http://freetodev.spaces.live.com

Microsoft SQL Server 2005 Query DeskSheet

Dynamic Management Views & Functions (prefix sys.dm_)

CLR
clr_appdomains
clr_loaded_assemblies
clr_properties
clr_tasks

Database Mirroring
db_mirroring_connections

Database
db_file_space_usage
db_session_space_usage
db_partition_stats
db_task_space_usage

Execution and Functions
exec_background_job_queue
exec_background_job_queue_stats
exec_cached_plans
exec_connections
exec_cursors
exec_plan_attributes
exec_query_optimizer_info
exec_query_plan
exec_query_stats
exec_requests
exec_sessions exec_sql_text

Full-Text Search
fts_active_catalogs
fts_crawls
fts_crawl_ranges
fts_memory_buffers
fts_memory_pools

Related Dynamic Management Views
db_index_operational_stats
db_index_physical_stats
db_index_usage_stats

I/O and Functions
io_backup_tapes
io_cluster_shared_drives
io_pending_io_requests
io_virtual_file_stats

Query Notifications
qn_subscriptions

Replication
repl_articles
repl_schemas
repl_tranhash
repl_traninfo

Service Broker
broker_activated_tasks
broker_connections
broker_forwarded_messages
broker_queue_monitors

SQL Operating System
os_buffer_descriptors
os_memory_pools
os_child_instances
os_performance_counters
os_cluster_nodes
os_schedulers
os_hosts
os_stacks
os_latch_stats
os_sys_info
os_loaded_modules
os_tasks
os_memory_cache_clock_hands
os_threads
os_memory_cache_counters
os_virtual_address_dump
os_memory_cache_entries
os_wait_stats
os_memory_cache_hash_tables
os_waiting_tasks
os_memory_clerks
os_workers
os_memory_objects

Transaction and Functions
tran_active_snapshot_database_transactions
tran_active_transactions
tran_current_snapshot
tran_current_transaction
tran_database_transactions
tran_locks
tran_session_transactions
tran_top_version_generators
tran_transactions_snapshot
tran_version_store

Index Performance

[**Slowest**] No Index -> Non-clustered non-covering -> Clustered -> Non-clustered covering -> Non-clustered covering with included non-key columns [**Fastest**]

Join Performance

[**Slowest**] Hash -> Loop -> Merge [**Fastest**]

Sample Data

```
SELECT TOP (100) *
FROM   aTable TABLESAMPLE
(500 ROWS | PERCENT)
REPEATABLE(9);
```

SQL Injection

Reject: ; ' -- /* ... */ xp_
Reject for Filename: AUX, CLOCK$, COM1 - COM8, CON, CONFIG$, LPT1 - LPT8, NUL, PRN.
Search for vulnerabilities:

```
SELECT object_Name(id)
FROM   syscomments
WHERE  UPPER(text) LIKE '%EXECUTE (%'
  OR UPPER(text) LIKE '%EXEC (%'
  OR UPPER(text) LIKE '%SP_EXECUTESQL%'
```

Non-clustered Index INCLUDE

```
CREATE INDEX idx_ABC
ON aTable (KeyId1, KeyId1)
INCLUDE (NonKeyColumn(s));
```

Metrics

```
SET STATISTICS IO | TIME |
PROFILE | XML ON
```

Ranking Functions

```
SELECT [list]
    ,ROW_NUMBER() OVER(ORDER BY column(s)) -- May need tiebreaker
    ,RANK() OVER(ORDER BY column(s))
    ,DENSE_RANK() OVER(ORDER BY column(s))
FROM aTable
```

Recursive Common Table Expressions

```
WITH cte_name ( column_name [,...n] )
AS
(
  CTE_query_definition -- Anchor member is defined.
  UNION ALL
  CTE_query_definition -- Recursive member is defined referencing
cte_name (1).
)
SELECT * FROM cte_name -- (2). Note cte_name in (1) & (2) are different
OPTION (MAXRECURSION 20);
```

Smallest Missing Numeric KeyId

```
SELECT CASE WHEN
    NOT EXISTS(SELECT * FROM
aTable
          WHERE  KeyId = 1)
    THEN 1
    ELSE (SELECT MIN(A.KeyId + 1)
       FROM   aTable AS A
       WHERE  NOT EXISTS
          (SELECT * FROM aTable AS B
           WHERE  B.KeyId = A.KeyId +
1))
END;
```

Logical Processing Phases

```
(8) SELECT (9) DISTINCT (11) TOP
[list]
(1) FROM   [LeftTable]
(3) [join type] JOIN   [RightTable]
(2)   ON [condition]
(4) WHERE [condition]
(5) GROUP BY [list]
(6) WITH [CUBE | ROLLUP]
(7) HAVING [condition]
(10)ORDER BY [list]
```

HOT KEYS

CTRL+L = Estimated Execution Plan
CTRL+M = Actual Execution Plan

EXCEPT

```
SELECT a, b, c FROM aTable
EXCEPT  -- all rows appearing in aTable
but not bTable
SELECT a, b, c FROM bTable
```

INTERSECT

```
SELECT a, b, c FROM aTable
INTERSECT   -- all rows appearing
in both aTable and bTable
SELECT a, b, c FROM bTable
```

PIVOT

```
SELECT *
FROM   (SELECT id, YEAR(rowdate) AS rowyear, qty
    FROM aTable) AS D
PIVOT  (SUM(qty) FOR rowyear IN([2001],[2002],[2003])) AS P;

SELECT *
FROM   (SELECT id, YEAR(rowdate) AS rowyear
    FROM aTable) AS D
PIVOT  (COUNT(rowyear) FOR rowyear IN([2001],[2002],[2003])) AS P;
GO
```

TOP

```
SELECT TOP(scalar expression | variable
| subquery) [PERCENT] [WITH TIES] *
FROM   aTable
```

OVER()

```
--This will delete duplicates
WITH Duplicates AS
(
 SELECT *,
  ROW_NUMBER()
OVER(PARTITION BY KeyId
ORDER BY KeyId) AS rowNumber
 FROM DuplicatesTable
)
DELETE FROM Duplicates
WHERE rowNumber > 1;
GO
```

Remember: SQL optimizes AND logic better than OR logic