Microsoft SQL Server 2000

개정판

Microsoft®

SQL Server

SQL Server 성능 향상을 위한 튜닝 가이드

Microsoft®

본 문서에 포함된 정보는 출판 당시 논의된 문제에 대한 마이크로소프트 사의 최신 견해를 나타내는것입니다. 마이크로소프트 사는 변화하는 시장 조건에 대응해야 하기 때문에, 이를 마이크로소프트 사 측의 약속으로 해석해서는 안되며 마이크로소프트는 출판 후 제공되는 모든 정보에 대한 정확성을 보증하지 않습니다.

본 문서는 정보 제공 목적으로만 제공됩니다. 마이크로소프트는 본 문서에서 명시적이고 암시적인 보증을 하지 않습니다.

SQL Server 성능 향상을 위한 튜닝 가이드

저자 약력

차주언 / SQL Server 선임 컨설턴트

- ㈜ 웹타임 전임 강사
- 한화 에쓰엔씨 개발팀
- KM(지식경영),그룹웨어 개발 & 컨설팅
- 삼성화재 애니카 서비스 DB 관리 컨설팅
- 저서
  → 윈도우 2000 웹 서버 보안
  → 현재 SQL 서버 보안과 튜닝 집필중

감수자 약력

차재호 / ㈜ 에이지소프트 대표이사

- 울산대학교 시스템공학부 E-Learning 시스템 개발
- 울산시청 웹메일 시스템 컨설팅 및 개발
- 현대중공업 해양산업부 기술연구소 용접 장비 실시간 모니터링 시스템개발
- 현대정공 갤로퍼 생산라인 관리 시스템 개발
- HMC 통합번제시스템 개발& HMC 완성차 생산실적관리 시스템 개발
- 삼성전자 해외영업부문 닷넷 컨설팅

정순용 / SQL Server 선임 컨설턴트

- ㈜ 웹타임 전임 강사
- ㈜ 미원상사 전산실 근무
- 중소기업 전산담당자 모임(전담모) 운영자 및 컨설턴트
- ㈜ 미원상사 인트라넷 자체 데이터베이스 솔루션 개발(4GL)
- 온라인교육(스터디피아) Q&A 및 업체 필드지원

저자의 글

쉬운 것은 그대로 쉽게, 어려운 것도 쉽게!

이러한 마음가짐으로 책을 집필했습니다. 아무쪼록 저의 이러한 목표가 여러분이 그 동안 어렵다고 여겨져 왔던 SQL 튜닝을 쉽게 이해하는데 도움이 되었으면 합니다.

백번 듣는 것보다 한번 보는 것이 나으며, 백번 보는 것보다 한번 행하는 것이 낫다!

최대한 쉬운 예제로 더 이상의 설명이 사족이 되는 것 같은 느낌의 간결한 예제를 사용했습니다. 전부 따라 해봅시다.

저자: 차 주 언 SQL Server 컨설턴트

감수의 글

이전에 SQL Server 튜닝에 관련된 책은 상당한 두께를 가지고 있었습니다.
그래서 인지 SQL Server 튜닝은 마치 고급 SQL 개발자만의 영역인 것처럼 느껴졌습니다.

그러나, 현실에서는 SQL 고급 개발자들에게 튜닝을 맡길 수 없는 환경이 대부분이며,
그저 응용프로그램 개발자들의 단편적인 튜닝에 관한 지식에 의존하였습니다. 이번 포켓 튜닝 가이드는 이러한 환경에서 응용프로그램 개발자들에게 SQL Server 튜닝의 모든 것을 제공해 줄 수는 없지만, SQL Server 튜닝에 관한 가이드를 충분히 제공하리라 믿습니다.

고급 SQL Server 튜닝을 적용할 수 없었던 많은 환경에서 일반 응용프로그램 개발자들이 늘 옆에 두고 가이드를 받을 수 있으리라 봅니다.

감수: 차재호 에이지소프트 대표이사

기획자의 글

IT 엔지니어의 양적인 팽창과 인터넷의 급속한 성장으로 특정 솔루션 튜닝 기법과 실무 사례들은 주위에서 쉽게 구할 수 있습니다. 그럼에도 불구하고 끊임없이 성능상의 문제가 발생하는 이유는 무엇인가? 그리고 기본적인 튜닝지식 있는데도, 실제 튜닝을 할수 없었던 이유들을 본 SQL 튜닝 포켓 가이드를 통해 체계적으로 해결할 수 있는 지침서가 되었으면 합니다.

본 포켓 가이드는 핵심부분만 요약한 지면의 한계로 튜닝에 대한 고급적인 설명까지는 언급되지 않고 있지만, Vmware or Virtual Server 를 통해 꼭 실습해 보시기 바랍니다. 그리고 한정된 지식만 전달하는 것 같아 걱정 되는 부분이 있지만, SQL 튜닝 포켓 가이드에서 제시하는 내용의 숙지를 통해, 더 깊이 있는 튜닝 전문가로 첫발을 내딛는데, 조금이라도 도움이 됐으면 합니다.

기획: 안덕중 ㈜웹타임 & MCPworld

Microsoft®

SQL Server

SQL Server 성능 향상을 위한 튜닝 가이드

본 포켓 튜닝 가이드는 경험 있는 SQL 관리자와 초보자 모두에게 도움이 되도록 기초부터 고급까지 튜닝에 꼭 필요한 내용을 최대한 쉬운 예제를 통해 기술했습니다.
예제는 쉽더라도 그 내용은 무척 중요하므로 꼭 따라 하면서 본 포켓 튜닝 가이드를 실습하여 봅시다.

Table of Contents

SQL Server for Developer : 개발자를 위한 튜닝 가이드

SQL Server for DBA : 관리자를 위한 튜닝 가이드

본 포켓 튜닝 가이드는 경험 있는 SQL관리자와 초보자 모두에게 도움이 되도록 기초부터 고급까지 튜닝에 꼭 필요한 내용을 최대한 쉬운 예제를 통해 기술했습니다. 예제는 쉽더라도 그 내용은 무척 중요하므로 꼭 따라 하면서 본 포켓 튜닝 가이드를 실습하여 봅시다.

SQL Server for Developer
개발자를 위한 튜닝 가이드

1. 쿼리 디자인

번호	수칙	체크
01	SELECT는 필요한 결과값만을 요구 하는가?	☐
02	적절한 WHERE 조건을 사용하는가?	☐
03	COUNT(컬럼명) 대신 COUNT(*)을 사용하는가?	☐
04	커서 및 임시 테이블의 내용을 최대한 자제하는가?	☐
05	VIEW의 총 사용을 줄였는가?	☐
06	저장 프로시저를 사용하는가?	☐
07	저장 프로시저를 적절하게 리컴 파일 하며 사용하는가?	☐
08	작명 된 프로시저 SP외의 접두어를 사용하는가?	☐
09	모든 개체의 소유자는 DBO로 지정하며 생성하는가?	☐
10	데드락이 발생하는 부분을 라이브락 형태로 변경했는가?	☐
11	SET NOCOUNT ON을 사용하는가?	☐
12	실무 사례 : 저장 프로시저 관리 방법	☐

스티브 맥코넬이 이런말을 했습니다.

뛰어난 디자이너는 습득한 지식을 사용하지 않는것과 그 지식을 처음부터 확보하지 못한 것을 동일하게 봅니다.

이말뜻을 다음과 같이 해석하고 싶습니다 여러분은 쿼리 분석기의 기능들이 어떤 것이 있고 단축키가 메뉴 우측에 작게 표시되어 있다는 것을 대부분 알고 있습니다. 하지만 잘 사용하지는 않고 있을 것입니다. 라고 말입니다. 그래서 먼저 단축키와 그 사용법에 대해 안내하는 시간을 우선 가지도록 하겠습니다.

다음을 실습해보고 자세한 것은 표를 참조합시다

```
-- CTRL + E , F5
-- 실행하기

use pubs
go

select * from titles

-- CTRL + T =〉 결과 Text로 보기

select * from titles

-- CTRL + D =〉 결과 Text로 보기

select * from titles

-- CTRL + K =〉 실행계획 보기

select * from titles

-- F8 =〉 개체브라우저 보이기/감추기

-- CTRL + R =〉 결과창 보이기/감추기

-- 그외 CTRL + C , CTRL + V , CTRL + X

-- CTRL + SHIFT + C =〉 주석달기

select * from titles
```

다음의 표를 참조합시다.

	없음	Shift+	Ctrl+	Alt+	Shift+Ctrl+
A			전체 선택		
B			중간 구분선 선택		
C			복사		주석 달기
D			표형태로 결과 표시	데이터베이스 선택	
E			실행		
F			찾기		파일로 결과 저장
G					
H			교체		
I			인덱스 튜닝마법사		
J					
K			실행 계획 보기		
L			예상 실행 계획 보기		선택 내용을 소문자로
M					
N			새 쿼리 윈도우		
O			연결		
P					
Q					
R			결과창 보이기/감추기		주석제거
S			저장		
T			텍스트로 결과 표시		
U					선택 내용을 대문자로
V			붙여넣기		
W					
X			자르기		
Y			다시하기		
Z			취소		
F1	도움말 선택 내용을 도움말로 보기				
	객체 브라우저보이기				
F8	감추기				

주요 단축키 사용 안내입니다.

수칙1. SELECT는 필요한 결과값만을 요구하는가?

불필요한 데이터가 시스템의 자원을 사용하는 것을 막아야 합니다. 다음의 쿼리 3개를 비교해 보면 어느 정도의 시스템 자원이 불필요 하게 소모 되고 있는지를 알 수 있습니다.

```
select title , price from titles
where title_id = 'BU1032'
```

Select하는 내용도 필요한 항목만을 가지고 오도록 되어 있어서 리소스가 전혀 낭비되지 않고 있습니다.

```
select title , price from titles
```

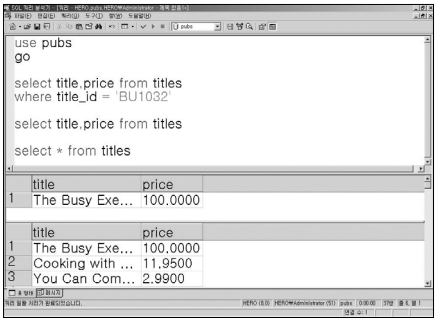
필요한 칼럼을 가져오기는 하지만 불필요한 전체 행(Row)들을 가져오고 있습니다.

```
select * from titles
```

불필요한 칼럼 정보, 행(Row) 데이터를 가져오고 있습니다.

[따라하기 - 3개의 쿼리를 한번에 실행하기]

1. 3개의 쿼리를 한 Session에서 실행하여 결과 3개를 동시에 살펴봅니다.



[01-01]

2. 결과 값으로 출력되는 데이터량의 차이를 확인합니다. 어느 쿼리가 가장 간결한 결과를 반환합니까?
   (반드시 꼭 필요한 결과만 반환하게 하는 것이 좋습니다.
   select title , price from titles where title_id = 'BU1032' 가 적절합니다.)

수칙2. 적절한 WHERE 조건을 사용하는가?

인덱스란 데이터를 빨리 찾기 위해서 사용됩니다. 인덱스가 없다면 특정 데이터를 찾기 위해서 모든 데이터 페이지를 검색(Table Scan)해야만 합니다 그에 비해 인덱스가 존재하고 그 인덱스가 사용되는 것이 효과적이라면 SQL 서버는 해당 인덱스 페이지를 사용하여 쉽게 데이터를 가져올 수 있는데 이를 인덱스 검색(Index Seek)이라 합니다.

그러나 이렇게 인덱스가 있더라도 이를 사용 불가능하게 하는 나쁜 쿼리가 있으니 이는 검색 조건에서 불필요하게 칼럼이 변형된 경우입니다. 다음의 여러 나쁜 예를 좋은 예와 비교해 봅시다.

SARG(Search Argument)란 쿼리가 반환하는 결과를 제한하기 위하여 옵티마이저가 인덱스와 결합해서 사용할 수 있는 쿼리내의 조건절을 말하는데 다음의 형태를 가집니다.

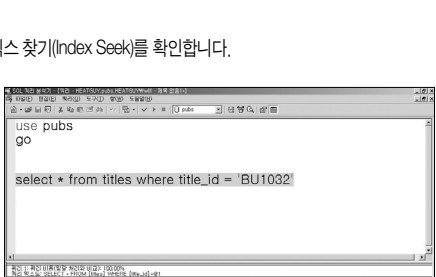
컬럼 연산자/변수

옵티마이저가 쓸모있게 변환하는 것은 CTRL+K 실행 계획 상부 표시에서 관찰할 수 있습니다.

```
set showplan_all on

select * from authors
where au_lname like 'Ma%'

-- OBJECT:([pubs].[dbo].[authors].[aunmind]), SEEK:([authors].[au_lname] >=
'Ma' AND [authors].[au_lname] < 'MB'),  WHERE:(like([authors].[au_lname],
'Ma%', NULL)) ORDERED FORWARD

-- set showplan_all off
```

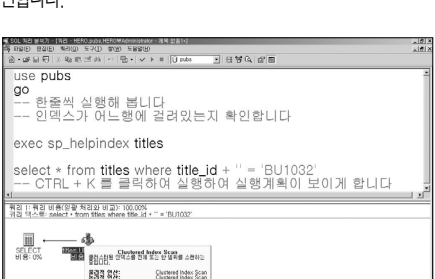
[따라하기]

1. 인덱스 찾기(Index Seek)를 확인합니다.



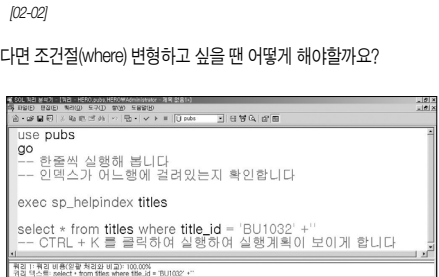
[02-01]

2. 다음과 같이 약간의 조건절(where) 변형만으로 인덱스 페이지가 사용되지 않음을 확인합니다.



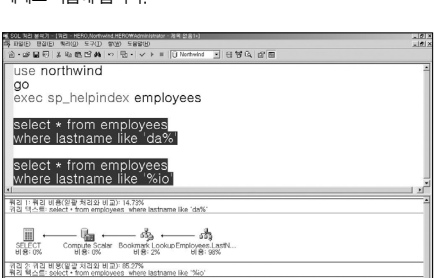
[02-02]

3. 그렇다면 조건절(where) 변형하고 싶을 땐 어떻게 해야할까요?

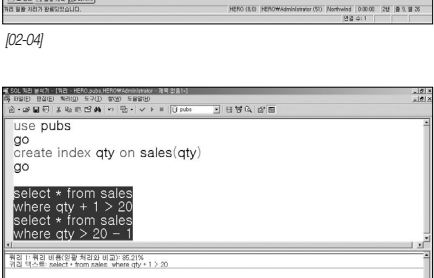


[02-03]

4. 다음 예제도 복습해 봅시다.



[02-04]



[02-05]

5. 항상 실행 계획을 참조하여 재차 쿼리를 확인해야 합니다.

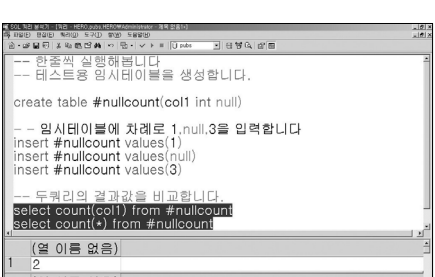
[참고] 쿼리 계획 은 다음의 몇 가지 단계로 이뤄집니다.

1. 평범한 계획을 식별
2. 계획을 단순화
   - having를 where로 != @param을 〈 @param OR 〉 @param으로 변환하는 것 같은 작업을 수행합니다
3. 통계를 로드한다
   - 쿼리 옵티마이저가 인덱스와 컬럼 통계 다른 지원 정보를 로드한다
4. 비교에 근거하여 계획들을 평가한다
   - 실행하는 비용이 충분히 저렴하다고 생각될 때 그 계획을 실행하도록 내놓는다
5. 병렬화를 위해 최적화한다
   - SMP

수칙3. COUNT(컬럼명) 대신 COUNT(*)을 사용하는가?

COUNT(*) 와 COUNT(컬럼명)의 차이는 중요합니다.
COUNT하는 해당 테이블 컬럼에 NULL 값을 포함하고 있다면 이 두 예제는 서로 다른 결과를 반환합니다.
COUNT(컬럼명)은 그룹에 포함된 각 행을 평가하여 NULL이 아닌 값의 개수를 반환합니다.
COUNT(*)는 NULL 값과 중복된 값을 포함한 그룹의 항목 개수를 반환합니다.

일반적으로, COUNT(컬럼명)을 사용하여 특정한 컬럼의 행 개수를 세는 것보다 COUNT(*)을 사용하여 옵티마이저가 행의 개수를 반환하는 최상의 방법을 선택하도록 해주는 것을 더 선호하는 방식이다.



[03-01]

[참고] NULL을 처리하는 방법

```
use pubs
go
-- 돈받고 파는 책을 출력하세요

select * from titles where price is not null

-- 비매품인 책을 출력하세요

select * from titles
where price is null

-- 비매품 책을 제외한 모든 책의 평균 가격?

select avg(price) from titles

-- 비매품 책을 0원으로 두고 계산한 평균 가격?

select avg(isnull(price,0)) from titles
```

[유용한 관용구]
칼럼의 중복행 수를 찾아봅시다

```
use pubs
go

-- 중복 칼럼이 각각 몇개 항목인지를 찾아보자
-- type별로 몇 개의 책이 있을까?

select type,count(*) as [중복행의 수]
from titles
group by type
having  count(*)〉 1
```

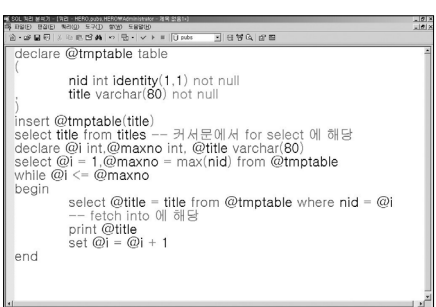
수칙4. 커서 및 임시 테이블의 내용을 최대한 자제하는가?

결론부터 말씀 드리자면 커서보다는 임시 테이블이 임시 테이블보다는 테이블 변수를 사용하는 것이 성능에 보탬이 됩니다. 단 SQL 2000에서만 테이블 변수가 가능합니다.

커서는 내부적으로 임시 테이블을 사용하기 때문에 임시 테이블을 쓴다고 부하가 더 발생하진 않습니다. 오히려 커서의 부가적 기능 때문에 서버 자원을 더 낭비하게 됩니다. (커서로 할 수 있는 건 임시 테이블이나 테이블 변수로도 모두 처리가 가능합니다.)

[따라하기 - 다음은 테이블 변수를 사용하여 기존 커서를 대체하는 것을 구현했습니다.]

1. 훌륭하게 커서를 대신하는 문장입니다.



```
declare @tmptable table
(
        nid int identity(1,1) not null
        title varchar(80) not null
)
insert @tmptable(title)
select title from titles -- 커서문에서 for select 에 해당
declare @i int,@maxno int, @title varchar(80)
select @i = 1,@maxno = max(nid) from @tmptable
while @i <= @maxno
begin
        select @title = title from @tmptable where nid = @i
        -- fetch into 에 해당
        print @title
        set @i = @i + 1
end
```

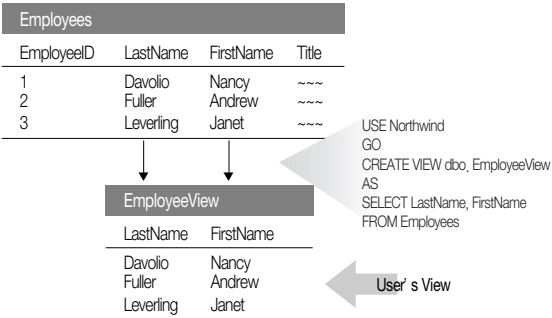
[04-01]

CTRL + K로 확인하면 테이블 변수로 사용할 경우 실제 테이블에 잠금을 전혀 걸지않는 것을 알 수 있습니다(중요) 그와 반대로 커서를 사용할 경우 프로시저 시작부터 끝까지 지속적으로 사용 부분을 계속해서 잠그고 있어서 다른 작업들이 대기해야되는 문제가 생깁니다.

수칙5. VIEW의 총사용을 줄였는가?

VIEW는 보안과 편리성에 관련된 이슈를 다루는 데 있어 최고입니다.

그러나, 일반적으로 보안상에 이슈를 제외한 경우에는 불필요한 부하가 가중될 수 있고 많은 경우에 더 불필요한 데이터를 반환합니다 예를 들면 VIEW에서 10개를 가져오고 거기에 WHERE 조건을 붙여서 7개만 가져오는 경우가 그렇습니다.



[05-01]

select lastname,firstname from employees VS select * from EmployeesView중간 단계가 있는 쪽이 효율이 떨어집니다.

수칙6. 저장 프로시저를 사용하는가?

저장 프로시저는 복잡한 SQL 문을 단순화 시켜주고, 보안 문제를 해결해주며 더 나아가 빠른 성능에 매개 변수,출력 매개 변수,리턴 값을 사용할 수 있습니다.

저장 프로시저의 역할 7가지

1. 데이터 무결성의 시행
2. 복잡한 비즈니스 규칙과 제약의 강화
3. 캡슐형 설계
4. 유지 보수
5. 네트워크 트래픽 감소(오고 가는 긴 SQL 구문을 축소)
6. 보다 빠른 실행(컴파일을 하지 않습니다)
7. 보안 강화

저장 프로시저의 생성과 반복 사용 시 발생하는 일

제작	
1. 구문 분석	2. 표준화
3. 보안 점검(프로시저 생성 권한)	4. 저장(syscomments)
첫 번째 실행 시	
1. 보안 점검(프로시저 실행 권한)	2. 최적화
3. 컴파일과 이에 따른 실행 계획을 캐쉬에 저장	
4. 실행	
반복해서 실행 시	
1. 캐쉬에 실행 계획 있을 때는 그대로 실행	
2. 캐쉬에 실행 계획이 없을 때는 첫 번째로 저장 프로시저 실행하는 것과 동일	

쿼리는 한번만 실행할 때는 일반 SQL이 훨씬 간단합니다. 그러나 반복적으로 실행되면 저장 프로시저가 월등히 빠르고 편리합니다.

수칙7. 저장 프로시저를 적절하게 리컴 파일 하는가?

데이터가 변화하면(인덱스를 추가하거나 인덱스된 열의 데이터를 변경하는 등의 작업 수행 시) 그에 걸맞게 실행계획도 변화해 갑니다. 그에 대처하기 위해서 다음과 같은 리컴 파일 방법를 제공합니다.

저장 프로시저 리컴 파일 모드에는 다음의 3가지가 있습니다.

```
CREATE PROCEDURE [WITH RECOMPILE]
EXECUTE [WITH RECOMPILE]
sp_recompile
```

CREATE PROCEDURE [WITH RECOMPILE] 는 SQL SERVER가 이 저장 프로시저의 계획을 캐시하지 않기 때문에 이 저장 프로시저가 실행 할 때 마다 다시 컴파일 됩니다(실행 속도가 느려짐).

EXECUTE [WITH RECOMPILE]는 지금 이순간만 리컴 파일 하고 다시 저장 프로시저 실행하면 예전 실행 계획대로 작동하는 것입니다. 제공하는 매개 변수가 불규칙하거나 저장 프로시저를 만든 다음 데이터가 많이 변경되었을 경우 이 옵션을 사용합니다.

sp_recompile는 저장 프로시저가 다음에 실행될 때 첫 실행처럼 컴파일되고 실행되도록 하는 것입니다.

[문서화되지 않은 DBCC 명령어]

```
-- pubs 데이터베이스의 모든 저장 프로시저를 재컴파일 해보자

select db_id('pubs')

dbcc flushprocindb(5)

-- 모든 인덱스를 재구축한다
-- 관리자가 사용할 경우 엄청난 시간이 소요될 수 있습니다

dbcc dbreindexall('pubs')
```

수칙8. 저장 프로시저 작명 시 SP외의 접두어를 사용한다.

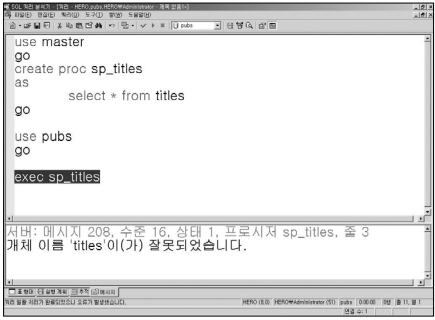
시스템 저장 프로시저는 master 데이터베이스내에서 sp_라는 접두어로 시작하는 것이 좋으며 모든 데이터베이스에서 실행될 수 있습니다. 각 사용자 데이터베이스에서는 다른 접두어를 사용하는 것이 보기에도 좋고 알아보기에도 수월합니다.

또한 시스템 저장 프로시저는 어느 데이터베이스에서 수행하건 해당 데이터베이스의 내용을 참조합니다.

[따라하기]

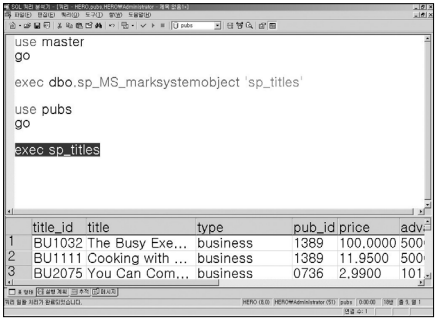
일반sp_ 저장 프로시저를 시스템sp_ 저장 프로시저로 만들어 봅니다.

1. 사용자 정의 저장 프로시저는 master 데이터베이스에 존재하더라도 master 내용만 참고합니다.



[08-01]

2. 그러나 다음과 같이 시스템 저장 프로시저화 한다면

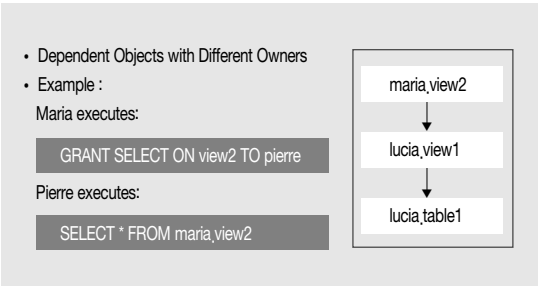


[08-02]

3. 부연하자면 모든 데이터베이스에서 사용하는 프로시저의 경우 sp_로 시작하게 작성한 후 sp_MS_marksystemobject로 시스템 프로시저화 작업을 하는게 필요합니다. 이 내용은 엄격하게 구분되서 실행되는 것이 혼란을 줄일 수 있습니다.

수칙9. 모든 개체의 소유자는 DBO이다

소유자가 다르면 복잡한 소유권 체인 문제가 발생합니다.

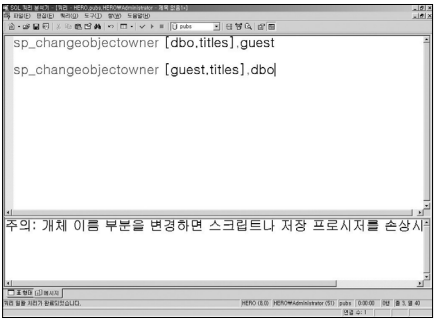


[09-01]

lucia가 테이블의 소유자 입니다 lucia는 뷰를 만들었는데 maria에게 뷰를 볼 수 있게 했습니다. maria는 이를 Pierre가 볼 수 있게 했는데 Pierre는 Maria가 만든 뷰를 select 권한을 받았음에도 불구하고 실행이 안됩니다 이는 소유권 체인이 중간에 분실 됐기 때문입니다. 불필요한 이런 시스템은 시스템의 성능 저하를 가져다 줍니다. 모든 소유자는 dbo로 통일하는 것을 권장합니다.

[따라하기 - 소유자를 dbo로 바꿔보자]

1. 소유자를 dbo로 바꿀 때는 다음의 저장 프로시저를 사용하면 됩니다.



[09-02]

2.추가로 시스템 테이블을 업데이트하는 방법을 통해 데이터베이스 차원에서 소유자를 바꾸는 방법도 있으며 커서를 사용하는 방법도 존재합니다.

[참고] 소유자가 dbo가 아닌 객체를 출력해봅시다

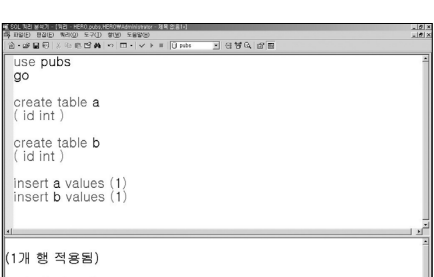
```
select name from sysobjects
where uid 〈〉 user_id('dbo')
```

수칙10. 데드락이 발생하는 부분을 라이브락 형태로 변경했는가?

데드락이란 라이브락과 반대되는 개념입니다. 둘 이상의 트랜잭션이 서로가 실행해야 될 내용을 이미 잠그고 있어 마치 교차로에서 서로 엉켜 꼼짝할 수 없는 상황을 의미 합니다. 이를 해결하기 위한 SQL 서버의 노력은 한쪽을 일방적으로 취소시키는 것인데 이는 시스템의 성능 저하로 나타납니다. 이를 해결하기 위한 가장 좋은 방법은 일방 통행 방식으로 변경하는 것입니다. 이것이 라이브락 입니다.

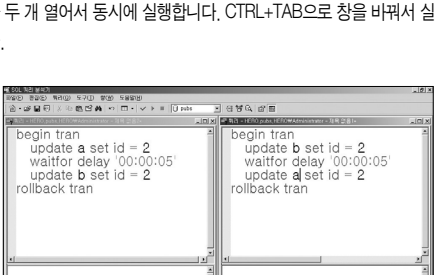
[따라하기]

1. 우선 준비를 위해 테이블을 만들고 데이터를 넣습니다.



[10-01]

2. 창을 두 개 열어서 동시에 실행합니다. CTRL+TAB으로 창을 바꿔서 실행해 봅니다.



[10-02]

3. 위의 데드락의 가장 바른 해결 방법은 순차적인 라이브락 형태로 변경하는 것입니다.

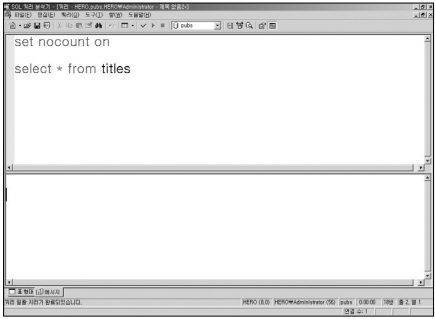
[10-03]

수칙11. SET NOCOUNT ON을 사용하는가?

불필요한 메시지가 네트워크 트래픽을 낭비하고 있습니다. 특히 '몇 개 행이 적용되었습니다' 같은 메시지가 그런 대표적인 예입니다.

[따라하기]

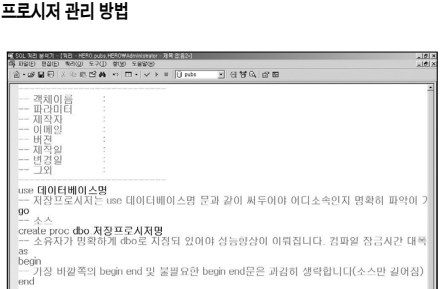
1. set nocount on을 실행하고 쿼리를 실행합니다.



[11-01]

수칙12. 실무 사례: 저장 프로시저 관리방법

* 저장 프로시저 관리 방법



[12-01]

```
- - - - - - - - - - - - - - - -
-- 객체 이름   :
-- 파라미터   :
-- 제작자     :
-- 이메일     :
-- 버젼       :
-- 제작일     :
-- 변경일     :
-- 그외       :
- - - - - - - - - - - - - - - -
```

use 데이터베이스명

- 저장 프로시저는 use 데이터베이스명 문과 같이 써두어야 어디 소속인지 명확히 파악이 가능합니다.

go

- 소스

create proc dbo.저장 프로시저명

- 소유자가 명확하게 dbo로 지정되 있어야 성능 향상이 이뤄집니다. 컴파일 잠금 시간 대폭 감소

as
begin
- 가장 바깥쪽의 begin end 및 불필요한 begin end 문은 과감히 생략합니다(소스만 길어짐)
end
- 실행 예제
exec 데이터베이스명.dbo.저장 프로시저명
- 데이터베이스 이름까지 명시해야 오브젝트 참조에서 발생할 수 있는 불명확성을 줄여줌으로 바람직합니다..

2. 그 외 개발자 권고 사항

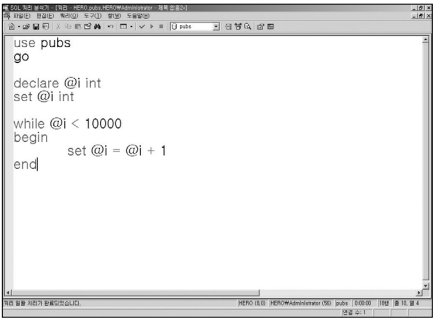
번호	수칙	체크
01	코드는 알아보기 쉽게 들여쓰기 합니다.	☐
02	주석을 적절히 사용합니다.	☐
03	ANSI-92 SQL을 사용합니다.	☐
04	반드시 오류 여부를 확인하는 코드를 작성합니다.	☐
05	기타 개발시 주요한 조언들	☐

수칙1. 코드는 알아보기 쉽게 들여쓰기 합니다.

기억을 잃어버린 자신이 알아볼 수 있도록 쓰라는 말이 있습니다. 쿼리는 알아보기 쉽게 필요한 경우 적당한 룰을 정해 알아보기 수월하게 작성하도록 합니다.

[따라하기]

1.적당한 들여쓰기와 띄어쓰기는 코드를 알아보기 수월하게 합니다.



[01-01]

선언부와 초기값을 앞에 몰아 두었습니다. 한 칸 띄우고 반복 실행문 안쪽의 문은 TAB으로 들여쓰기를 했습니다.
또한 다음과 같이 불필요한 begin end 문도 생략할 수 있습니다.

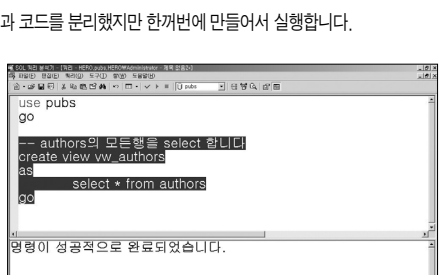
```
use pubs
go

create proc up_test1
as
begin
        select * from authors
end
go

create proc up_test2
as
        select * from authors
go

sp_helptext up_test1
sp_helptext up_test2
```

수칙2. 주석을 적절히 사용합니다.

저장 프로시저나 뷰에 주석을 포함시킬 수 있는데 쿼리 안이 아니라 쿼리 밖에도 저장 가능합니다. 이를 이용에 뷰의 제작일, 작성자, 주요 내용들을 소스와 함께 저장할 수 있습니다.

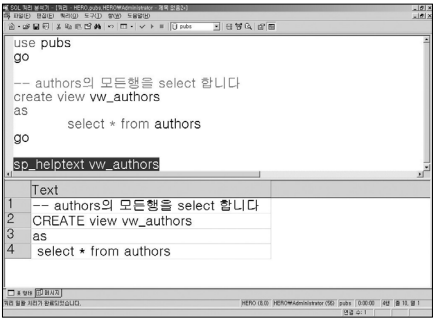
[따라하기]

1.주석과 코드를 분리했지만 한꺼번에 만들어서 실행합니다.



[02-01]

2. 소스를 보기 위에 sp_helptext를 사용합니다. Syscomments 테이블에는 주석까지 함께 저장되어 있는 것을 알 수 있습니다. 소스 관리는 코드 원문 뿐 아니라 주석도 함께 관리 합니다!



[02-02]

수칙3. ANSI-92 SQL을 사용합니다

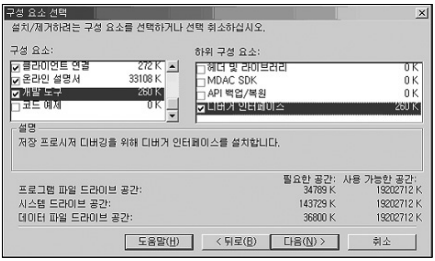
가능한 한 표준 ANSI 쿼리를 사용하도록 합니다

수칙4. 반드시 오류 여부를 확인하는 코드를 작성합니다.

코드는 철저한 디버깅 속에 개발되어야 합니다.

[따라하기 - 디버깅 툴을 설치하는 방법]

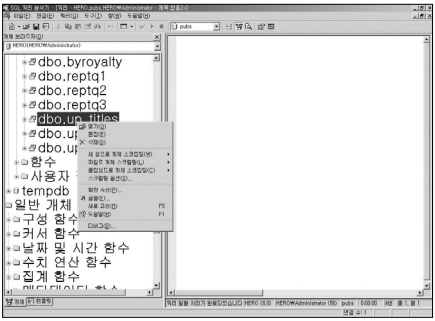
1. SQL 설치를 할 때 다음 항목을 설치하면 디버깅 도구가 설치되는 것입니다.



[04-01]

[따라하기- 디버깅을 사용하는 방법]

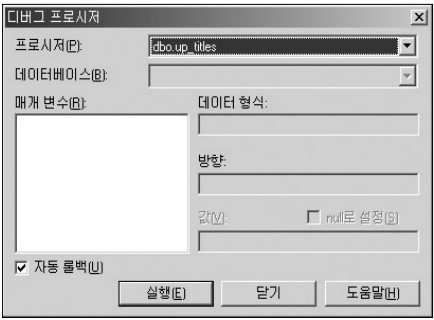
1. 쿼리 분석기를 실행합니다.
2. 목표로 하는 저장 프로시저를 객체 브라우저에서 선택합니다.
3. 저장 프로시저를 오른쪽 클릭하고 팝업 메뉴에서 디버그를 선택합니다.



[04-02]

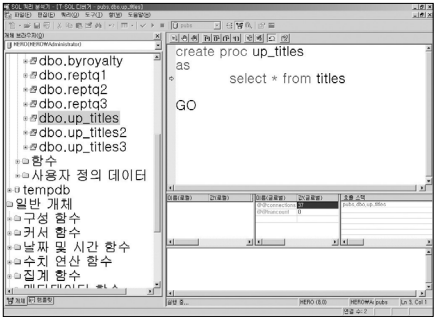
4 .

4. 파라미터 리스트의 각각의 파라미터를 클릭하여 값을 입력하고 [실행]을 클릭하면 SQL 서버는 디버거를 띄웁니다



[04-03]

5. 그 다음은 일반 디버거 프로그램과 동일하므로 각자 연구해보도록 합니다.



[04-04]

[참고] 트리거 및 사용자 정의 함수 디버깅

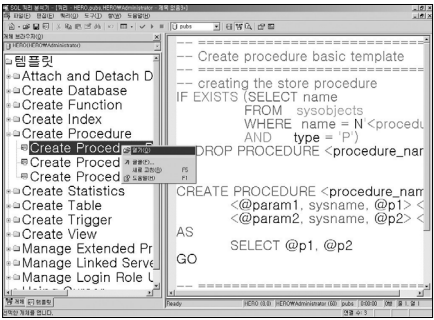
트리거나 함수를 디버깅하려면 그 트리거나 함수를 사용하는 프로시저를 만들어 디버거를 돌리면 됩니다.

수칙5. 기타 개발 시 중요한 조언들

템플릿을 사용해봅시다!

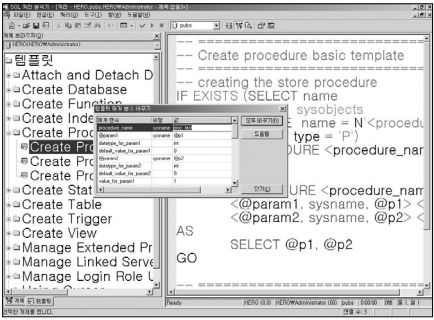
[따라하기 - 템플릿 활용]

1. 기존의 템플릿을 사용해보고 마음에 들면 대량의 작업 시 중요한 기초 템플릿이 될 수 있어 개발시간을 절약할 수 있습니다.
2. 쿼리 분석기의 템플릿을 선택 후 만들고자 하는 항목 선택 후 마우스 오른쪽 클릭 팝업 메뉴에서 열기 선택을 합니다.



[05-01]

3. 나오는 템플릿 화면에서 CTRL+SHIFT+M 을 클릭해봅니다. 놀라운 화면이 나타납니다. 아주 쉽게 제작의 토대를 그대로 사용할 수 있습니다.



[05-02]

4. 적절하게 기본값을 넣어봅니다. 더 나아가 주석도 위와 같은 방법으로 제작해 둡니다. 더욱 편리한 개발 환경이 구축됩니다.

가상 데이터를 다음과 같은 방법을 사용해 입력해 봅시다

```
-- 가상 데이터 입력 방법

-- 1.while 관용구
--
-- 2.더블링
-- 원하는 크기가 될때까지 테이블로 자신의 데이터를 삽입하는 것

use pubs
go

set nocount on

drop table tb_double

create table tb_double
(
                id int identity
,               ph int null
)

insert tb_double(ph) values(30)
insert tb_double(ph) values(35)
insert tb_double(ph) values(40)
insert tb_double(ph) values(45)
insert tb_double(ph) values(50)

set nocount off
declare @cnt int,@rcnt int,@targ int
```

```
select @targ= 100,@cnt = count(*),@rcnt=@targ-count(*) from tb_double
while @rcnt 〉 0
begin
                set rowcount @rcnt
                insert tb_double(ph) select ph from tb_double
                set @cnt = @cnt + @@rowcount
                set @rcnt = @targ - @cnt
end

set rowcount 0

select * from tb_double

go

-- 3.랜덤값 집어넣기
-- 다음의 filltable 유틸을 사용하면 간편합니다.
```

[참고] filltabl 사용방법.

FILLTABL.EXE(별첨) 라는 다음의 C로 만든 간단한 파일을 사용하면 됩니다.

명령 프롬프트에서

```
filltabl -Tpubs..테이블명 -N100000 -Usa -Ppassword -S서버명
```

하면 됩니다.

SQL Server for DBA
관리자를 위한 튜닝 가이드

3. 운영체제 환경 설정편

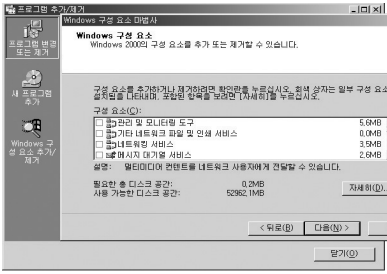
다음의 내용은 솔직히 아주 간단한 내용입니다. 당연하지만 잘 지켜지지 않는 부분이라고 도 할 수 있습니다. 꼭 정확히 설정을 당부드립니다.

번호	수칙	체크
01	필요한 윈도우 구성 요소만 설치했는가?	☐
02	최신 서비스팩과 핫픽스를 설치했는가?	☐
03	파티션은 NTFS를 사용하는가?	☐
04	불필요한 서비스를 사용중지 설정을 해두었는가?	☐

수칙1. 필요한 윈도우 구성 요소만 설치했는가?

불필요한 서비스의 설치는 관리자의 관리가 필요한 부분을 쓸데없이 확대시킴으로써 알수없는 위험 및 리소스 소모가 발생합니다.

OS를 윈도우 2000으로 사용하는 경우는 먼저 모든 선택된 기 구성 요소의 체크를 해제한 후 필요한 항목만을 설정하면 편리합니다.



[그림01-01 제어판:윈도우 구성 요소 추가/삭제]

앞의 그림처럼 SQL 서버에는 사실 위의 모든 구성 요소가 설치될 필요가 없습니다.

수칙2. 최신 서비스팩과 핫픽스를 설치했는가?

OS및 SQL 서버는 항시 최신 버전을 유지하거나 이에 상응하는 대응책을 취해 두어야합니다.

[Microsoft Baseline Security Analyzer를 다운받는 곳]
http://www.microsoft.com/korea/technet/security/tools/mbsahome.asp

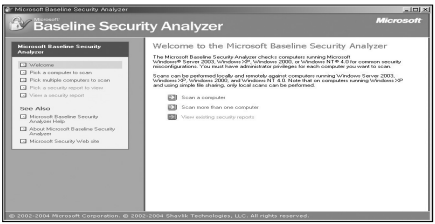
[SQL 서버 서비스팩을 다운받는곳]
http://www.microsoft.com/korea/sql/
우측의 Top 다운로드 메뉴에서 가장 최신의 SQL 서비스팩을 다운받아 설치하면 됩니다.

[따라하기]

Microsoft Baseline Security Analyzer를 다운받고 SQL 최신 패치를 다운 받아보자

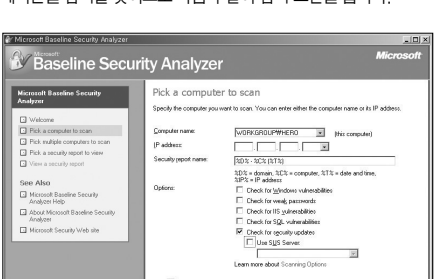
1. 그럼 Microsoft Baseline Security Analyzer를 실행하여 로컬시스템에 필요한 최신 SQL 패치 버전을 알아봅니다. 시작〉프로그램〉Microsoft Baseline Security Analyzer를 클릭합니다.

아래의 화면에서 Scan a computer를 선택합니다



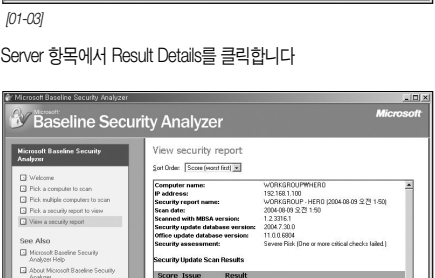
[01-02 실행 초기 화면]

2. 추가 패치만을 검색할 것이므로 다음과 같이 검색 조건을 줍니다.

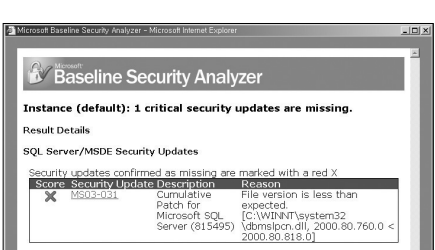


[01-03]

3. SQL Server 항목에서 Result Details를 클릭합니다

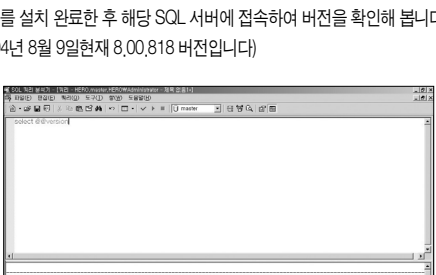


[01-04]



[01-05 패치가 하나 더 필요합니다.]

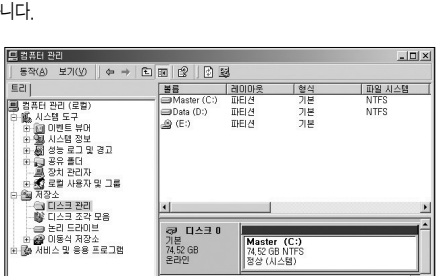
4. 파란색 숫자 부분을 클릭하여 해당 패치를 다운 로드받습니다.
5. 패치를 설치 완료한 후 해당 SQL 서버에 접속하여 버전을 확인해 봅니다.
   (2004년 8월 9일현재 8.00.818 버전입니다)



[01-06 버전 확인]

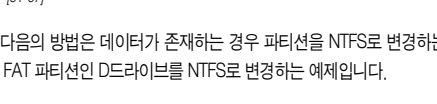
수칙3. 파티션은 NTFS를 사용하는가?

시스템내의 모든 파티션이 NTFS(보안 및 하드디스크 오류 검증 지원)로 설정되어 있는지 확인합니다. 제어판의 컴퓨터 관리 항목의 디스크 관리 항목에서 확인할 수 있습니다.



[01-07]

그리고 다음의 방법은 데이터가 존재하는 경우 파티션을 NTFS로 변경하는 방법입니다. FAT 파티션인 D드라이브를 NTFS로 변경하는 예제입니다.



[01-08]

수칙4. 불필요한 서비스를 사용중지 설정을 해두었는가?

불필요한 서비스에 대하여 가동시키는 것은 리소스 낭비일 뿐 아니라 보안상 허점을 만들수 있으므로, 사용중지 설정을 해두는 것이 좋다

[관련 링크 윈도우 2000 서비스 백서]
http://www.microsoft.com/korea/technet/prodtechnol/windows2000serv/deploy/prodspecs/win2ksvc.asp

[01-09]

장황하게 설명하면 내용이 많으므로 대부분의 DB 관리자가 사용한함 설정을 하는 권장 항목을 나열합니다.

Application Management
DHCP CLient
Fax
Messanger
Print Spooler
Remote Registry Service
Smart Card
Smart Card Helper
Telnet

Application Management
할당,게시,제거 같은 소프트웨어 설치 서비스를 제공합니다. SQL 서버와는 관련 없는 기능입니다

DHCP Client
SQL 서버는 고정 IP를 부여받아야 하므로 동적 IP 할당 클라이언트를 켜놓을 필요가 없습니다.

Fax
팩스를 SQL 서버에 부착하는 것은 말도 안됩니다.

Messanger
화면에 뿌려지는 메신저보다는 이벤트(파일)로 관리하거나 메일링을 사용하는 것이 기록에 남아서 더욱 편리합니다.

Print Spooler
프린터를 SQL 서버에 부착하는 것은 말도 안됩니다.

Remote Registry Service
원격 레지스트리 조작용 서비스 입니다

Smart Card
Smart Card Helper
스마트 카드 판독 관련 서비스 입니다

Telnet
원격 사용자가 시스템에 로그온하여 명령줄을 사용하여 콘솔 프로그램을 실행할 수 있게하는것입니다.

4. SQL 인스턴스 환경 설정

다른 SQL 인스턴스 설정값들은 상황에 따라 특수하게 공부를 해야하겠지만 아래의 3가지에 비에 그 중요도가 무척이나 떨어집니다 왜냐하면 SQL이 거의 자동으로 최적값에 세팅되어 있기 때문입니다.

번호	수칙	체크
01	서비스 이상 발생시 자동 시작을 설정해두었는가?	☐
02	고정 메모리를 사용하도록 해두었는가?	☐
03	SET OPTION을 사용하는가?	☐

수칙1. 서비스 이상 발생시 자동 시작을 설정해두었는가?

첫번째는 시스템 가동시 항상 서비스는 시작되어야 한다는 것이고 두번째는 문제가 생겨서 갑자기 작동을 멈추면 알아서 다시 시작되어야 한다는 것입니다.

[따라하기]

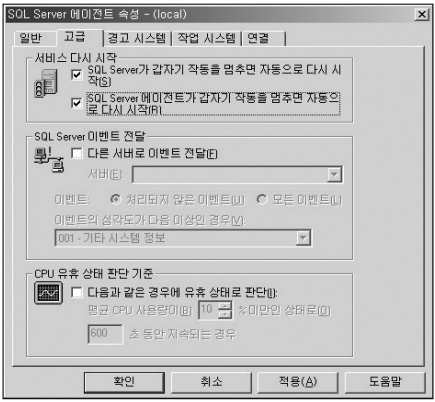
SQL Server및 Agent 서비스 자동 시작 모드로 설정하기

1. 자동 시작 설정을 해보겠습니다.[서비스 관리자]를 선택한 후 다음의 [자동 시작] 항목을 확인합니다.

[01-01]

[01-02]

2. 그 다음은 문제 발생시 자동 시작 설정입니다. SQL 서버 엔터프라이즈 관리자에서 SQL Server 에이전트 서비스에서 [등록 정보]를 선택한 후 [고급] 탭을 선택한 후 다음을 체크합니다.



[01-03 자동으로 다시 시작]

3. 위와 같이 적용해둔 후 SQL서버 서비스 및 에이전트 서비스를 재시작 합니다.

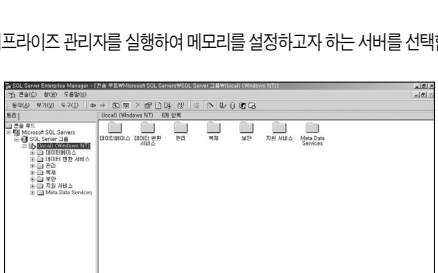
수칙2. 고정 메모리를 사용하도록 해두었는가?

SQL 서버는 필요할 때 메모리를 필요한 만큼 가져갔다가 다른 응용프로그램이 가져가게 할 수 도 있으며 또한 SQL 서버 전용으로 메모리를 아예 고정하는 방법도 있습니다. 상식적으로 생각해보아도 전용으로 사용하는 것이 SQL 서버를 위해 효과적인 것을 알 수 있습니다.

[따라하기]

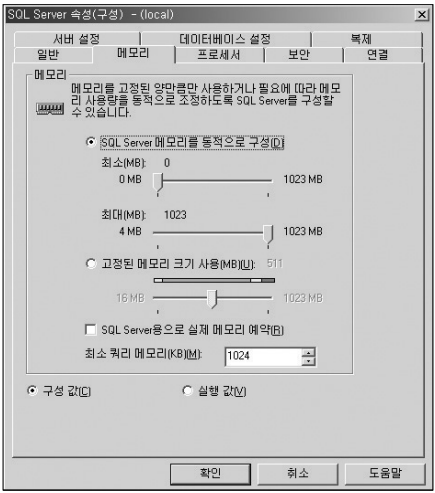
SQL 서버는 오직 데이터베이스 서비스만을 위해 사용하는 것이 좋다 운영체제가 사용할 약간량만을 제외한 나머지를 SQL 서버 서비스만을 위해 강제 할당 하는 것이 가장 효율적입니다.

1. 엔터프라이즈 관리자를 실행하여 메모리를 설정하고자 하는 서버를 선택합니다.



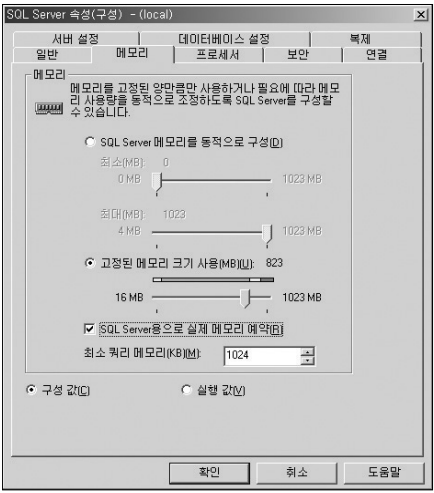
[02-01 엔터프라이즈 관리자 실행]

2. 서버 노드를 선택한후 등록 정보를 실행합니다. 실행후 메모리탭을 선택합니다. 기본으로 [SQL Server 메모리를 동적으로 구성]에 체크 되어있습니다.



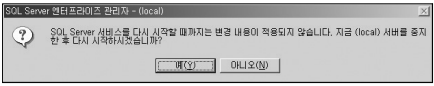
[02-02 메모리탭(기본은 동적으로 구성)]

3. 운영체제를 위한 적당량(200메가) 를 남겨두고 나머지는 SQL 서버를 위해 할당합니다.
[고정된 메모리] 체크 박스를 선택하고 [SQL 서버용으로 실제 메모리 예약]도 선택합니다. 이렇게 세팅해야만 다른 응용프로그램이 SQL 서비스에게서 메모리를 빼앗아가지 못합니다.



[02-03 고정된 메모리 크기 사용 및 실제메모리 예약]

4. 설정이 끝나면 역시 서비스를 재시작해야합니다.



[02-04 서비스 재시작]

수칙3. SET OPTION을 사용하는가?

가. SET NOCOUNT ON
나. SET ANSI_NULLS ON
다. SET QUOTED_IDENTIFIER ON
라. SET LOCKTIMEOUT 값 지정(디폴트 -1 : 무한대)

SET NOCOUNT ON이란? 메시지 창에 ~행이 적용되었습니다 하고 나타나는 시스템 메시지 입니다. 쿼리수가 많아지면 이런 메시지는 네트워크상에 쓰레기가 되니 아예 발생하지 않도록 하는 것이 시스템에 유리합니다.

SET ANSI_NULLS ON이란? SQL-92 표준에서는 NULL 값에 대한 Equals(=) 또는 Not Equal(<>) 비교의 결과가 검색되지 않아야 합니다. 쿼리 분석기에서 도움말 SET ANSI_NULLS ON를 검색한 후 예제를 따라해보십시오

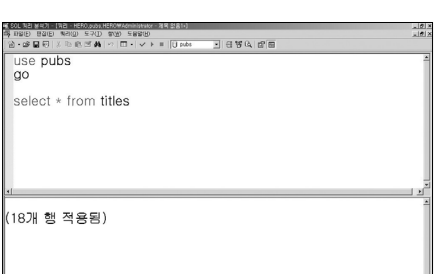
SET QUOTED_IDENTIFIER ON을 설정하면 SET QUOTED_IDENTIFIER 옵션을 ON으로 설정하면 식별자를 큰따옴표로 구분할 수 있으며, 리터럴은 작은따옴표로 구분해야 합니다.

SET LOCKTIMEOUT으로 잠금이 해제될 때가지 명령문이 기다려야 할 시간을 밀리초 단위로 지정할 수 있습니다. -1은 초기 상태 무한 대기 입니다.

다음의 따라하기를 참조로 위의 4가지 설정을 서버 차원에서 하도록 합니다.

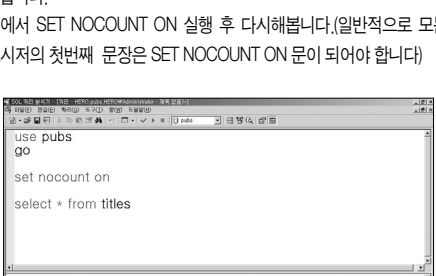
[따라하기 - SET NOCUNT ON의 의미 실험]

1. 쿼리 분석기에서 다음의 쿼리를 실행합니다.
USE pubs
GO
SELECT * FROM titles
2. 결과창에서 메시지 탭을 선택합니다.



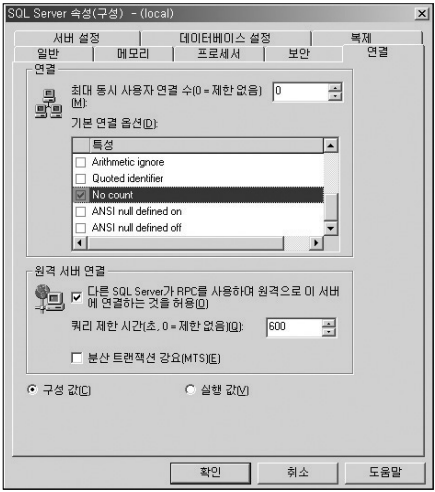
[02-08]

3. 메시지 탭에 (18개 행 적용됨) 이 보입니다. 이런 메세지도 수가 많아지면 부하가 발생합니다.
4. 쿼리에서 SET NOCOUNT ON 실행 후 다시해봅니다.(일반적으로 모든 저장 프로시저의 첫번째 문장은 SET NOCOUNT ON 문이 되어야 합니다)



[02-09]

5. 훌륭하게 적용되었으나, SET 문은 단지 지금 세션에만 적용되는 것 입니다. 서버 차원에서 다시 설정할 필요가 있습니다.



[02-10]

6. 훌륭하게 적용 되었습니다. 다른 것도 실험해 봅시다.

5. 데이터베이스 설정

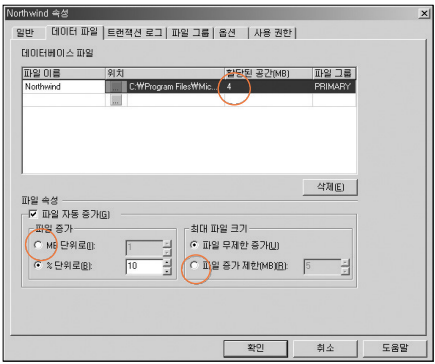
번호	수칙	체크
01	데이터베이스 기본 크기 및 증가율을 넉넉히 잡아 두었는가?	☐
02	필요할 경우 읽기 전용을 사용하는가? (온라인 분석 서버)	☐

수칙1. 데이터베이스 기본 크기 및 증가율을 넉넉히 잡아 두었는가?

데이터베이스 등록 정보를 보고 기본 크기를 여유있게 잡으며 증가 단위는 메가 단위로 1달에 1번 이상 데이터베이스 자동 증가가 이뤄지지 않도록 작성하는 것이 좋습니다. 가장 중요한 것은 최대 크기를 설정해두는 것입니다.

[따라하기 데이터베이스 속성 설정]

1.다음의 붉은 동그라미 부분을 설정하면 됩니다.



[01-01]

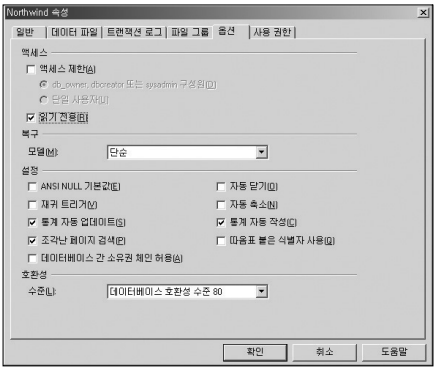
2. 로그 부분도 위의 3부분을 수정해줍니다.

수칙2. 필요할 경우 읽기 전용을 사용하는가?(온라인 분석 서버)

OLTP 즉 다수의 사용자가 데이터를 읽는 경우는 적절한 잠금 장치로 데이터의 일관성을 유지할 필요가 있으나 OLAP 분석 서버인 경우는 그럴필요가 없으므로 읽기 전용 속성이 유리합니다.

[따라하기 속성]

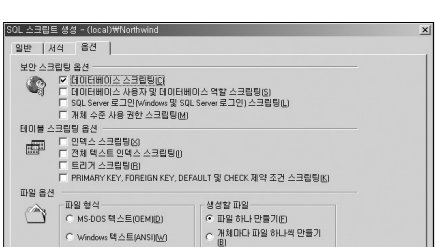
1. 데이터베이스를 선택한 후 등록 정보를 선택합니다. 옵션을 선택합니다.



[02-02]

[팁 데이터베이스 작성 스크립트 뽑아내기]

1. 엔터프라이즈 관리자를 실행합니다. 데이터베이스 항목을 선택한 후 모든 작업 〉 SQL 스크립트 생성 을 선택합니다.
2. 옵션에서 데이터베이스 스크립팅을 선택한 후 확인을 클릭합니다.



[02-03]

3. 다음은 저장된 파일을 열기만 하면됩니다.

6. 인덱스

번호	수칙	체크
01	적절한 인덱스가 걸려있는가?(I/O가 많은 경우 실행 계획 재검사)	☐
02	인덱스 튜닝 마법사로 점검했는가?	☐
03	상황 발생시 인덱스 채우기 비율을 조정하는가?	☐

수칙1. 적절한 인덱스가 걸려 있는가?

적절한 인덱스가 걸려있는지 인덱스 튜닝 마법사로 확인할 수 있습니다. 또는 CTRL + K로 실행 계획을 관찰 해도 됩니다.

인덱스를 만들어야 하는 장소

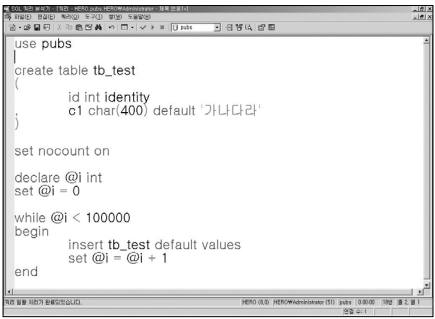
가. 참조키
나. 참조키가 아니더라도 join에 빈번히 사용되는 경우
다. select절에 자주 사용되는 칼럼
라. where,group by,order by절에 자주 사용되는 곳

수칙2. 인덱스 튜닝 마법사로 점검했는가?

인덱스란 책의 목차나 책 뒤쪽의 찾아 보기와 매우 유사합니다. 예를 들면 40메가의 데이터중 필요한 내용을 찾고자 한다면 40메가를 모두 검색해야하지만 인덱스를 만들어 둔다면 인덱스만 읽음으로써 보다 적은 리소스에 사용만으로, 필요한 내용 검색을 끝낼수 있습니다.

[따라하기 적절한 인덱스 자동 만들기]

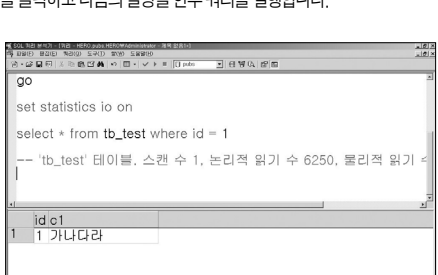
1. 다음과 같은 테이블을 예제 테이블을 pubs 데이터베이스에 만들어서 가상 Data 10만개를 입력해봅시다.



```
use pubs
create table tb_test
(
        id int identity
        c1 char(400) default '가나다라'
)

set nocount on

declare @i int
set @i = 0

while @i < 100000
begin
        insert tb_test default values
        set @i = @i + 1
end
```

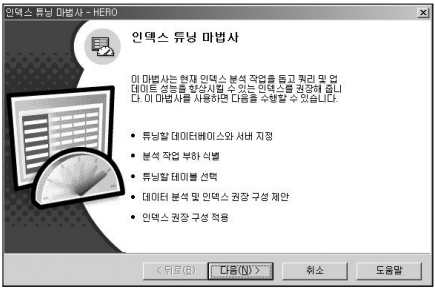
[02-01]

2. 10만개의 데이터가 입력됐으면 select 쿼리로 실험을 시작하도록 합니다. 먼저 총 I/O가 얼마가 일어나는지 또 실행 계획은 무엇인지 알아보겠습니다. 단축키 CTRL + K를 클릭하고 다음의 설정을 한후 쿼리를 실행합니다.

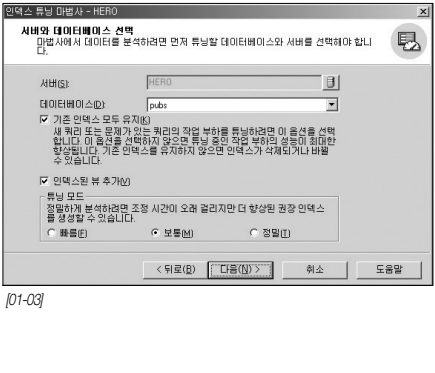


[02-02]

3. 데이터 한 개를 가져 오기 위해 테이블 전체를 검색하고 Data 페이지 6250페이지를 읽었군요. 6250페이지는 6250 * 8 다시말해 약 50메가를 검색하고 있습니다.
4. 여기에서 인덱스를 만들어 보겠습니다.
   create index idx on tb_test(id)
5. 다시 쿼리를 실행하면 I/O 는 몇페이지가 나옵니까?
   select * from tb_test where id = 1
6. 인덱스를 제거하고 이번에는 자동 인덱스 추천을 한번 해보겠습니다.
   drop index tb_test.idx
7. select 쿼리를 드래그 하여 선택 후 도구 메뉴에서 인덱스 튜닝 마법사(CTRL+I)를 선택합니다.

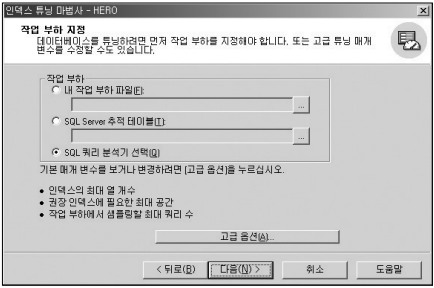

인덱스 튜닝 마법사 - HERO
인덱스 튜닝 마법사
이 마법사는 현재 인덱스 분석 작업을 돕고 쿼리 및 업데이트 성능을 향상시킬 수 있는 인덱스를 권장해 줍니다. 이 마법사를 사용하면 다음을 수행할 수 있습니다.
튜닝할 데이터베이스와 서버 지정
분석 작업 부하 식별
튜닝할 테이블 선택
데이터 분석 및 인덱스 권장 구성 제안
인덱스 권장 구성 적용
< 뒤로(B)
다음(N) >
취소
도움말

[01-02]


인덱스 튜닝 마법사 - HERO
서버와 데이터베이스 선택
마법사에서 데이터를 분석하려면 먼저 튜닝할 데이터베이스와 서버를 선택해야 합니다.
서버(S):
HERO
데이터베이스(D):
pubs
기존 인덱스 모두 유지(K)
새 쿼리 또는 문제가 있는 쿼리의 작업 부하를 튜닝하려면 이 옵션을 선택합니다. 이 옵션을 선택하지 않으면 튜닝 중인 작업 부하의 성능이 최대한 향상됩니다. 기존 인덱스를 유지하지 않으면 인덱스가 삭제되거나 바뀔 수 있습니다.
인덱스된 뷰 추가(V)
튜닝 모드
정밀하게 분석하려면 조정 시간이 오래 걸리지만 더 향상된 권장 인덱스를 생성할 수 있습니다.
빠름(F)
보통(M)
정밀(T)
< 뒤로(B)
다음(N) >
취소
도움말

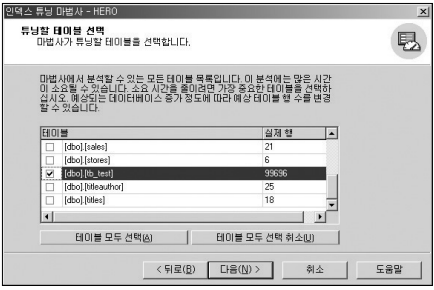
[01-03]

8. SQL 쿼리 분석기 선택 체크를 선택합니다.



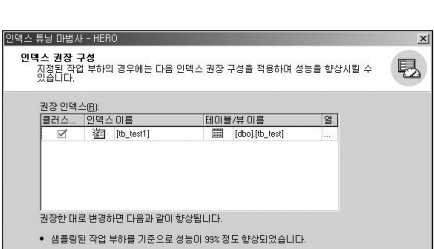
[01-04]

09. 튜닝할 테이블만을 선택합니다.



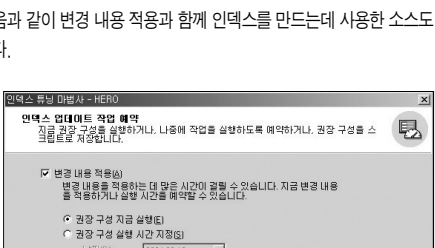
[01-05]

10. 다음과 같은 인덱스가 권장되었습니다.



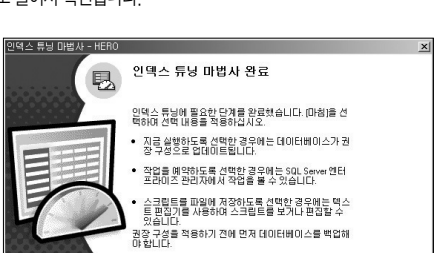
[01-06]

11. 다음과 같이 변경 내용 적용과 함께 인덱스를 만드는데 사용한 소스도 저장합니다.

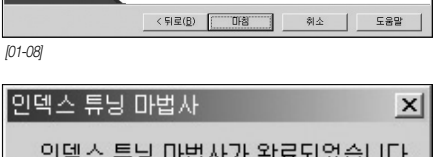


[01-07]

12. 마침을 클릭합니다. 인덱스도 훌륭히 적용된 것을 알 수 있습니다. 생성된 쿼리도 열어서 확인합니다.



[01-08]



[01-09]

13. 인덱스 결과도 확인합니다. 응용으로는 한번에 많은 쿼리를 선택한 후 인덱스 튜닝 마법사를 실행 할 수 있다는 것입니다.

sp_helpindex tb_test

[관련링크]
http://support.microsoft.com/default.aspx?scid=kb;ko;271509

위의 파일에 오류가 있는데 수정한 것은 다음과 같습니다.

```
use master
go

f exists (select * from sysobjects where id = object_id('dbo,sp_blocker_pss80') and
sysstat & 0xf = 4)
   drop procedure dbo,sp_blocker_pss80
GO

create proc sp_blocker_pss80 (@fast int = 0)
as
set nocount on
declare @spid varchar(6), @uid varchar(6), @blocked varchar(6)
declare @tmpchar varchar(255)
declare @time datetime

select @time = getdate()

declare @probclients table(spid smallint, blocked smallint, waittype binary(2),
primary key (blocked,spid))
insert @probclients select spid, blocked, waittype from sysprocesses where
blocked!=0 or waittype != 0x0000
if exists (select spid from @probclients)
begin
   select @tmpchar='Start time: ' + convert(varchar(26), @time, 113)
   print @tmpchar

 insert @probclients select blocked, 0, 0x0000 from @probclients
      where blocked not in (select spid from @probclients) and blocked != 0
```

```
print ' '
 if (@fast = 1)
 begin
   select spid, status, blocked, open_tran, waitresource, waittype,
     waittime, cmd, lastwaittype, cpu, physical_io,
     memusage,last_batch=convert(varchar(26), last_batch,113),
     login_time=convert(varchar(26), login_time,113), net_address,
     net_library,dbid, ecid, kpid, hostname,hostprocess,
     loginame,program_name, nt_domain, nt_username, uid, sid
   from master..sysprocesses
   where blocked!=0 or waittype != 0x0000
   or spid in (select blocked from @probclients where blocked != 0)
   or spid in (select spid from @probclients where blocked != 0)

   select      spid = convert (smallint, req_spid),
     ecid = convert (smallint, req_ecid),
     rsc_dbid As dbid,
     rsc_objid As ObjId,
     rsc_indid As IndId,
     Type = case rsc_type when 1 then 'NUL'
                         when 2 then 'DB'
                         when 3 then 'FIL'
                         when 4 then 'IDX'
                         when 5 then 'TAB'
                         when 6 then 'PAG'
                         when 7 then 'KEY'
                         when 8 then 'EXT'
                         when 9 then 'RID'
                         when 10 then 'APP' end,
     Resource = substring (rsc_text, 1, 16),
     Mode = case req_mode + 1 when 1 then NULL
```

```
                    when 2 then 'Sch-S'
                    when 3 then 'Sch-M'
                    when 4 then 'S'
                    when 5 then 'U'
                    when 6 then 'X'
                    when 7 then 'IS'
                    when 8 then 'IU'
                    when 9 then 'IX'
                    when 10 then 'SIU'
                    when 11 then 'SIX'
                    when 12 then 'UIX'
                    when 13 then 'BU'
                    when 14 then 'RangeS-S'
                    when 15 then 'RangeS-U'
                    when 16 then 'RangeIn-Null'
                    when 17 then 'RangeIn-S'
                    when 18 then 'RangeIn-U'
                    when 19 then 'RangeIn-X'
                    when 20 then 'RangeX-S'
                    when 21 then 'RangeX-U'
                    when 22 then 'RangeX-X'end,
    Status = case req_status when 1 then 'GRANT'
                    when 2 then 'CNVT'
                    when 3 then 'WAIT' end
    ,req_transactionID As TransID,
     req_transactionUOW As TransUOW
  from master.dbo.syslockinfo s,
    @probclients p
  where p.spid = s.req_spid
end  -- fast set
```

```
else
  begin  -- Fast not set
    select spid, status, blocked, open_tran, waitresource, waittype,
      waittime, cmd, lastwaittype, cpu, physical_io,
      memusage,last_batch=convert(varchar(26), last_batch,113),
      login_time=convert(varchar(26), login_time,113), net_address,
      net_library,dbid, ecid, kpid, hostname,hostprocess,
      loginame,program_name, nt_domain, nt_username, uid, sid
    from master..sysprocesses

    print ''
    print 'SPIDs at the head of blocking chains'
    select spid from @probclients
      where blocked = 0 and spid in (select blocked from @probclients where
spid != 0)
    print ''

    select      spid = convert (smallint, req_spid),
      ecid = convert (smallint, req_ecid),
      rsc_dbid As dbid,
      rsc_objid As ObjId,
      rsc_indid As IndId,
      Type = case rsc_type when 1 then 'NUL'
                          when 2 then 'DB'
                          when 3 then 'FIL'
                          when 4 then 'IDX'
                          when 5 then 'TAB'
                          when 6 then 'PAG'
                          when 7 then 'KEY'
                          when 8 then 'EXT'
                          when 9 then 'RID'
                          when 10 then 'APP' end,
```

```
    Resource = substring (rsc_text, 1, 16),
    Mode = case req_mode + 1 when 1 then NULL
                        when 2 then 'Sch-S'
                        when 3 then 'Sch-M'
                        when 4 then 'S'
                        when 5 then 'U'
                        when 6 then 'X'
                        when 7 then 'IS'
                        when 8 then 'IU'
                        when 9 then 'IX'
                        when 10 then 'SIU'
                        when 11 then 'SIX'
                        when 12 then 'UIX'
                        when 13 then 'BU'
                        when 14 then 'RangeS-S'
                        when 15 then 'RangeS-U'
                        when 16 then 'RangeIn-Null'
                        when 17 then 'RangeIn-S'
                        when 18 then 'RangeIn-U'
                        when 19 then 'RangeIn-X'
                        when 20 then 'RangeX-S'
                        when 21 then 'RangeX-U'
                        when 22 then 'RangeX-X'end,
    Status = case req_status when 1 then 'GRANT'
                        when 2 then 'CNVT'
                        when 3 then 'WAIT' end
    ,req_transactionID As TransID,
     req_transactionUOW As TransUOW
  from master,dbo,syslockinfo
end -- Fast not set
dbcc traceon(3604)
```

```
Print ''
Print ''
Print '********************************************************************************'
Print 'Print out DBCC Input buffer for all blocked or blocking spids.'
Print 'Print out DBCC PSS info only for SPIDs at the head of blocking chains'
Print '********************************************************************************'

declare ibuffer cursor fast_forward for
select cast (spid as varchar(6)) as spid, cast (blocked as varchar(6)) as blocked
from @probclients
where (spid <> @@spid) and (blocked!=0 or waittype != 0x0000
  or spid in (select blocked from @probclients where blocked != 0))
open ibuffer
fetch next from ibuffer into @spid, @blocked
while (@@fetch_status != -1)
begin
  print ''
  print ''
  print 'DBCC INPUTBUFFER FOR SPID ' + @spid
  exec ('dbcc inputbuffer (' + @spid + ')')
  print ''
  if (@blocked = '0')
  -- if DBCC PSS is not required, comment the line above, remove the
  -- comment on the line below and run the stored procedure in fast
  -- mode
  -- if (@blocked = '0' and @fast = 0)
  begin
    print 'DBCC PSS FOR SPID ' + @spid
    exec ('dbcc pss (0, ' + @spid +')')
    print ''
    print ''
end
```

```
 fetch next from ibuffer into @spid, @blocked
end
deallocate ibuffer

if datediff(millisecond, @time, getdate()) > 1000
begin
  select @tmpchar='End time: ' + convert(varchar(26), getdate(), 113)
  print @tmpchar
end

dbcc traceoff(3604)
end -- All
go
```

[참고]

그러나 현업에선 프로 파일러가 UI 부하 때문에 사용이 망설여 집니다 그래서 proc 로 제작해서 사용하는 것을 권장합니다. 위의 시스템 프로시저를 제작한 후 사용해봅니다.

직접 아래 쿼리를 수행하거나 결과를 파일로 만들어 저장할 수 있습니다.

```
-- checkblk.sql

DBCC TRACEON (3604)
GO
WHILE 1=1
BEGIN
  -- EXEC sp_blocker_pss80
  -- Or for fast mode
  EXEC sp_blocker_pss80 1
  WAITFOR DELAY '00:00:15'
END
GO
```

실제 위의 파일을 저장한 곳에서 다음의 명령 프롬프트에서 이렇게 수행하면 됩니다.

```
osql -E -icheckblk,sql -ocheckblk,out -w2000
```

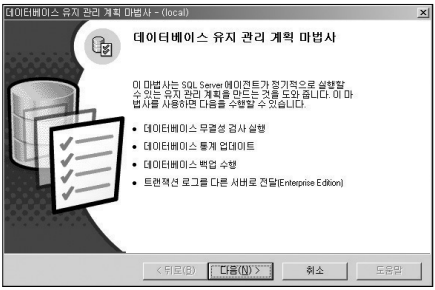
그 다음은 결과만 분석하면 됩니다. 락에 대기중인 쿼리가 딱 나와있습니다.

수칙3. 상황 발생시 인덱스 채우기 비율을 조정하는가?

인덱스는 검색할 때 속도는 무척 우수합니다. 그러나 insert 작업시 인덱스의 많은 변화가 요구되는 페이지 분할(Page Split) 발생할 수가 있어서 아예 여유 공간을 비워두는 것이 좋습니다.

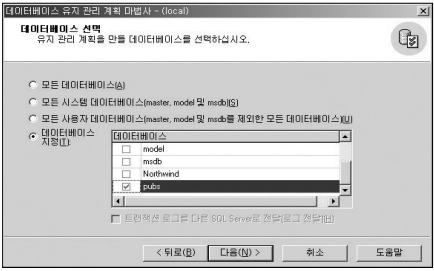
[따라하기]

1. 데이터베이스 유지 관리 마법사 노드에서 마우스 오른쪽 클릭 후 새 유지 관리 계획 마법사를 실행합니다.



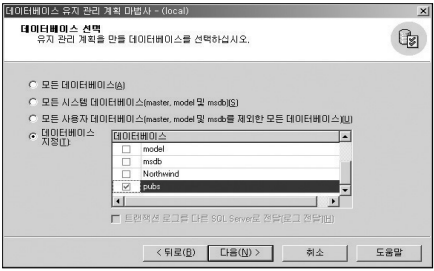
[03-01]

2. 인덱스를 정돈할 데이터베이스를 선택합니다

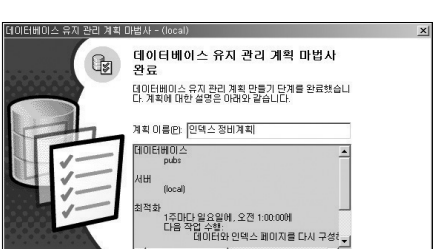


[03-02]

3. 다음과 같이 인덱스 비우기 비율을 적당량(상황에따라)을 선택한 후 다음 버튼을 클릭합니다.



[03-04]



[03-05]

4. 정기적으로 재정비 해주면 좋습니다.

7. 잠금

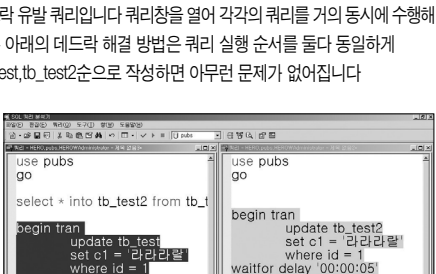
번호	수칙	체크
01	시간이 너무 오래걸리는 트랜잭션이 있는가?	☐
02	데드락을 모니터링해서 개발자에게 해결을 요청했는가?	☐

수칙1과2. 시간이 너무 오래걸리는 트랜잭션이나 데드락을 모니터링 해서 뽑아내는가?

위의 두가지 사항은 손쉽게 프로필러로 뽑아낼 수 있습니다. 다음의 따라하기는 단순한 모니터링일뿐 근원적인 해결책은 아닙니다. 해결은 도움말 잠금편을 읽어보고 해결해야만 합니다. 또는 외부에 해결을 의뢰하기 위해 파일로 저장할 수도 있습니다.

[따라하기]

1. 데드락 유발 쿼리입니다 쿼리창을 열어 각각의 쿼리를 거의 동시에 수행해 봅시다. 물론 아래의 데드락 해결 방법은 쿼리 실행 순서를 둘다 동일하게 tb_test,tb_test2순으로 작성하면 아무런 문제가 없어집니다



```
use pubs
go

select * into tb_test2 from tb_t

begin tran
        update tb_test
        set c1 = '라라라랄'
        where id = 1
waitfor delay '00:00:05'
        update tb_test2
        set c1 = '라라라랄'
        where id = 1
rollback tran
```

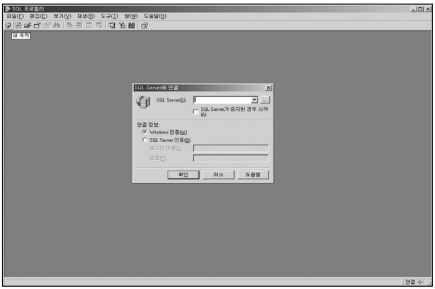
```
use pubs
go


begin tran
        update tb_test2
        set c1 = '라라라랄'
        where id = 1
waitfor delay '00:00:05'
        update tb_test
        set c1 = '라라라랄'
        where id = 1
rollback tran
```

[01-01]

2. 프로필러를 실행시킵니다. 새추적을 클릭합니다

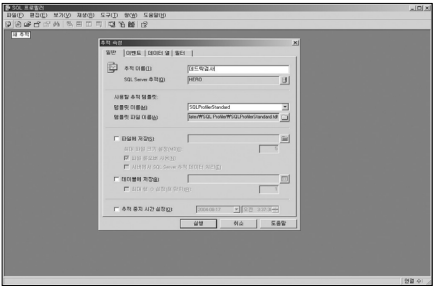
[01-02]

3. 데드락을 감시하고자 하는 서버에 접속합니다.



[01-03]

4. 데드락을 검사하려면 이름을 정하고

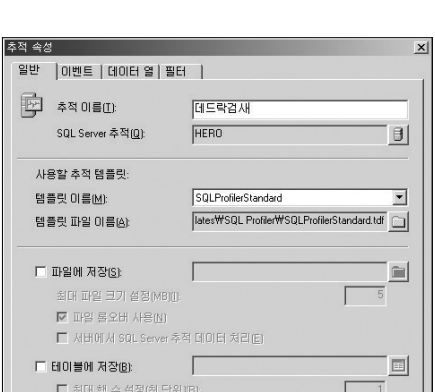


[01-04]

5. 이벤트탭에서 잠금 항목의 데드락 및 체인을 선택합니다.

[01-05]

6. 데드락으로 표시되는 부분은 붉게 표시되어 나타납니다. 개발자와 상담하여 데드락을 유발시키는 쿼리를 라이브락으로 변경 시켜주어야합니다.



[01-06]

7. 기다리면 시스템에서 발생하는 쿼리들중 데드락인 부분은 붉게 표시되어 나타납니다.
   추가로 앞서 제작해 두었던 sp_blocker_pss80도 꼭 실험을 해보도록 해야합니다.

그렇다면 보다 자세한 잠금 관련 공부를 해보겠습니다

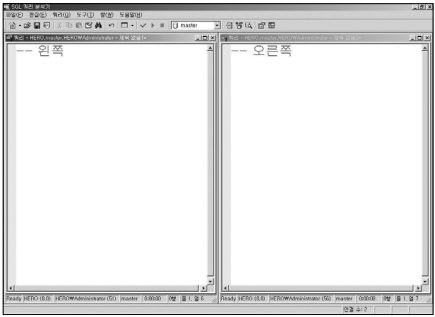
잠금 관련 권고 사항

가. 트랜잭션은 가능한 짧게 한다
나. 데드락을 피하기 위해
A. 같은 방향으로 트랜잭션을 진행합니다.
B. 잠금 수준을 올려줄 수도 있어야 합니다
다. 잠금 수준을 내려서 불필요한 잠금을 없애야합니다(read uncommitted)
라. 트리거를 사용하지 않습니다.
마. 대규모 데이터 변경시에만 커서를 사용합니다.

[따라하기- 트랜잭션은 왜 짧아야 하는가?]

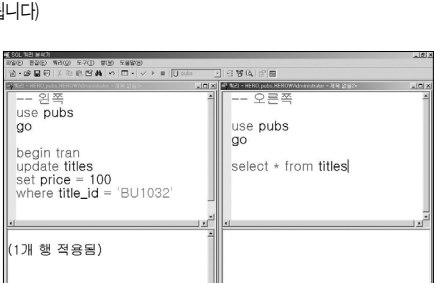
트랜잭션이 길어지면 잠금이 길어지고 그 만큼 다른 작업도 지연된다 다음 대표적인 잠금 대기 실험을 해보겠습니다.

1. 다음과 같이 창을 두개 만듭니다. CTRL+N을 클릭하여 새창을 만든 후 메뉴에서 창 〉 새로 바둑판식 배열을 선택합니다.



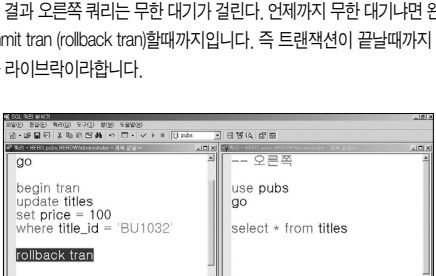
[01-07]

2. 왼쪽창에선 트랜잭션 중간에 멈추고 오른쪽에서 그 트랜잭션이 잠그는 부분과 충돌하는 장면을 연출해봅시다.
3. 왼쪽창을 다 실행한 다음 오른쪽을 실행합니다.(price는 기존의 값이 아닌값이면 됩니다)



[01-08]

3. 실행 결과 오른쪽 쿼리는 무한 대기가 걸린다. 언제까지 무한 대기냐면 왼쪽에서 commit tran (rollback tran)할때까지입니다. 즉 트랜잭션이 끝날때까지 입니다. 이를 라이브락이라합니다.

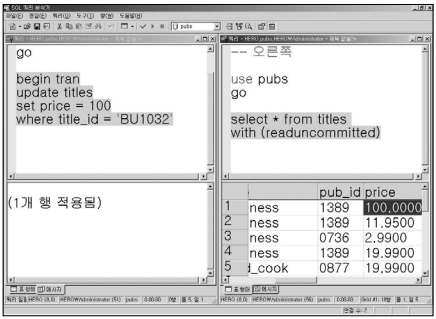


[01-09]

4. 성공적으로 오른쪽 쿼리가 실행됩니다. 따라서 트랜잭션은 빠르면 빠를 수 록 좋습니다.

[따라하기 - 잠금 수준을 낮춰서 잠금 충돌을 회피합시다]

1. 왼쪽 쿼리를 commit tran을 하지 않은 상태에서 다음과 같은 오른쪽 쿼리를 실행합니다.

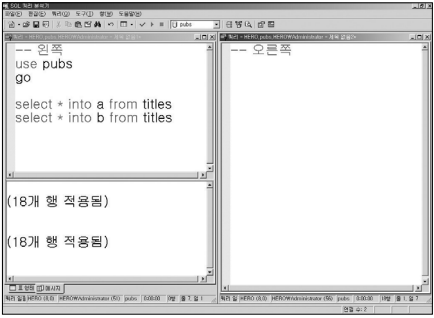


[01-10]

2. 실제 데이터가 아닌 버퍼에 있는 값을 읽어오므로 잠금과는 상관이 없는 readuncommitted를 사용한 예제입니다.

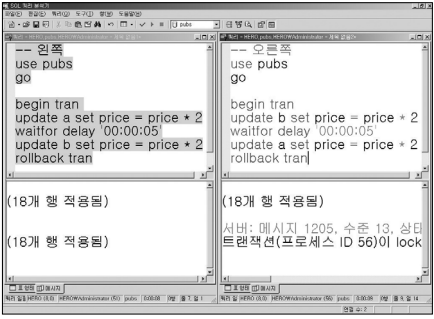
[따라하기 - 데드락을 피하는 노력들!]

1. 실험에 사용할 테이블 두개를 만듭니다.



[01-11]

2. 데드락 중에서 순환 데드락을 구현해보겠습니다 순환 데드락이란 서로가 서로 업데이트할 내용을 막고 있는 형태입니다.



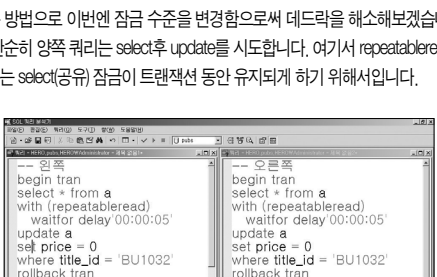
[01-12]

3. 위의 상황은 업데이트 잠금 때문에 다른 트랜잭션의 업데이트가 실행되지 못하는 것입니다 해결하려면 다음과 같이 해줘야 합니다. 같은 방향으로 트랜잭션을 진행합니다.

[01-13]

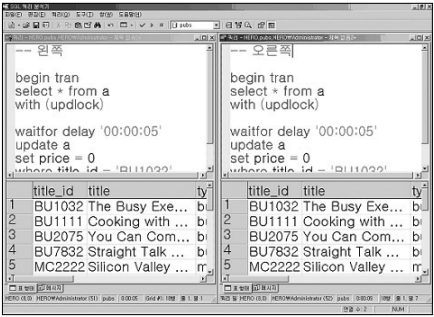
[따라하기 - 변환 데드락]

1. 같은 방법으로 이번엔 잠금 수준을 변경함으로써 데드락을 해소해보겠습니다. 우선 단순히 양쪽 쿼리는 select후 update를 시도합니다. 여기서 repeatableread를 한 이유는 select(공유) 잠금이 트랜잭션 동안 유지되게 하기 위해서입니다.



[01-14]

2. select와 공유는 동시에 되지 않습니다. 차라리 select할때 다른 트랜잭션도 select 못하도록 잠금 수준을 항상시킵니다.



[01-15]

데드락의 대표예 두가지를 성공적으로 해결했습니다. 결론을 다시 반복하자면 순환 데드락의 경우는 트랜잭션 방향을 일방통행으로 변환 데드락은 잠금 수준을 조정함으로써 해결해야합니다.

8. 모델링

번호	수칙	체크
01	정규화 및 적절한 경우의 비정규화가 잘 이뤄졌는가	☐

수칙1. 정규화 및 적절한 경우의 비정규화가 잘 이뤄졌는가?

정규화 및 비정규화의 궁극적인 목표는 데이터의 중복을 제거하고 최소한의 논리적 단위로 테이블로 분리하는데 있습니다. 자세한 내용은 관련 모델링 서적을 참고하고 여기서는 정규화 수칙만 언급하겠습니다.

제1정규화 : 반복되는 그룹 속성을 제거한 뒤 기본 테이블의 기본키를 추가해 새로운 테이블을 생성하고 기존 테이블과 1:n관계를 만듭니다.

제2정규화 : 복합키에 전체적으로 의존하지 않는 속성들을 제거합니다

제3정규화 : 기본키에 의존하지 않고 일반 컬럼에 의존하는 컬럼들을 제거합니다.

비정규화 : 1,2,3정규화가 끝나면 필요에 따라 특수 테이블을 만들어 사용할수있습니다.

마치면서

본 포켓 튜닝 가이드는 어디까지나 기초적인 내용을 담고 있습니다. 자세한 내용은 이를 바탕으로 더욱 정진하시기 바랍니다.

- SQL Server 컨설턴트 차주언 저

비매품

SQL Server 성능 향상을 위한 튜닝 가이드

- 저 자: 차 주 언
- 감 수: 차 재 호, 정 순 용
- 기 획: 안 덕 중 ㈜웹타임
- Contents 관련문의: djahn@wtime.net

- 발 행: ㈜한국마이크로소프트
- 제 작: ㈜웹타임
- 초판 발행일: 2004년 9월 23일
- 개정판 발행일: 2004년 11월 1일

SQL Server 성능 향상을 위한
튜닝 가이드

Microsoft
(주)한국마이크로소프트
서울특별시 강남구 대치동 892번지 포스코센터 서관 5층
전화 : 02-531-4500(대)
팩스 : 02-555-1724
인터넷 : http://www.microsoft.com/korea/sql