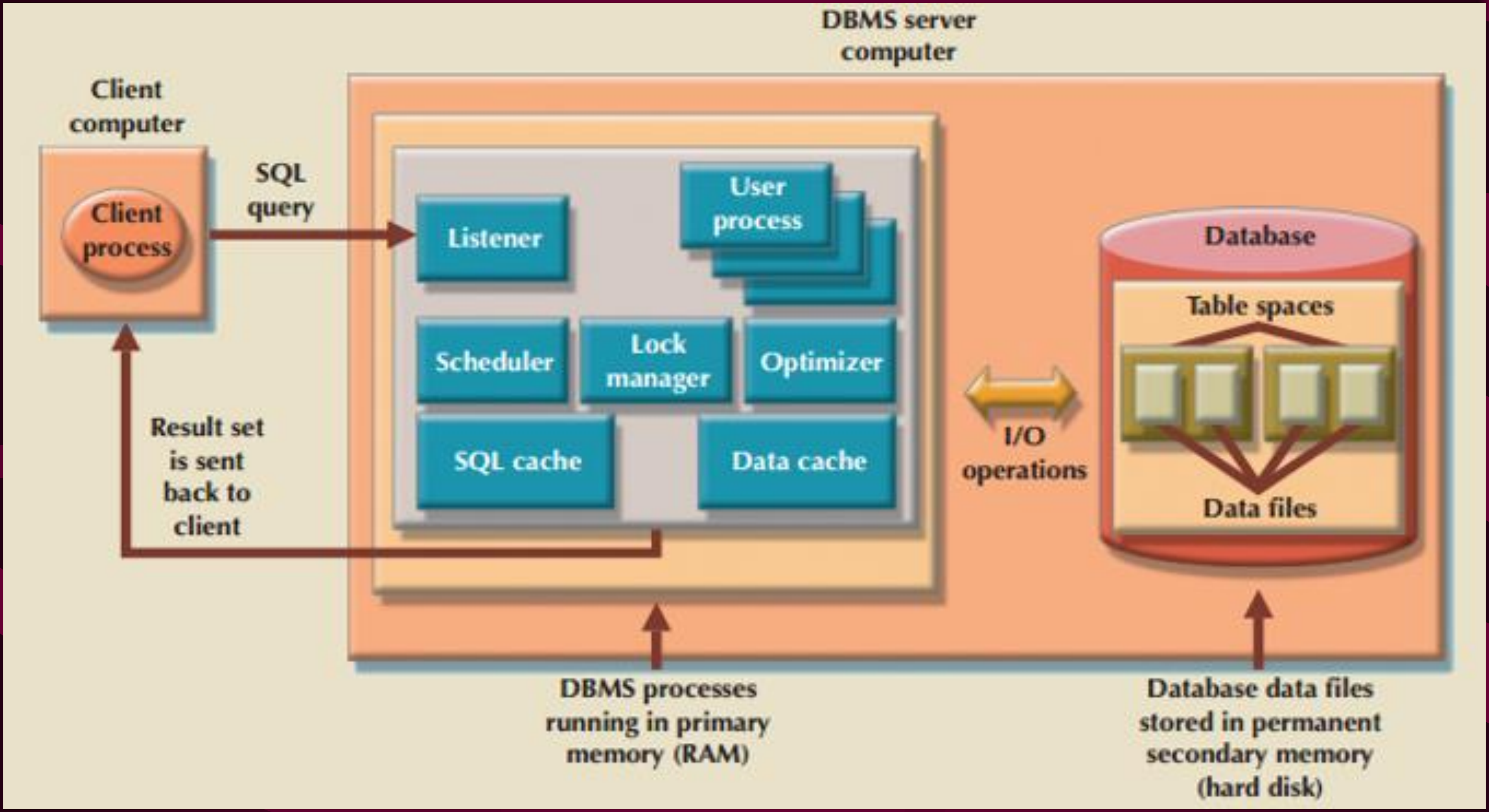


	System Resources	Client	Server
Hardware	CPU	The fastest possible Dual-core CPU or higher "Virtualized Client desktop technologies could also be used"	The fastest possible Multiple processors (quad-core technology or higher) Cluster of networked computers "Virtualized server technology could be used"
	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

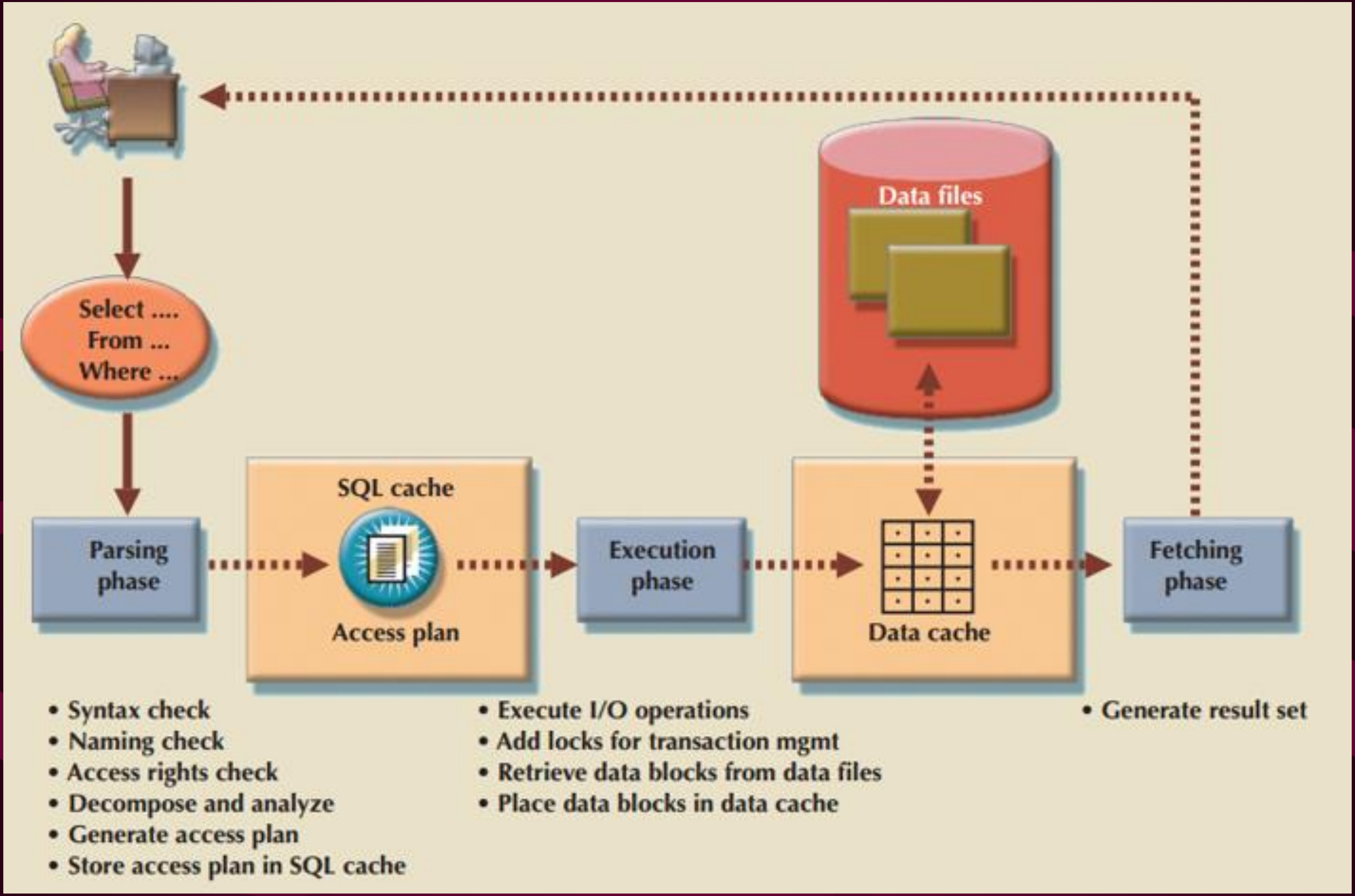
$\leftarrow$ Covered in DBA lesson

$\leftarrow$ Focus here

Typical RDBMS Environment



Query Processing



- Syntax check
- Naming check
- Access rights check
- Decompose and analyze
- Generate access plan
- Store access plan in SQL cache

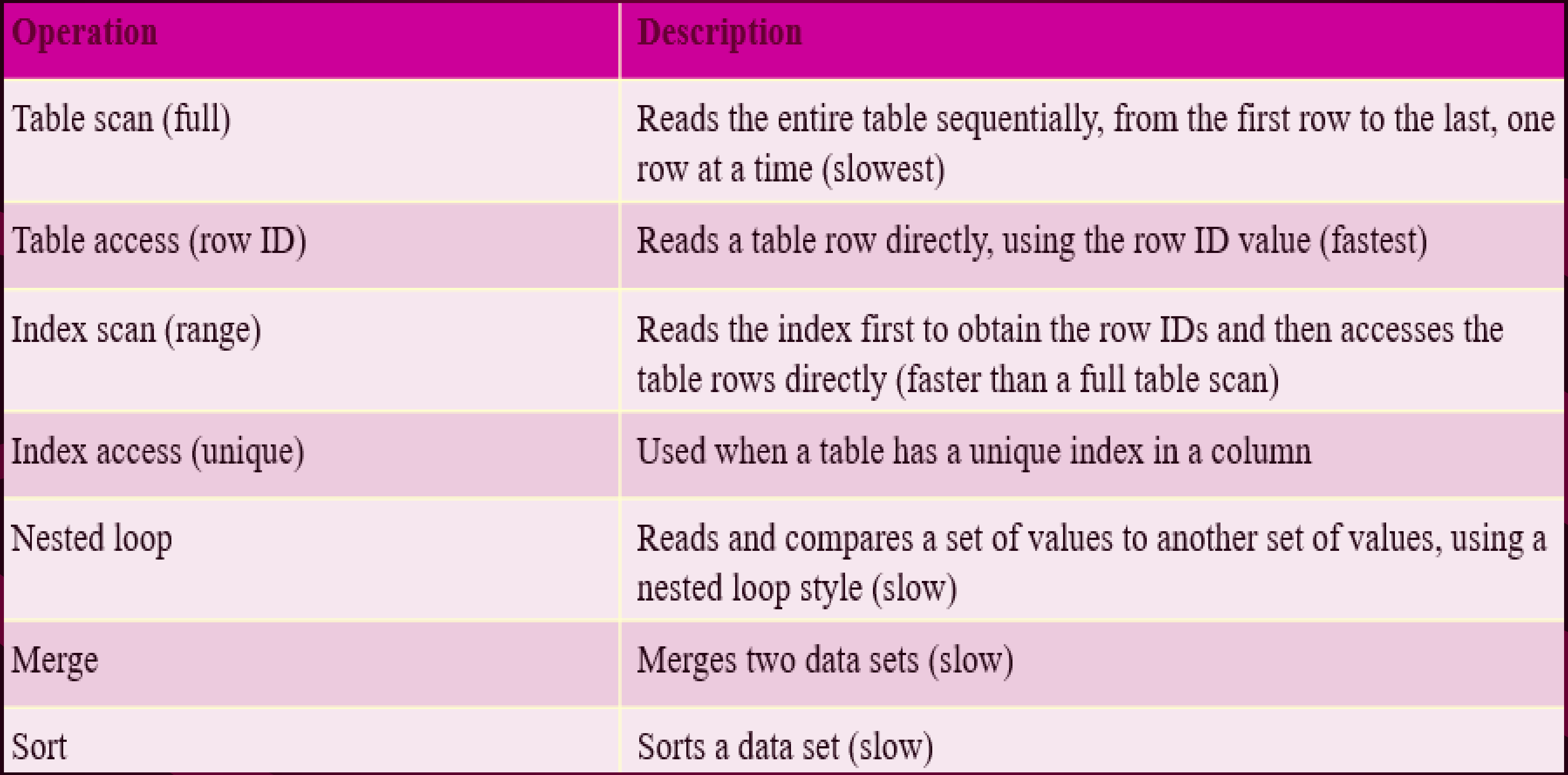
- Execute I/O operations
- Add locks for transaction mgmt
- Retrieve data blocks from data files
- Place data blocks in data cache

- Generate result set

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

SQL Parsing Phase (con't)



Operation	Description
Table scan (full)	Reads the entire table sequentially, from the first row to the last, 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 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

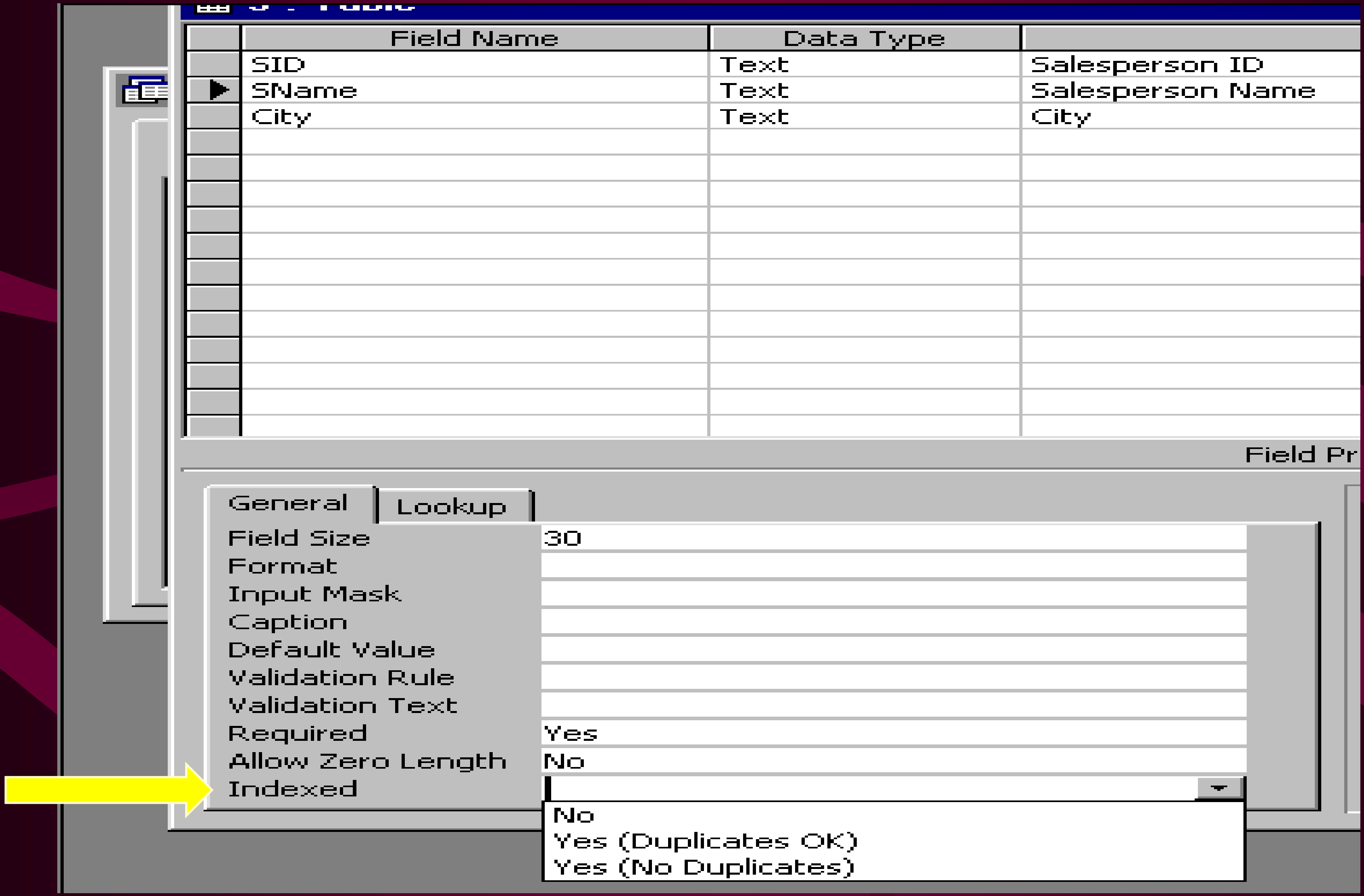
Query Processing Bottlenecks

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

INDEXES

- One can create indexes or unique indexes on tables; ascending (default) or descending
- Automatically created for primary keys (unique); and for most RDBMS, for foreign keys
- One or more columns, can be created and dropped dynamically
- Used for speed of access, sorting, & to force uniqueness
- Not referenced directly in SQL, but used by RDBMS
- CREATE INDEX CITYINDEX
  - ON S (City)

Indexes in Access



Field Name	Data Type	
SID	Text	Salesperson ID
SName	Text	Salesperson Name
City	Text	City

Field Pr

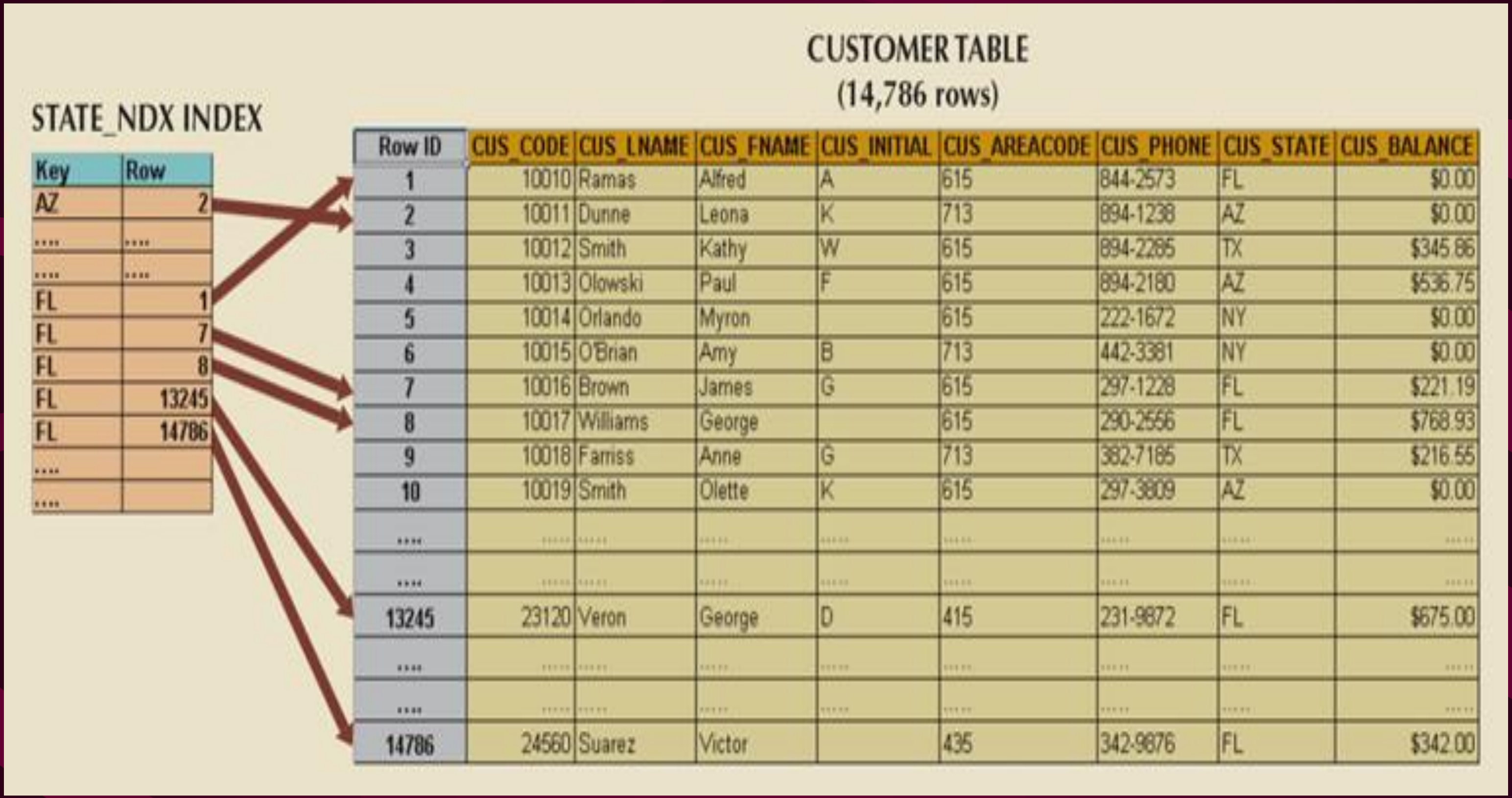
General | Lookup

Field Size	30
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	No
Indexed	

Indexes and 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 Optimization (con't)



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 Optimization (con’t)

- 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 may determine best type of index to use

Indexing Techniques

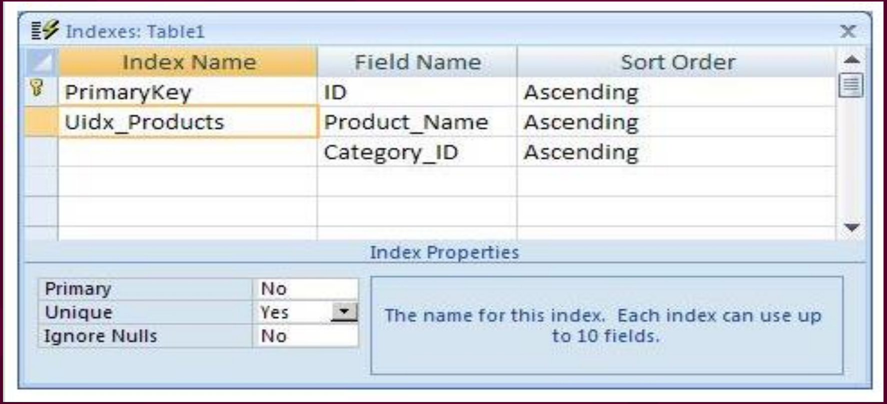
- Use Indexes on columns in “Where” clauses; do not compromise indexes with surrounding functions (ie UPPER(PName))
- Select *
  - FROM P
  - WHERE PName = “Shirt”;
- Place an index on PName to speed up such a search if it is common
- Use indexes sparingly, indexes do take up space (memory & disk), and also increase table add/update time

- Use concatenated indexes to speed up common searches where SELECT and WHERE clauses have reoccurring ties:
- In the customer table (ID, name, address, phone, ...):
  - SELECT name
  - FROM customer
  - WHERE phone between 750000 and 7599999;
- An index on the combination phone/name will not only speed up the search, but the table itself may not need to be accessed (all information is in the index file)

- In the customer table (ID, name, address, zip, areacode, exchange, ...):
  - SELECT name
  - FROM customer
  - WHERE areacode = 901 AND zip = 38138;
- An index on both WHERE conditions will speed up the search since the query optimizer should AND the indexes before going to the table to get the data:
  - Index1: areacode/exchange
  - Index2: zipcode

Access Indices on Multiple Fields (columns)

- It's a very useful practice that we sometimes need to store unique data across multiple columns in a table. One way of doing this is to create a primary key which contains these columns, but what if we already got a primary key (such as an AutoNumber field) in the table and we still want to keep and use it? The answer is to create a unique composite index on these columns. This type of index prevents duplicate values from being entered into the combination of these columns.
  - **Open the table in Design View**
  - **Click Indexes button**
  - **Enter the first column for the index (pick a unique index name) – note that the table primary key already shows up as an index**
  - **Enter the second column for the index (don't enter anything in the Index Name field (leave it blank))**
  - **Click the Index Name cell for the index; then, in the Index Properties section of the window, select Yes in the dropdown for the Unique property**



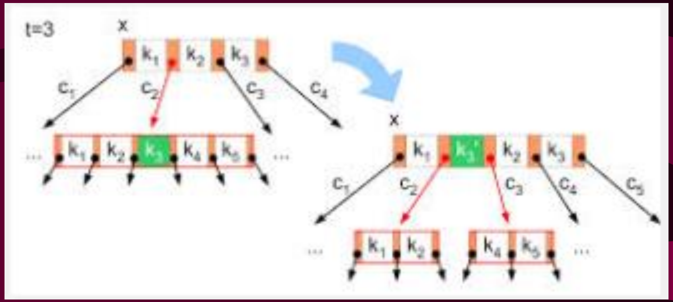
Index Name	Field Name	Sort Order
PrimaryKey	ID	Ascending
Uidx_Products	Product_Name	Ascending
	Category_ID	Ascending

Index Properties

Primary	No
Unique	Yes
Ignore Nulls	No

The name for this index. Each index can use up to 10 fields.

Index Types



- Normally B-tree indexes are used (balanced trees)
- Cluster indexes - keep commonly used data indexes together (i.e. customers and their orders)
- Bitmapped indexes - best for queries on multiple columns that have only a few distinct values (i.e. yes/no attributes)

Database Statistics

- Database statistics are measurements about database objects, such as the number of rows in a table, number of disk blocks used, maximum and average row length, number of columns in each row, and number of distinct values in each column
- Such statistics provide a snapshot of database characteristics

Database Statistics (con’t)

- Database statistics can be gathered manually by the DBA or automatically by the DBMS
- Many DMBS vendors support the ANALYZE command in SQL to gather statistics
- In addition, many vendors have their own routines to gather statistics

Database Statistics (con't)

- Database statistics typically include:
  - Tables: Number of rows, number of disk blocks used, row length, number of columns in each row, etc.
  - 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, etc.
  - Environment Resources: Logical and physical disk block size, location and size of data files, and number of extends per data file

Query Optimization

- Most modern RDBMS have "query optimizers"
- Some are static and some are dynamic (look not only at the SQL but the sizes of the tables involved)
- Static optimizers are often called "rule based" and use a set of rules (such as it is better to use and index that to read the entire table)
- Dynamic optimizers are often called "cost based" and use statistics (such as the number of rows and the number of distinct values in index's) to determine the cost of the query in computer resources
  - Statistics are often gathered at planned intervals and may be "samples"

Query Optimization (con’t)

- 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

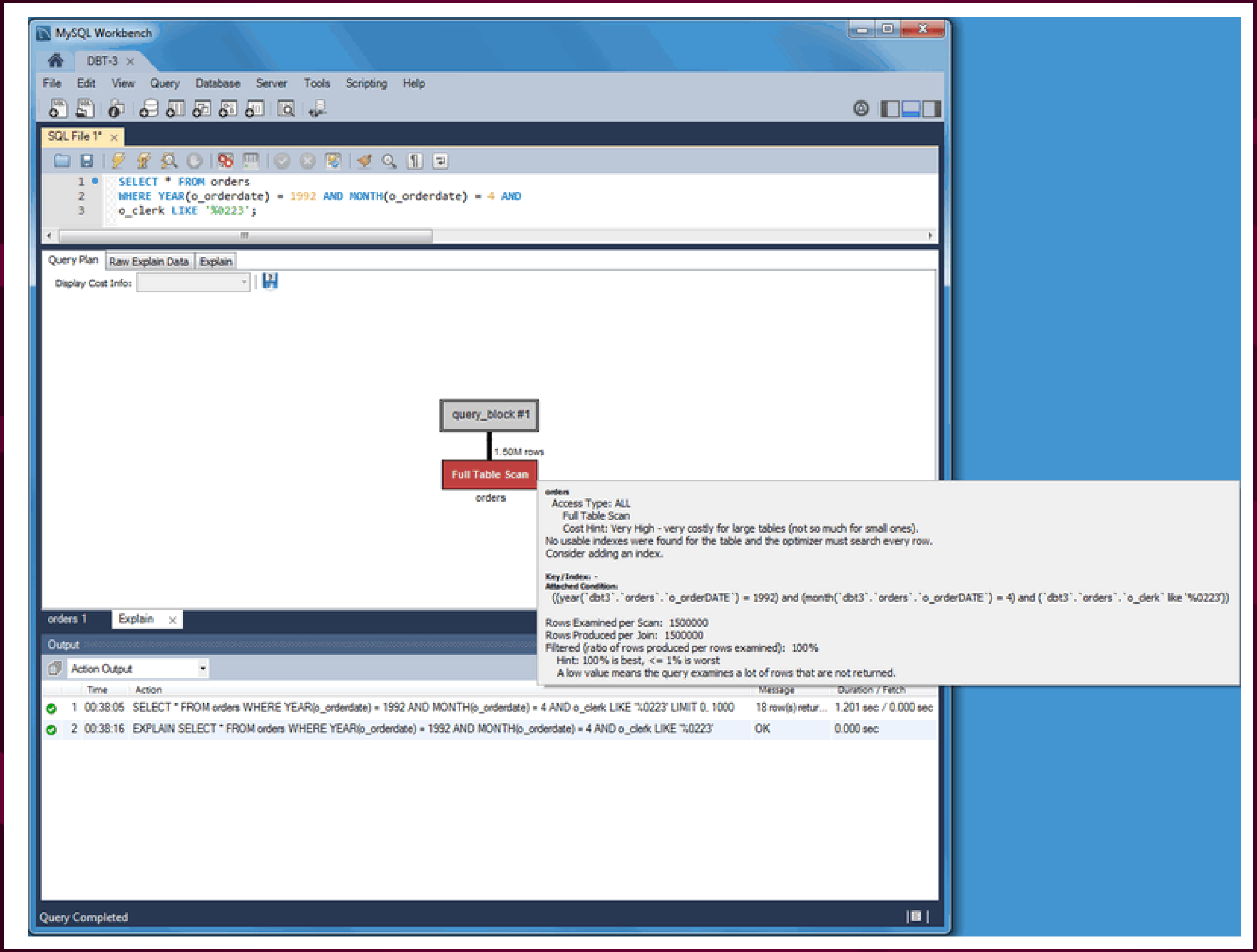
Statistics Used by Optimizers

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 Optimization (con’t)

- The RDBMS have a reporting tool (“Explain” or “Trace”) that will show you what is going on inside of a query and a relative “cost”
- However, for the best results you should properly structure your table design, proper use of indexes, physical media layout, and try SQL alternatives

MySQL Workbench Visual Explain



```sql
SELECT * FROM orders
WHERE YEAR(o_orderdate) = 1992 AND MONTH(o_orderdate) = 4 AND
o_clerk LIKE '%0223';
```

```
orders
  Access Type: ALL
     Full Table Scan
     Cost Hint: Very High - very costly for large tables (not so much for small ones).
No usable indexes were found for the table and the optimizer must search every row.
Consider adding an index.

Key/Index: -
Attached Condition:
  ((year(`dbt3`.`orders`.`o_orderDATE`) = 1992) and (month(`dbt3`.`orders`.`o_orderDATE`) = 4) and (`dbt3`.`orders`.`o_clerk` like '%0223'))

Rows Examined per Scan:  1500000
Rows Produced per Join:  1500000
Filtered (ratio of rows produced per rows examined):  100%
   Hint: 100% is best, <= 1% is worst
   A low value means the query examines a lot of rows that are not returned.
```

		Time	Action	Message	Duration / Fetch
✓	1	00:38:05	SELECT * FROM orders WHERE YEAR(o_orderdate) = 1992 AND MONTH(o_orderdate) = 4 AND o_clerk LIKE '%0223' LIMIT 0, 1000	18 row(s) retur...	1.201 sec / 0.000 sec
✓	2	00:38:16	EXPLAIN SELECT * FROM orders WHERE YEAR(o_orderdate) = 1992 AND MONTH(o_orderdate) = 4 AND o_clerk LIKE '%0223'	OK	0.000 sec

SQL Optimizer Advise (“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`

Optimization Considerations

- 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

Optimization Considerations (con’t)

- Do selections before joins
- Denormalization where appropriate (use triggers to keep database consistent)
- Experiment with different SQL methods
- Use Union, Intersection, Difference (where available in RDBMS)
- Distributed database queries - covered later !

Optimization Considerations (con't)

- Guidelines to write efficient conditional expressions in SQL code
  - Use simple columns or literals as operands
  - Numeric field comparisons are faster than character, date, and NULL comparisons
  - Equality comparisons are faster than inequality comparisons
  - Transform conditional expressions to use literals
  - Write equality conditions first when using multiple conditional expressions
  - 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

- SELECT customer_ID
  - FROM customers
  - WHERE area_code IN (901)
  - AND zip_code NOT IN (38138, 38139);
- Replace the above with a difference (even if both area_code and zip_code have indexes, the NOT IN will cause an entire table search):
- SELECT customer_ID
  - FROM customers
  - WHERE area_code IN (901)
  - MINUS
    - (SELECT customer_ID
      - FROM customers
      - WHERE zip_code IN (38138, 38139);

Limiting Queries

- SELECT *
  - FROM customers
  - WHERE last_name like ‘S%’ and ROWNUM < 1000
  - ORDER BY last_name;

- Will limit result set to 1000, before any sorting
- Some RDBMS will allow you to set overall or by user query limits (on # of rows or time)

"Real World Queries" [combinations of joins and nested subqueries]

```
SELECT   p.adrno,
         p.adradd,
         p.adrdir,
         p.adrstr,
         p.adrsuf,
         p.nbhd_p,
         p.nbhd_s,
         s.saledt as saledate,
         s.price as saleprice,
         a.rtotapr,
         d.source,
         d.sub_source,
         d.source_date,
         d.stories,
         d.fixhalf,
         d.extwall,
         d.cond,
         d.heatsys,
         d.fuel,
         d.style,
         p.class,
         p.zoning,
         d.sfla,
         d.rmtot,
         d.rmbed,
         d.fixbath,
         d.yrblt,
         l.acres
FROM     par p,
         dwel d,
         asm a,
         leg l,
         sales s
WHERE    p.nbhd_p = '#form.nbhd_p#' and
         p.nbhd_s = '#form.nbhd_s#' and
         p.parid = d.parid and
         d.parid = a.parid and
         a.parid = l.parid and
         l.parid = s.parid and
         p.adrno > 0 and
         s.saledt Between #date1# and #date2# and
         s.price > 0.5 * #assessor_amt# and
         d.source in ('1','2') and
         p.class = 'R' and
         d.sfla Between #nMin# And #nMax# and
         d.rmtot > 0 and
         d.rmbed > 0 and
         d.fixbath > 0 and
         d.yrblt > 0 and
         l.acres > 0 and
         (s.numpars is null or s.numpars < 2) and
         d.source = (select max(a.source)
                         from dwel a
                         where a.parid = p.parid) and
                         d.source_date = (select max(b.source_date)
                                              from dwel b
                                              where b.parid = d.parid and
                                              b.source = d.source)
```

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
    - Larger memories and diminishing costs
    - Technology advances of components

DBMS Performance Tuning (con't)

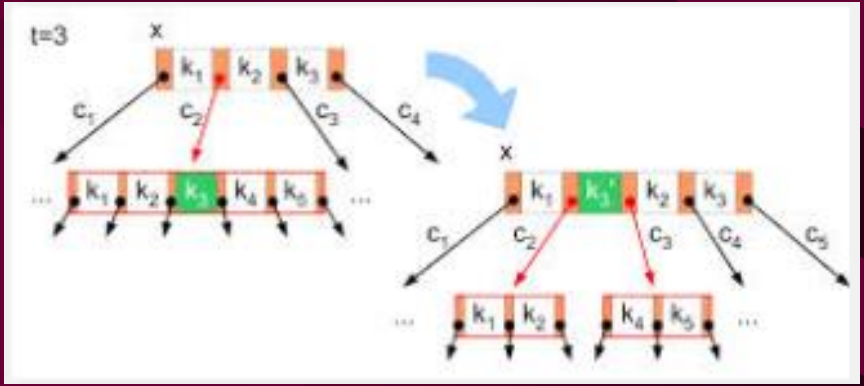
- Recommendations for physical storage of databases
  - Utilize I/O accelerators
  - Use RAID (Redundant Array of Independent Disks) to provide a balance between performance improvement and fault tolerance
  - Minimize disk contention
  - Physical allocation to drives (see later database administration lesson)
    - 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

References

- Oracle Database Performance Tuning Tips & Techniques (Oracle Press) by Richard Niemiec
- MySQL Query Performance Tuning: A Systematic Method for Improving Execution Speeds by Jesper Wisborg Krogh
- Microsoft SQL Server Query Tuning & Optimization by Benjamin Nevarez

Homework

- Textbook Chapter 11
- Review Questions 1 thru 10


t=3
x
k1 k2 k3
c1 c2 c3 c4
k1 k2 k3 k4 k5
x
k1 k3' k2 k3
c1 c2 c3 c4 c5
k1 k2
k4 k5