Microsoft SQL Server 2005

# Microsoft SQL Server 2005 개발자 가이드

Microsoft

## 저자 약력

김정선 (주)필라넷 수석 컨설턴트
MVP, MCDBA, 멀티캠퍼스 전임강사, 전 삼성 중공업
컨설팅: 삼성 반도체, 무림제지, 대우 중공업, 롯데 제과, Yes24, 대정 화금 외 다수

성대중 (주)필라넷 책임 컨설턴트
전, 영림원, SQL Server 매거진 전문 번역
컨설팅: 롯데 칠성, 삼성 반도체, LG 상사, Yes24, 두산그룹, 홍진 크라운 외 다수

현중균 (주)필라넷 주임 컨설턴트
전 한양여대 겸임교수, MCSE, MCDBA, SQL Server 매거진 전문 번역
컨설팅: CyberMBA, Yes24, Quest Tech Engineer

## 감수자 약력

정원혁 (주)필라넷 상무 이사 , DB사업부장, MCDBA, MCT
전문가로 가는 지름길, MS SQL Server 2000외 7권 저술, SQL Server 매거진 전문 번역
전, 마이크로소프트, 이랜드
컨설팅: 두산, 롯데 카드, 삼성 반도체, KT, CJ CGV, 대한 생명, 동양 증권, 동양 생명 ,
신한 은행, 제일 은행 외

## 비매품

# SQL Server 2005 개발자 가이드


- 저 자: 김 정 선, 성 대 중, 현 중 근
- 감 수: 정 원 혁
- Contents 관련문의: mjkim@feelanet.com



- 발 행: 한국마이크로소프트(유)
- 제 작: ㈜필라넷
- 초판 발행일: 2006년 4월 1일

Microsoft
# SQL Server 2005
## 개발자 가이드

Microsoft
한국마이크로소프트(유)

서울특별시 강남구 대치동 892번지 포스코센터 서관 5층
고객지원센터 : 1577-9700
인터넷 : http://www.microsoft.com/korea/sql

SQL-200603-02

# 서문

첫 아이를 봤을 때, 그 아이가 세상에 나오기만을 고대하는 마음을 기억합니다. 물론 어찌 비유의 대상이 되겠습니까만, 2005년 11월 7일 샌프란시스코에서 SQL Server 2000이 나온 지 5년 만에 드디어 SQL Server 2005가 출시되었습니다.

베타 테스터를 포함한 전문가들은 이미 오래 전부터 SQL Server 2005의 모습을 보고 있었으며, 출시될 날 만을 기다리고 있습니다. 그래서인지 출시가 계속 지연될 때마다 아쉬움 마음이 가시질 않았습니다.

SQL Server 2005는 우리들의 기대를 져버리리 않을 만큼 훌륭한 기술들과 서비스로 변신을 했습니다. 관리자와 개발자 그리고 BI 분석가에 이르기까지 그 향상된 기술의 넓이와 폭은 상당하다고 말할 수 있습니다.

SQL Server 6.5에서 7.0으로 넘어갈 때의 그 놀라움을 넘어선 새로운 변화와 시도가 SQL Server 2005를 통해서 나타납니다.

SQL Server 2005 New Features 가이드 북은 이러한 기술들과 서비스들의 핵심을 요약하고 정리해서 작성되었습니다. SQL Server 2005를 알고자 하는 모든 사용자들에게 좋은 참고 자료가 될 수 있기를 기대하면서, 많은 IT 비즈니스 현장에서 SQL Server 2005를 통해 가치를 창조할 수 있었다는 행복한 이야기들을 들을 수 있게 되기를 기원합니다.

2006. 3. ㈜ 필라넷 DB 사업부

# 목차


서문 ····· 2

SQL Server 2005 소개 ····· 8

- SQL Server 2005 도입효과 ····· 9
- SQL Server 2005 구성요소 ····· 10
- SQL Server 2005의 새로운 기능 ····· 13
- Microsoft SQL Server 2005 에디션별 특징 ····· 14
- SQL Server 2005 관련자료 ····· 17

SQL Server 2005와 .NET의 통합 ····· 19

- 공용 언어 런타임 (CLR, Common Language Runtime) ····· 19
- SQLCLR, .NET Framework과의 통합 ····· 20
- SQLCLR 구성 개요 ····· 21
- CLR 함수 작성 예제 ····· 22
- SQL Server에 어셈블리/함수 카탈로그 등록 ····· 24
- SQLCLR 통합으로 인한 이득 ····· 26
- .NET 코드 vs. T-SQL ····· 27
- SQLCLR 메타데이터 ····· 28
- SQLCLR 모니터링 ····· 29

Transact-SQL Enhancements ····· 30

- TOP 연산자 ····· 30
- TABLESAMPLE 절 ····· 31
- CTE (Common Table Expressions) ····· 32
- 큰 값 데이터 형식 ····· 37

T-SQL 오류 처리 ····38
DDL 트리거 ····40
이벤트 알림(Event Notifications) ····46
스냅숏 격리(Snapshot Isolation) ····50
순위 함수들 ····51
새로운 관계 연산자들: PIVOT, UNPIVOT, 그리고 APPLY ····55
OUTPUT 절 ····58
BULK 행 집합 공급자(BULK Rowset Provider) ····60
새로운 선언적 참조 무결성 동작 ····62
메타데이터 뷰와 동적 관리 뷰(Dynamic Management Views) ····64
기타 ····65

ADO.NET 2.0 ····66
ADO.NET 2.0 주요 기능 리스트 ····67
새로운 연결(Connection) 구성 방법 ····67
Server Enumeration 기능 ····68
연결 풀링(Connection Pooling) 향상 ····69
일괄처리 업데이트(Batch Update) ····70
비 동기적인 명령 실행 ····72
대량 입력(Bulk Import) ····73
SQL Native Client ····74
데이터베이스 미러링 - 클라이언트 장애 조치 ····75
다중 결과 집합
(MARS, Multiple Active Result Sets) ····76
쿼리 알림(Query Notification) ····79
암호 변경/만료 ····80
스냅숏 격리(Snapshot Isolation) ····81

SQL Server Service Broker ····82
SOA 란? ····82
SOA 용어 ····84
Service Broker의 기능 ····85
Service Broker의 시스템 기술구조 ····87
Service Broker 대화 기술구조 ····89
Service Broker 보안 기술구조 ····91
데이터베이스에서 Service Broker 활성화 ····92
Service Broker 구현 ····93
Notification Service ····106
Notificaiton Service 기술구조 소개 ····106
Notification Service 프로세스 ····110
Notification Service 솔루션 구현 ····113

웹서비스와 네이티브 HTTP 지원 ····128
웹서비스와 SOAP이란? ····128
네이티브 HTTP 지원 ····130
네이티브 HTTP를 사용하는 이유 ····131
네이티브 HTTP 기술구조 ····131
네이티브 HTTP 지원 구성 ····132
.NET 기반 HTTP 엔드포인트 클라이언트 개발 ····142

XML 개선기능 ····148
FOR XML 절 개선 ····148
OPENXML 함수 개선기능 ····152
XML 데이터형 사용 ····156
형식화되지 않은 XML 사용방법 ····158
XML 스키마 관리 ····161
형식화된 XML 사용방법 ····164

XML 인덱스 관리 ····· 167
XQuery 지원기능 ····· 170
XQuery 구문 ····· 170
FLOWR 문장 ····· 171
XQuery 표현식 사용예제 ····· 172
XML 데이터형에서 제공하는 메서드를 사용하여
쿼리실행 ····· 178
Modify 메서드를 사용하여 XML 데이터 변경 ····· 180
Nodes 메서드를 사용하여 XML 데이터
부분추출 ····· 183

SMO(SQL Management Objects) 지원 ····· 187
SMO 란? ····· 187
SMO와 SQL Server 분산 관리 개체(SQL-DMO)
비교 ····· 189
SMO 개체모델 ····· 190
SMO 개체 참조 ····· 192
서버속성조회 ····· 194
SMO 개체 생성 ····· 197
SMO를 사용하여 기존 개체 변경 ····· 200
SMO를 사용한 어플리케이션 개발 및
서버정보조회 ····· 202
SMO를 사용하여 데이터베이스 백업 ····· 204
SMO를 사용하여 데이터베이스 목록 생성 및
새 데이터베이스 생성 ····· 206

RMO(Replication Management Objects) 지원 ····· 209
RMO 란? ····· 209
RMO 서버에 연결 ····· 210
복제 관리 작업 ····· 211

향상된 전체 텍스트 검색(Full Text Search) ····· 212
전체 텍스트 카탈로그 백업 및 복원 ····· 212
데이터베이스 연결 및 분리 작업에 전체
텍스트 카탈로그 포함 ····· 213
XML 데이터의 전체 텍스트 인덱싱 ····· 213
전체 텍스트 인덱싱 성능 개선 및 업그레이드 ····· 213
병렬 서비스 보안 ····· 214
다양한 상태 보고 ····· 214

SQL Server 2005의 향상된 보안기능 ····· 215
SQL 로그인 계정에 대한 암호정책 가능 ····· 216
인가 (계층적인 인가 방식) ····· 219
사용자와 스키마의 분리 ····· 223
메타데이터에 대한 제한적인 접근 ····· 231
모듈 실행 컨텍스트 ····· 234
암호화 지원 ····· 237
SQL Server 2005 데이터베이스 엔진의 보안 값
설정 ····· 239

# SQL Server 2005 소개

Microsoft SQL Server 2005는 엔터프라이즈 환경에 사용할 수 있는 데이터 관리 및 분석 어플리케이션에 강력한 보안, 확장성 및 가용성을 제공하며, 어플리케이션을 더욱 쉽게 구축, 배치 및 관리할 수 있도록 돕는 차세대 데이터 관리 및 분석 솔루션입니다.

SQL Server 2005를 통해, 각 기업에서는 데이터를 기반으로 신속하게 의사결정을 내릴 수 있고, 개발 인력의 생산성과 유연성을 향상시키며, IT 부문의 총소유비용을 절감하고, 지속적으로 증가하는 비즈니스 요구사항을 충족시킬 수 있습니다.
SQL Server 2000의 강점을 기반으로 구축된 SQL Server 2005는 모든 규모의 기업들에게 다음과 같은 이점을 제공하는 통합 데이터 관리 및 분석 솔루션입니다.

- 보안, 확장성 및 안정성이 더욱 강화된 엔터프라이즈 어플리케이션의 구축, 배치 및 관리
- 데이터베이스 어플리케이션 구축, 배치 및 관리의 복잡성을 해소함으로써 IT 생산성 극대화
- 다수의 플랫폼, 어플리케이션 및 장치 간 데이터 공유 통해 내부 및 외부 시스템에 더욱 쉽게 연결
- 성능, 가용성, 확장성, 보안의 침해없이 총소유비용 통제

## SQL Server 2005 도입효과

SQL Server 2005 데이터 플랫폼은 모든 규모의 조직에 다음과 같은 도입효과를 제공합니다.

| 도입효과 | 설명 |
|---|---|
| 데이터 자산 활용 | SQL Server 2005는 업무용 및 분석 어플리케이션을 위한 안전하고 신뢰할 수 있는 데이터베이스를 제공합니다. 또한, 고객이 보유하고 있는 데이터의 가치를 최대한 활용할 수 있도록 하기 위해, 강력한 리포팅, 분석 및 데이터 마이닝 등과 같은 기능을 제공합니다. |
| 생산성 향상 | SQL Server 2005는 비즈니스 인텔리전스 기능과 Microsoft Office System 과 같은 친숙한 도구와의 통합을 통해, 조직 전반의 정보 근로자가 필요로 하는 최신 비즈니스 정보를 제공합니다. Microsoft의 목표는 조직 내 모든 정보 사용자에게 확장된 비즈니스 인텔리전스 기능을 제공하고, 조직내 가장 중요한 자산인 데이터를 기초로 보다 나은 비즈니스 의사결정을 내릴 수 있도록 하는 것에 있습니다. |
| IT 복잡성 해소 | SQL Server 2005는 개발자를 위해, 유연한 개발 환경을 제공하고, 데이터베이스 관리자을 위해 자동화된 통합 관리 도구를 지원하기 때문에 업무용 및 분석 어플리케이션의 개발, 구축, 관리 작업을 보다 용이하게 수행할 수 있습니다. |
| 저렴한 총소유비용(TCO) | 사용자 편이성 및 배포 용이성에 초점을 맞춘 통합적인 접근 방법을 통해, 기초투자, 구현 및 유지 보수 비용을 업계 최저 수준으로 줄일 수 있으며, 데이터베이스 투자에 대한 ROI 를 신속하게 회수할 수 있습니다. |

## SQL Server 2005 구성요소

SQL Server 2005는 엔터프라이즈 데이터관리 및 BI(Business Intelligence) 어플리케이션을 위해 뛰어난 보안, 안정성 및 생산성을 갖춘 플랫폼을 제공합니다. SQL Server 2005는 정보 근로자 뿐만 아니라 IT 전문가들에게 강력하고 친숙한 도구를 제공하며, 모바일 장치에서 엔터프라이즈 데이터 시스템에 이르기까지, 다양한 플랫폼에서 엔터프라이즈 데이터관리 및 분석 어플리케이션을 구축, 배포, 관리, 운영하기 위한 업무적인 복잡성을 줄여 줄 수 있습니다. SQL Server 2005에서 제공하는 광범위하고 다양한 기능과, 기존 시스템과의 상호 운용성, 일상 업무의 자동화를 통해, 모든 규모의 기업에서 완벽한 데이터 솔루션으로 사용될 수 있습니다.

| 구성요소 | 설명 |
|---|---|
| 관계형 데이터베이스 엔진<br>(Relational Database Engine)<br>  | SQL Server 관계형 데이터베이스 엔진은 SQL Server 2005의 핵심으로 관계형 또는 XML형식의 데이터를 저장하고,추출하고,수정하는데 있어 탁월한 성능과 확장가능하고 안전한 환경을 제공하는 강력한 관계형 데이터 베이스 엔진입니다. |
| 분석 서비스<br>(SQL Server Analysis Services)<br>  | 온라인 분석 처리 어플리케이션(OLAP) 과 데이터 마이닝을 지원하는 강력한 비즈니스 인텔리전스 솔루션입니다. |
| 통합서비스<br>(Integration Services, or SSIS)<br>  | 데이터를 가져오고 내보내는 솔루션으로서 데이터의 이동이 이루어질 때 변환과정을 수행합니다 |

| | |
|---|---|
| 알림서비스<br>(Notification Services)<br>  | 이벤트와 요청된 데이터에 근거하여 이메일,텍스트 메시지 기타 다른 방식으로 알림(notifications)을 발생시킬 수 있습니다 |
| 리포팅 서비스<br>(Reporting Services)<br>  | 데이터 원본으로부터 데이터를 추출하여 보고서를 만들고 브라우저로 볼 수 있게 하거나 파일로 내보내거나 이메일로 보낼 수 있습니다 |
| 서비스 브로커<br>(Service Broker)<br>  | 소프트웨어 서비스간의 메시지기반 통신에 관한 서비스 입니다 |
| 네이티브 http 서비스<br>(Native HTTP Service)<br>  | Microsoft Windows Server™ 2003에 설치되어 있을시 SQL Server 2005는 HTTP (Hypertext Transfer Protocol)로 이루어진 요구에 응답할 수 있습니다.<br>Native HTTP Service 는 SQL Server 2005가 IIS (Microsoft Internet Information Services)없이도 웹서비스 인터페이스를 만들수 있도록 하여 줍니다. |
| SQL Server 에이전트<br>(SQL Server Agent)<br>  | 데이터베이스 유지 및 작업,이벤트,경고 관리를 자동화 하도록 하는 예약 관리 업무 엔진입니다. |

| | |
|---|---|
| 닷넷 CLR 기반 서비스<br>.NET Common Language Runtime<br>  | CLR(Common Language Runtime)이 SQL Server 에 내재되어 있어서 데이터베이스 솔루션이 Microsoft Visual C#® .NET 또는 Microsoft Visual Basic® .NET 과 같은 언어에서 생성된 관리 코드 (managed code)를 이용할 수 있도록 합니다. |
| 복제<br>(Replication)<br>  | 한쪽 데이터베이스에서 다른 데이베이스로 데이터 및 데이터베이스 객체들을 복사 이동시키고난 후 데이터베이스간의 일관성이 동기화되도록 하여줍니다. |
| 전체 텍스트 검색<br>(Full Text Search)<br>  | SQL Server 데이터베이스에 있는 텍스트로 저장된 키워드 기반 쿼리에 대한 빠르고 유연한 인덱스 검사를 가능하게 하여 줍니다 |

## SQL Server 2005의 새로운 기능

### ■ 데이터베이스 개발측면

| 주요특징 | 설명 |
|---|---|
| .NET Framework 호스팅. | SQL Server 2005를 통해 개발자들은 Microsoft Visual C# .NET 및 Microsoft Visual Basic .NET과 같은 친숙한 언어를 이용해 데이터베이스 개체를 만들 수 있습니다. 또한 개발자는 사용자 정의 형식과 집계 등 두 가지 새로운 개체를 만들 수 있습니다. |
| XML 기술 | XML은 로컬 네트워크와 인터넷을 통해 여러 다양한 어플리케이션에 데이터를 배포할 수 있도록 지원하는 중요한 표준입니다. SQL Server 2005는 네이티브(native) XML 문서 저장소 및 쿼리를 지원합니다 |
| ADO.NET Version 2.0 | SQL Server 2005의 ADO.NET은 새로운 SQL 형식 지원에서 MARS(Multiple Active Result Sets)에 이르기까지 데이터 집합 액세스와 처리 기능을 개선하여 보다 뛰어난 확장성과 융통성을 발휘합니다. |
| 향상된 보안 기능 | SQL Server 2005의 새로운 보안 모델은 사용자들을 개체에서 분리하고 세분화된 액세스 권한을 부여하며 데이터 액세스에 대한 한층 강화된 제어 기능을 제공합니다. 또한, 모든 시스템 테이블이 뷰로 구현되기 때문에 데이터베이스 시스템 개체를 더욱 강력하게 제어할 수 있습니다. |
| Transact-SQL 기능 향상 | 확장 가능한 데이터베이스 어플리케이션을 개발하기 위한 새로운 언어 기능들이 제공됩니다. 이들 향상된 기능에는 오류 처리, 새로운 재귀 쿼리 기능, 관계형 연산자 PIVOT, APPLY, ROW_NUMBER 및 기타 행 순위 지정 함수 등이 있습니다. |
| SQL Service Broker | SQL Service Broker는 대규모 업무용 어플리케이션을 위한 비동기식 분산 어플리케이션 프레임워크를 제공합니다. |

| | |
|---|---|
| Notification Services | 알림 서비스를 사용하여 주식 정보, 뉴스 구독, 소포 배달 알림, 항공권 가격 등과 같은 개인화된 최신 정보를 모든 장치에 전달하는 풍부한 알림 어플리케이션을 작성할 수 있습니다.<br>웹 서비스 개발자들은 SQL Server 2005를 통해 데이터베이스 계층에서 웹 서비스를 개발함으로써 SQL Server를 웹 서비스 중심 어플리케이션을 위해 새로운 유형의 데이터 액세스 기능을 제공하는 HTTP 수신기로 만들 수 있습니다. |
| 리포팅서비스<br>(Reporting Services) | SQL Server 2005를 이용한 Reporting Services에서는 Visual Studio 2005에 포함된 보고서 컨트롤을 제공합니다. 통합된 보고 컨트롤은 엔터프라이즈 어플리케이션에 대한 향상된 보고 기능을 제공합니다. |
| 전체 텍스트 검색 향상 | SQL Server 2005는 풍부한 기능의 전체 텍스트 검색 어플리케이션을 지원합니다. 카탈로그 작성 기능은 카탈로그로 작성할 대상을 지정하는 데 있어 한층 높은 융통성을 발휘할 수 있도록 향상되었습니다. 또한 쿼리 성능과 확장성이 대폭 향상되었으며 새로운 관리 도구를 통해 전체 텍스트 구현을 보다 심층적으로 파악할 수 있습니다. |

## Microsoft SQL Server 2005 에디션별 특징

SQL Server 2005는 다양한 규모의 기업의 요구 사항에 가장 부합하는 솔루션을 선택할 수 있도록 유연한 라이센스 전략을 제공하며, 즉시 사용가능한 통합 데이터관리 플랫폼을 구축하기 위해 필요한, 데이터 저장소, 관리, 분석, 리포팅 기능이 단일 제품에 포함되어 있습니다.

초소형 기업에서 초대형 기업까지 확장할 수 있도록 설계된 SQL Server 2005는 모든 고객에게 동일한 성능, 보안, 안정성 및 비즈니스 가치를 제공합니다. SQL Server 2005는 멀티테라바이트 데이터웨어하우스에서부터 SQL Server CE에디션 기반 포켓 PC(Pocket PC)에 이르는 다양한 구현을 지원합니다.

## ■ 에디션별 특징

| 에디션 | 이점 | 크기 | 주요 기능 |
|---|---|---|---|
| Express (무료) | 간단한 데이터 중심 어플리케이션을 배우고 구축 및 배포하는 가장 빠른 방법 | 1개의 CPU<br>1-GB RAM<br>4-GB DB 크기 | **간단한 관리 도구 간단한 보고**<br>복제 및 SSB 클라이언트 |
| Workgroup | 소규모 부서와 성장하는 기업을 위한 가장 합리적인 가격대의 사용하기 쉬운 데이터베이스 솔루션 | 1-2개의 CPU<br>3-GB RAM | Management Studio<br>가져오기/내보내기 제한된<br>복제 게시 클러스터링<br>**백업 로그 전달** |
| Standard | 중견 기업 및 대규모 부서를 위한 완벽한 데이터 관리 및 분석 플랫폼 | 1-4개의 CPU<br>무제한 RAM | **데이터베이스 미러링**<br>기본적인 ETL<br>Analysis Services가 지원되는 표준 OLAP 서버<br>Reporting Services가 지원되는 표준 보고 Data Mining<br>완전 복제 및 SSB 게시<br>원시 32 및 64비트 에디션에서 사용 가능<br>Itanium2 및 x64 지원 |

| | | | |
|---|---|---|---|
| Enterprise | 비즈니스 크리티컬 엔터프라이즈 어플리케이션을 위한 완전히 통합된 데이터 관리 및 분석 플랫폼 | **무제한 확장 및 파티셔닝** | **향상된 데이터 베이스 미러링, 완벽한 온라인 및 병렬 조작, 데이터베이스 스냅샷**<br>전체 OLAP 및 데이터 마이닝을 비롯한 향상된 분석 도구<br>**사용자 정의되고 확장성이 뛰어난 임의(ad hoc) 보고 기능을 통한 향상된 보고 기능**<br>복잡한 데이터 라우팅 및 변환 기능을 통한 향상된 ETL<br>원시 32 및 64비트 에디션에서 사용가능<br>Itanium2 및 x64 지원 |

***[참고]***
*굵은 글꼴은 Microsoft SQL Server 2005에 새로 추가된 기능을 나타냅니다. 각각의 상위 에디션에는 하위 에디션과 동일한 기능이 포함되어 있습니다.*

## SQL Server 2005 관련자료

- Microsoft SQL Server 홈 페이지
  http://www.microsoft.com/sql

- Microsoft SQL Server TechCenter
  http://www.microsoft.com/technet/prodtechnol/sql/default.mspx

- Microsoft SQL Server Developer Center
  http://msdn.microsoft.com/SQL

- Microsoft SQL Server 신제품발표회 홈페이지
  http://www.ready2005.com/

- SQL Server 2005 에디션별 기능 비교
  http://www.microsoft.com/sql/2005/productinfo/sql2005features.asp

- 가상 Hands-On Labs
  http://msdn.microsoft.com/SQL/2005/default.aspx

- SQL Server 2005 E-Learning
  https://www.microsoftelearning.com/sqlserver2005/

- Microsoft SQL Server TechNet 교육용 웹캐스트
  http://www.microsoft.com/events/series/technetsqlserver2005.mspx

- Microsoft SQL Server MSDN 교육용 웹캐스트
  http://msdn.microsoft.com/SQL/2005Webcasts/

- 비즈니스 인텔리전스 정보
  http://msdn.microsoft.com/SQL/sqlwarehouse/SSIS/default.aspx

- SQL Server 2005 Express 와 XM
  http://lab.msdn.microsoft.com/express/sql/

- SQL Server 2005 관련 백서
  http://www.microsoft.com/sql/2005/techinfo/default.asp

- SQL Server 2005 가격정책 및 라이센스 정책
  http://www.microsoft.com/sql/howtobuy/default.asp

# SQL Server 2005와 .NET의 통합

마이크로소프트가 .NET 이라고 불리는 개발 플랫폼을 발표한지도 5년여가 지났으며, .NET Framework은 윈도우즈, 웹, 그리고 모바일 어플리케이션들을 아울러서 혁신적인 차 - 세대 개발 플랫폼으로 발전시켜 나아가고 있습니다.

더블어 .NET에 관련된 기술들은 마이크로소프트의 많은 소프트웨어와 서비스의 적용되는 기반 기술이 되고 있습니다. SQL Server 또한 예외가 아닙니다. 어쩌면 .NET Framework를 가장 포괄적으로 통합시킨 제품이 아닐까 합니다.

SQL Server 2005는 이제 .NET Framework과 통합되었습니다. 데이터베이스 사용자 혹은 개발자들은 C# 이나 VB 같은 .NET 언어를 사용해서 저장 프로시저, 함수, 트리거, 사용자 - 정의 집계, 그리고 사용자-정의 형을 작성하고 이를 SQL Server에서 기존의 데이터베이스 개체처럼 사용할 수 있도록 지원됩니다.

이번엔 데이터베이스 관리자 측면에서 .NET Framework과의 통합이 가져오는 의미와 그 내용 그리고 기타 관련된 사항들을 간략하게 살펴보고 SQL Server 2005에서 어떻게 적용해 나갈 것인지 판단하는데 도움이 되고자 합니다.

## 공용 언어 런타임(CLR, Common Language Runtime)

CLR은 .NET 코드의 실행 환경을 제공하는 Framework 핵심부입니다. .NET 코드로 작성된 어플리케이션을 실행하는데 있어서 필요한 많은 핵심 서비스들을 제공하는 것입니다. 예를 들어, 메모리 관리, 개체 생존주기 관리, 쓰레드 관리, 타입 안전성 검사, 보안, 그리고 I/O 관리 등등에 해당합니다. 이러한 CLR 실행 환경하에서 실행되는 코드들을 흔히 관리 코드(Managed Code)라고 부릅니다. 반대로 .NET 에서 Win32 API 나 COM 오브젝트를 호출하는 경우 비 관리 코드(Unmanaged Code)라고 부릅니다. Win32 API나 COM 기반 코드는 CLR에 의해서 제어되지 않기 때문입니다.

CLR은 또한 다른 프로그램에서 의해서 호스트(Host)되어질 수 있습니다. 즉 프로그램에서 .NET 런타임을 로드하고(Hosting, 혹은 Host 프로그램), .NET 관리 환경하에서 코드를 실행하는데 사용하는 것입니다. 그 대표적인 호스트 프로그램이 바로 Internet Explorer, ASP.NET, 그리고 SQL Server 2005에 해당합니다.

## SQL CLR, .NET Framework과의 통합

SQL Server 2000 버전까지 데이터베이스의 개발과 관리를 위한 주된 언어는 Transact-SQL(T-SQL) 이었습니다. T-SQL로는 해결할 수 없는 경우 예를 들어, 운영체제 수준의 기능이나 암호화와 같은 별도의 서비스 기능이 요구되는 경우엔 C++ 과 개방형 데이터 서비스(ODS, Open Data Service) API를 사용한 확장 저장 프로시저(Extended Stored Procedure)를 개발해서 구현이 가능했었습니다. 반면에 확장 저장 프로시저가 SQL Server 내부 프로세스 공간에서 직접 수행되므로 신뢰성과 안정성 측면에서 문제가 될 수 있었습니다.

SQL Server 2005에서도 물론 가장 주된 언어는 T-SQL 입니다. 더욱이 T-SQL은 많은 향상 기능들로 인해서 그 활용도와 중요성은 더욱 강조가 될 것입니다. 그리고 이제 SQLCLR을 통해서 T-SQL 을 확장시키기 위한 보다 안전하고, 신뢰할 수 있는, 효율적이고도 쉬운 방법이 새로 제공이 됩니다.

■ .NET Framework과 SQL Server 2005의 통합으로 제공되는 기능

- .NET 언어를 사용한 함수, 프로시저와 트리거 구현
- T-SQL 내장 함수 라이브러리를 보다 쉽게 확장
- SQL Server 외부 데이터 접근을 위한 보다 쉽고 효율적인 방법 제공
- T-SQL 보다 빠른 프로시저 로직과 연산 처리
- 투명한 구현 방법 - 기존 T-SQL 개체와 동일한 사용 방법을 제공

■ 또한 두 가지 새로운 확장성 기능을 제공합니다.

- 스칼라 타입 시스템을 확장할 수 있는, 사용자 - 정의 형(UDT, User-Defined Types)
- 집계 프레임워크를 확장할 수 있는, 사용자 - 정의 집계(UDA, User-Defined Aggregates)

## SQLCLR 구성 개요

SQLCLR을 통해서 CLR 모듈(함수, 프로시저 등)을 작성하고 SQL Server 2005로 등록 및 호출하기 위한 기본 구성은 다음의 절차를 따릅니다.

■ SQLCLR 구성 절차

- Visual Studio 2005에서 SQL Server Project 형식으로 필요한 CLR 모듈을 작성
- 빌드를 통해서 어셈블리를(*.DLL) 생성한 후 SQL Server 자동 배포하거나 혹은 관련 DDL 구문을 사용해서 어셈블리와 해당 오브젝트를 직접 등록
- T-SQL에서 일반 개체와 동일하게 호출 및 사용

***[참고]***
*현재 버전에서, SQLCLR은 서버 속성으로 정의되어 있으며, 기본적으로 Disabled 상태입니다. 어셈블리와 CLR 모듈을 등록하는 것은 관계가 없지만, 실행을 위해서는 해당 속성을 Enabled로 설정해야만 합니다.*
*기본적으로 두 가지 방법을 사용할 수 있습니다.*
*1) sp_configure 를 사용해서 설정하는 방법*

```
EXEC sp_configure 'clr enabled', 1
RECONFIGURE
```

*2) SQL Server 구성 도구 중의 하나인, SQL Server Surface Area Configuration 툴을 사용하는 방법*

그러면, CLR 함수를 하나의 예로 해서, SQLCLR를 만들어가는 과정을 순서대로 살펴보도록 하겠습니다.

## CLR 함수 작성 예제

간단히 다음 절차에 따라서 CLR 작성합니다.

1. Visual Studio 2005에서 새로운 프로젝트(File | New Project)를 시작합니다. 예제는 C#으로 구성했습니다.
2. 프로젝트 타입은 데이터베이스(Database) | SQL Server Project 이며, 프로젝트 이름은 SQLCLRExample 이라고 해서 특정 폴더에 지정합니다.
3. 데이터베이스 참조 추가(Add Database Reference) 대화 상자에서 자동 배포에서 적용할 데이터베이스를 선택합니다. 예제에서는 AdventureWorks 를 선택했습니다.
4. 프로젝트(Project) 메뉴 | 사용자 정의 함수 추가(Add User-Defined Function) 메뉴를 선택해서, 함수를 추가합니다.
5. 새 항목 추가(Add New Item) 대화상자에서 사용자 - 정의 함수가 선택되고, 이름은 Factorial.cs 라고 지정합니다.
6. Factorial.cs 에 Factorial 함수를 추가 작성합니다. 소스 코드는 다음과 같습니다.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class UserDefinedFunctions
{
    [Microsoft.SqlServer.Server.SqlFunction]
    public static SqlInt32 Factorial(SqlInt32 Number)
    {
        if (Number < 1)
            return 1;
        else
```

```
                                    return Number * Factorial(Number - 1);
        }
};
```

7. 빌드(Build) 메뉴에서 프로젝트를 빌드합니다(Ctrl-Shift-B). 빌드를 수행한 결과로 어셈블리 파일(SQLCLRExample.dll)이 생성됩니다.
8. 해당 어셈블리를 SQL Server 2005로 불러와서 등록하고 어셈블리 내의 메서드를 T-SQL 함수로 연결하는 작업을 수행할 수 있습니다. 이 때 개발자와 SQL Server 관리자 측면에서 두 가지 방법을 사용할 수 있습니다. 이 두 가지 방법의 차이점을 이해하는 것도 중요한 내용이 될 것입니다.
   A. Visual Studio 2005 개발자는 IDE에서 제공하는 자동 배포 기능을 사용할 수 있습니다. 빌드(Build) 메뉴에 있는 Deploy 메뉴를 이용하면 어셈블리를 빌드, 등록, T-SQL 개체로의 연결 작업이 자동으로 수행됩니다. 아주 편리하지만 실제 내부에서 수행하는 작업 방식에는 부담스러운 부분이 많이 있습니다. 실제 수행 방법을 보고자 한다면, SQL 프로필러를 작동시키고 서버에서 수행되는 작업을 추적해 보면 알 수가 있습니다. 그 내용을 간단하게 정리를 하면, 우선 기존의 연결된 TSQL 개체 즉 저장 프로시저, 함수, 트리거 등을 제거하고, 다음에 어셈블리를 제거합니다, 그 어셈블리를 다시 로드한 다면, 어셈블리 내에 정의된 T-SQL 개체를 다시 불러들여서 재 정의(연결) 합니다. 이 때 Visual Studio 2005가 데이터베이스에 추가하는 파일이 해당 어셈블리 파일 뿐만 아니라 관련 소스 코드, 디버거 용 파일(*.pdb) 등을 모두 추가하게 됩니다. 이는 카탈로그 뷰 sys.assembly_files 를 실행해서 확인할 수 있습니다. 전체 파일이 등록되는 것을 조정하고 싶다면, 프로젝트의 옵션을 변경을 하거나 T-SQL DDL 구문을 사용해서 직접 작업을 하는 것입니다.

   B. 위에서 언급한 대로, 어셈블리와 CLR 모듈에 대한 작업은 T-SQL DDL 구문을 직접 사용해서 처리하는 것이 최적화 됩니다. 다음에서 관련된 DDL 구문과 예제를 통해서 작업 방법을 살펴보겠습니다.

## SQL Server에 어셈블리/함수 카탈로그 등록

SQL Server 사용자는 물론이고, SQL Server 관리자에겐 특히 .NET 어셈블리와 CLR 모듈에 대한 관련 작업들은 DDL 구문을 이용해서 직접 작업하는 것이 권장됩니다.

어셈블리 등록 작업은 CREATE ASSEMBLY 구문을 사용합니다.

***[참고]***
*CREATE ASSEMBLY assembly_name*
*[ AUTHORIZATION owner_name ]*
*FROM { 〈client_assembly_specifier〉 | 〈assembly_bits〉 [ ,...n ] }*
*[ WITH PERMISSION_SET = { SAFE | EXTERNAL_ACCESS | UNSAFE } ]*
*[ ; ]*

Visual Studio 에서 배포(Deploy)를 통해서 등록되는 경우 FROM 절의 〈assembly_bits〉 로 직접 처리가 되는 반면, T-SQL로 작업 시에는 〈client_assembly_specifier〉를 사용해서 어셈블리 파일 경로만 지정하면 내부적으로 바이너리 코드를 읽어 들여서 저장하게 됩니다. 따라서 어셈블리가 데이터베이스에 등록되고 나면 그 파일은 필요 없어집니다. 그러나 해당 소스 코드나 필요한 파일들은 ALTER ASSEMBLY...ADD FILE 구문을 사용해서 데이터베이스에 저장해 두는 것을 권장합니다. CREATE 나 ALTER ASSEMBLY 구문에서는 WITH PERMISSION_SET 절에서 권한 관련 옵션을 선언할 수 있습니다.

■ 어셈블리에 대한 세 가지 권한 옵션은

- SAFE - 가장 제한적인 권한 집합으로, 어셈블리 내의 코드는 파일, 네트워크, 또는 레지스트리와 같은 외부 자원에 접근할 수 없습니다.
- EXTERNAL_ACCESS - 파일, 네트워크, 또는 레지스트리와 같은 외부 자원에 접근이 가능합니다.
- UNSAFE - 제한이 없습니다. 또한, .NET 이 아닌 비 관리 코드도 호출될 수 있습니다. 가장 위험한 권한 집합이므로 신뢰할 수 있는 어셈블리에 대해서 부여되어야 합니다.

■ 어셈블리가 등록되고 나면, CLR 모듈을 작성하고 어셈블리 내 메서드(혹은 타입)와 연결하는 작업을 수행합니다. 이는 기존의 T-SQL 개체에서 사용하는 구문들 예를 들어 CREATE PROC, CREATE FUNCTION 등과 동일하며 다만, AS EXTERNAL NAME 절에서 어셈블리 이름, 클래스 이름, 그리고 메서드 이름을 사용 연관된 CLR 모듈과 메서드를 지정해 주면 됩니다.

### [따라하기] T-SQL 구문을 사용한 CLR 모듈 등록

```
USE AdventureWorks
GO

-- 어셈블리 등록
-- 어셈블리 파일이 C:\Temp 에 있다고 가정합니다.
CREATE ASSEMBLY SQLCLRDemo
FROM 'C:\Temp\SQLCLRExample.dll'  -- 실제 어셈블리 경로를 지정
WITH PERMISSION_SET = SAFE
GO

-- 옵션: 필요한 경우 스키마 생성
CREATE SCHEMA CLRDemo
GO

-- T-SQL 함수로 연결
CREATE FUNCTION CLRDemo.uf_Factorial(@Number int)
RETURNS int
AS
EXTERNAL NAME SQLCLRDemo.UserDefinedFunctions.Factorial
GO

-- 함수 호출 및 테스트
```

```
SELECT CLRDemo.uf_Factorial(3)
SELECT CLRDemo.uf_Factorial(10)
GO
```

DROP ASSEMBLY 구문을 사용해서 어셈블리를 제거할 수 있습니다. 이 때 종속된 모든 T-SQL 개체가 먼저 제거되어야 합니다.

## SQLCLR 통합으로 인한 이득

데이터베이스 엔진 내에서 .NET 코드의 실행을 지원함으로써 얻을 수 있는 이득은 다음과 같습니다.

| 특징 | 설명 |
|---|---|
| 향상된 프로그래밍 모델 | ADO.NET을 사용하는 경우, 어플리케이션과 데이터베이스에 단일 언어를 사용해서 유사한 코드로 개발할 수 있으며 서버에도 개체 지향적 프로그램 방법을 적용할 수 있습니다. 또한 구조적 예외 처리와 같은 기능들을 접목할 수 있습니다. |
| .NET Framework 기본 클래스 라이브러리 활용 가능 | .NET Framework이 제공하는 방대한 분량의 클래스 라이브러리를 사용해서, 전문적인 산술 연산, 통계, 암호화, XML 및 문자열 처리 등을 손 쉽게 처리할 수 있습니다. |
| 성능 | T-SQL 은 인터프리터 언어인 반면, .NET 코드는 처음 실행될 때 JIT(Just-In-Time) 컴파일러와 캐시 처리를 통해서 보다 나은 성능을 제공할 수 있습니다. |
| 확장 저장 프로시저 대체 | 앞서 언급한 대로, 기존 확장 저장 저장 프로시저의 문제점인 안정성과 신뢰성을 제공할 수 있으며 지금까지 언급한 .NET 관리 코드의 이득을 취할 수 있습니다. |

| | |
|---|---|
| 안전성과 신뢰성 | SQLCLR 은 기본적으로 Disabled 상태입니다.<br>어셈블리에 대한 권한 제어를 통해서 코드 보안을 강화할 수 있으며, SQL Server 가 .NET CLR의 호스트로서 메모리, 쓰레드 관리, 동기화 등을 제어합니다.<br>또한 .NET의 HPA(Host Protection Attributes)를 사용함으로 잘못된 시스템 자원의 접근으로 인한 문제에서 보호할 수 있습니다. 그 외에도 다양한 모니터링, 추적 방법을 통해서 성능 및 안전성 관리가 가능합니다. |

## .NET 코드 vs. T-SQL

이제 SQL Server 개발자의 양 손의  두 가지 도구가 제공이 됩니다. T-SQL 언어가 관계형 데이터베이스 주된 언어임에는 틀림이 없지만 활용할 수 있는 보다 폭 넓은 도구가 가능한 만큼 그 최적 사례에 따라서 좋은 결과를 만들 수 있을 것입니다. 두 가지 접근 방법의 장점들을 간단히 비교하면 다음과 같이 정리할 수 있습니다.

■ 데이터 액세스 중심적인 코드

- Transact-SQL 이 한증 더 유리합니다.
- .NET 런-타임을 로드할 필요가 없으며, 데이터 계층에 직접 액세스가 수행됩니다.
- 절차적 프로그래밍을 통해서 데이터 처리에 보다 밀접한 작업을 수행합니다.

■ 비-데이터베이스 코드

- .NET 코드가 더 유리합니다.
- 복잡하거나 전문적인 산술/통계 연산을 수행할 수 있습니다.
- SQL Server 외부의 자원 접근에 보다 강력한 기능을 제공합니다.
- 개체 지향적 프로그래밍 모델을 사용할 수 있습니다.

## SQLCLR 메타데이터

SQL Server 2005에서 CLR 어셈블리와 관련 개체들을 관리하기 위한 여러 가지 카탈로그 뷰들을 제공합니다. 다음은 관련된 내용입니다.

| 카탈로그 뷰 | 설명 |
|---|---|
| sys.assemblies | 데이터베이스에 저장된 어셈블리에 대한 정보 |
| sys.assembly_files | 어셈블리 관련 각 파일들에 대한 정보 |
| sys.assembly_modules | 저장 프로시저, 함수, 트리거와 같은 각 모듈에 대한 정보 |
| sys.assembly_references | 다른 어셈블리를 참조하는 경우의 참조 정보 |
| sys.assembly_types | 사용자-정의 형에 대한 정보 |

## SQLCLR 모니터링

동적 관리 뷰(DMV, Dynamic Management View), SQL 프로필러 추적, 성능 모니터 개체와 관련 카운터 값 등을 통해서 SQLCLR 의 동작을 모니터 할 수 있습니다.

■ 동적 관리 뷰는 sys_dm_clr_* 시작하는 뷰들로 다음과 같습니다.

- sys.dm_clr_appdomains
- sys.dm_clr_loaded_assemblies
- sys.dm_clr_properties
- sys.dm_clr_tasks

■ SQL 프로필러에서는 CLR 이벤트 그룹을 추적할 수 있습니다.

- Assembly Load 이벤트 항목을 추적할 수 있으며, 이벤트는 어셈블리가 로드될 때 발생합니다. 어셈블리 로드에 대한 성공/실패 여부, 관련 성능 문제를 확인할 수 있습니다.

■ 성능 모니터 개체

- SQLServer: CLR - CLR Execution 카운터가 제공됩니다. 이는 CLR 전체 실행 시간(ms)을 반영합니다.

# Transact-SQL Enhancements

SQL Server 2005에서는 .NET 런타임의 호스팅을 통해서 VB.NET이나 C#과 같은 언어로 저장 프로시저, 함수 등의 쿼리 개체 작성이 가능합니다. 이러한 구현은 기존 T-SQL 언어의 부족한 부분을 지원할 수 있으나 궁극적으로 관계 데이터 처리 언어인 T-SQL를 대체하는 것은 아닙니다. 특히, SQL Server 2005 T-SQL은 많은 향상 기능을 제공하고 있습니다. 어떠한 기능들이 추가되고 향상되었는지 살펴보도록 하겠습니다.

## TOP 연산자

그 동안 TOP 연산자에 대해서 아쉬운 점들이 많으셨죠? 특히 웹 어플리케이션에서 페이지를 처리하는 부분 등에 있어서 동적으로 변하는 결과 집합의 수를 처리하기 위해서는 서버 측에서 동적 쿼리 형식을 사용해야 하는 등의 어려움이 있었습니다. 이제는 보다 쉽고 편리하게 그러한 형식의 쿼리를 구현할 수 있습니다. 물론 이것은 익히 잘 알고 있는 성능 관련 문제 들을 배제하고 순수하게 구문 그 자체로서 비교하는 것입니다.

■ SQL Server 2005에서 TOP 연산자의 향상된 기능은
- TOP 구문에서 변수나 서브쿼리 같은 수식 지원
- INSERT, UPDATE 그리고 DELETE 구문 지원

이제 몇 가지 따라 하기를 통해서 향상된 기능을 경험해 보도록 하겠습니다.

### [따라하기] TOP 연산자 향상 기능

1. 변수를 사용한 경우

```
USE AdventureWorks

DECLARE @n AS smallint
SET @n = 5

SELECT TOP(@n) *
FROM Purchasing.PurchaseOrderHeader
ORDER BY OrderDate DESC
```

2. 서브쿼리를 사용한 경우

```
SELECT TOP(SELECT COUNT(*)/DATEDIFF(month, MIN(OrderDate),
MAX(OrderDate))
        FROM Purchasing.PurchaseOrderHeader) *
FROM Purchasing.PurchaseOrderHeader
ORDER BY OrderDate DESC
```

## TABLESAMPLE 절

앞서 살펴 본 TOP 연산자가 결과 집합을 제한하는 방법이라면, SQL Server 2005에 새로 도입된 TABLESAMPLE 은 SELECT 결과 집합을 샘플링으로 선택된 행 집합으로 제한하고자 할 경우에 사용될 수 있습니다. 즉 모든 데이터를 처리할 필요가 없거나, 정확한 결과가 아닌 대략적인 결과를 필요로 하는 경우입니다. 예를 들어, 주문 상세에서 샘플링을 통한 결과 집합에 대해서 대략적인 평균 주문 수량을 보고자 하는 경우 등입니다.

[따라하기] TABLESAMPLE 절

```
USE AdventureWorks

-- 아래는 전체 집합에 대한 평균 주문 수량을 구합니다.
SELECT AVG(OrderQty) FROM Purchasing.PurchaseOrderDetail

-- 아래는 30퍼센트의 샘플링 집합에 대한 평균 주문 수량을 구합니다.
SELECT AVG(OrderQty) FROM Purchasing.PurchaseOrderDetail
        TABLESAMPLE (30 PERCENT)
```

## CTE (Common Table Expressions)

SQL-99에 정의된 CTE는 파생 테이블(Derived Table, 혹은 일반적으로 인-라인 뷰라고 부르는)의 일종으로 생각할 수 있습니다. 그러나 파생 테이블은 외부 쿼리에서 한 번 이상 참조할 수가 없습니다. 이를 위해서 뷰를 만들 수도 있지만 이는 쿼리 개체를 만들고 데이터베이스에 저장하기를 요구합니다. CTE는 이와 같은 두 가지 어려움을 해소할 수 있는 방법을 제공합니다. 즉 파생 테이블처럼 쿼리에 직접 테이블의 표현 식을 작성하면서도 뷰처럼 한 번 이상 참조가 가능한 것입니다. CTE의 또 다른 형식은 바로 재귀적인(Recursive) 호출 기능입니다. 이제 관계 형 데이터 모델의 순환 관계를 처리할 수 있는 구현 방법을 제공하는 것입니다.

■ CTE의 두 가지 형식은

- 비재귀적 CTE 는 파생 테이블 형식으로 테이블 표현을 정의하고 이를 외부 쿼리에서 한 번 이상 참조할 경우에 유용하게 적용될 수 있습니다.
- 재귀적 CTE는 순환 관계 모델을 처리할 경우 기본적으로 사용됩니다.

우선, 비재귀적 CTE를 살펴보시죠.
다음은 비재귀적 CTE의 대략적인 구문 형식과 따라 하기입니다.

### [따라하기] 비재귀적 CTE

1. 구문 형식

```
WITH NonRecursiveCTE(〈column_aliases〉)
AS
(
   〈CTE_쿼리_정의〉
)
SELECT * FROM NonRecursiveCTE ;
```

2. 예제

```
USE AdventureWorks ;

WITH YearlyOrders(orderyear, numorders)
AS
(
  SELECT YEAR(OrderDate), COUNT(*)
  FROM Sales.SalesOrderHeader
  GROUP BY YEAR(OrderDate)
)
SELECT
          CurYear.orderyear
, CurYear.numorders
, PrevYear.numorders AS prev
FROM
```

```
        YearlyOrders AS CurYear
LEFT OUTER JOIN YearlyOrders AS PrevYear
        ON CurYear.orderyear = PrevYear.orderyear + 1
ORDER BY orderyear ;
```

***[참고]***
*CTE가 한 일괄 처리(Batch)의 일부분으로 실행되는 경우, 이전 구문들은 ; (세미콜론)으로 구별되어야 합니다.*

다음으로 재귀적 CTE에 대한 구문 형식과 예제입니다.

■ 아래의 재귀적 CTE 구문 형식에서
- 본문에는 UNION ALL 연산자로 구분된 두 개의 (멤버)쿼리를 포함한다.
- 앵커 멤버는, SQL Server가 단 한번 호출한다.
- 재귀 멤버는, 앵커 멤버를 시작으로 이전 단계에서 반환된 결과 집합을 표현하는 CTE 이름을 참조한다.
- SQL Server는 쿼리가 빈 결과 집합을 반환할 때까지 재귀 멤버를 반복 호출한다.

### [따라하기] 재귀적 CTE

1. 구문 형식

```
WITH RecursiveCTE(〈column_aliases〉)
AS
(
  -- 앵커 멤버(Anchor Member)
  SELECT ...
  FROM 〈some_table(s)〉
  ...
```

```
    UNION ALL

    -- 재귀 멤버(Recursive Member)

    -- RecursiveCTE 를 참조
    SELECT ...
    FROM <some_table(s)>
    JOIN RecursiveCTE
    ...
)
SELECT ...
FROM RecursiveCTE ;
```

2. 예제

```
USE AdventureWorks ;

-- 회사의 조직 구성도를 보여주기 위한 CTE 예제
WITH EmpsCTE(EmployeeID, ManagerID, FirstName, LastName, Title, EmpLevel,
SortKey)
AS
(
-- 앵커 멤버
SELECT
   EmployeeID, ManagerID, FirstName, LastName, em.Title
, 0, CAST(EmployeeID AS VARBINARY(900)) AS SortKey
FROM HumanResources.Employee em INNER JOIN Person.Contact ct
        ON em.ContactID = ct.ContactID
```

```
WHERE ManagerID IS NULL

UNION ALL

-- 재귀 멤버
SELECT
   E.EmployeeID, E.ManagerID, ct.FirstName, ct.LastName, E.Title
, ct.EmpLevel + 1, CAST(SortKey + CAST(E.EmployeeID AS BINARY(4)) AS
VARBINARY(900))
FROM HumanResources.Employee AS E
  INNER JOIN EmpsCTE AS ct
    ON E.ManagerID = ct.EmployeeID
)
SELECT EmpLevel
   , REPLICATE( ' | ', EmpLevel) + FirstName + ' ' + LastName
   , Title
FROM EmpsCTE
```

## 큰 값 데이터 형식

SQL Server 2000까지 한 칼럼의 대용량 데이터 처리를 위해서 text, ntext 그리고 image 데이터 형식을 사용했었습니다만, 이러한 데이터 형식 사용을 사용한 개발 작업이 결코 쉽지가 않았습니다.

■ 기존 대용량 데이터 형식의 한계점
- 변수나 출력 파라미터로 선언할 수 없다.
- SUBSTRING, LEFT 와 같은 일반적인 문자열 함수를 사용할 수 없는 경우가 대부분이다.

- 특별한 형식의 작업을 위해서는 TEXTPTR, WRITETEXT 그리고 UPDATETEXT 와 같은 별도의 명령을 사용해서 약간의 절차적 프로그램을 작성해야 한다.

SQL Server 2005의 도입된 큰 값 데이터 형식(MAX 지시어)은 이러한 기존 한계점들을 해결함으로써 대용량 데이터 형식을 일반 데이터 형식과 동일하게 프로그램하고 손 쉽게 다룰 수 있도록 지원합니다.

varchar, nvarchar 그리고 varbinary 형식의 선언 시의 MAX 지정자를 사용함으로써 최대 2GB 까지 저장할 수 있습니다.

### [따라하기] 큰 값 데이터 형식

컬럼의 자료형이 큰 값 데이터 형식이어도 데이터 입력시 데이터를 큰 값 데이터 형식으로 바꾸어 주어야 합니다

```
use tempdb
go
create table a (id int identity,a varchar(max))
insert a values(replicate( 'A' ,9000))
select * from a
select len(a) from a

insert a values(replicate(cast( 'A'  as varchar(max)),9000))
select len(a) from a
```

큰 값 데이터 형식을 사용한 경우, 그 값의 일부를 변경하거나 새로운 값을 입력하는 등의 부분 데이터 처리가 필요할 수 있습니다. SQL Server 2005에서 MAX 지정자와 함께 도입된 .WRITE(expression, @offset, @length) 절은 이전보다 훨씬 편리한 방법으로 그와 같이 데이터 처리에 사용될 수 있습니다. 간단한 따라 하기를 통해 그 편리함을 경험해 보시죠.

[따라하기] .WRITE( ) 절 사용

```
-- big 문자열 부분을 funny 라는 문자열을 대체
UPDATE dbo.EmployeeNote
        SET Notes.Write( 'funny' , CHARINDEX( 'big' , Notes)-1, 3)
WHERE EmployeeID = 2

SELECT * FROM dbo.EmployeeNote
```

***[참고]***
*SQL Server 2005에서 새롭게 도입한 또 다른 데이터 형식이 바로 xml 입니다. 변수나 파라미터 뿐만 아니라 테이블 칼럼의 데이터 형식으로 선언하고 xml 데이터를 원시 수준에서 저장할 수 있습니다. 또한 XQuery 언어와 관련 메서드들을 사용해서 xml 데이터에 대한 검색 및 수정 작업이 가능합니다. 이에 대한 xml 관련 장에서 살펴보겠습니다.*

## T-SQL 오류 처리

T-SQL 개발자들의 또 다른 고충 중의 하나가 바로 사용자 정의 트랜잭션 내에서의 오류 처리 루틴의 작성이었습니다. SQL Server 2000까지는 일종의 인-라인 오류 처리를 해야만 했습니다. 각각의 INSERT, UPDATE, 그리고 DELETE 구문마다 매번 @@ERROR 전역 변수를 사용해서 오류 검사를 하고 오류 발생 시 ROLLBACK 과 RETURN을 수행하도록 처리하거나 혹은 GOTO 문을 사용해서 오류 처리 루틴으로 분기하는 등의 코드 형식을 사용했습니다. 그 결과 코드의 길이가 방대해지고, 트랜잭션 코드 개발과 관리가 어려워졌으면 무엇보다 잘못된 오류 처리로 인해서 데이터 무결성 문제가 빌생할 수도 있었습니다.

현재 대부분의 고급 언어들(예를 들어, C#, C++나 자바 등)은 예외 처리(Exception Handling)라는 개념으로 보다 향상된 구조적 오류 처리 루틴을 제공합니다. SQL Server 2005에서도 이와 유사한 구조적 오류 처리 방식을 도입했습니다. 바로 C#, VB.NET 등의 언어에서 지원하는 것과 유사한 TRY...CATCH 구문입니다.

**[구문] TRY…CATCH**

```
BEGIN TRY
  { sql_statement | statement_block }
END TRY
BEGIN CATCH
  { sql_statement | statement_block }
END CATCH
[;]
```

TRY 블록 내에서 오류가 발생하는 경우, 이를 CATCH 블록 내에서 처리할 수가 있습니다. 또한 CATCH 블록 내에서, 오류 처리에 필요한 (이전에는 사용할 수 없었던) 다양한 정보 들을 관련된 함수를 통해서 얻을 수가 있습니다.

■ 오류 처리 관련 함수들

- ERROR_NUMBER( ) 는 오류 번호를 반환합니다.
- ERROR_SEVERITY( ) 는 오류 심각도 수준을 반환합니다.
- ERROR_STATE( ) 는 오류 상태 번호를 반환합니다.
- ERROR_PROCEDURE( ) 는 오류가 발생한 저장 프로시저 또는 트리거의 이름을 반환합니다.
- ERROR_LINE( ) 은 오류가 발생한 루틴 내의 줄 번호를 반환합니다.
- ERROR_MESSAGE( ) 는 전체 오류 메시지를 반환합니다. 이 메시지에는 매개 변수에 제공된 값을 포함합니다.

[따라하기] TRY…CATCH

```
BEGIN TRY
  -- ErrorNumber = 547번, FK 관련 참조 무결성 오류 발생
  DELETE Production.Product
  WHERE ProductID = 1
END TRY
BEGIN CATCH
  SELECT
    ERROR_NUMBER( ) AS ErrorNumber,
    ERROR_SEVERITY( ) AS ErrorSeverity,
    ERROR_STATE( ) AS ErrorState,
    ERROR_PROCEDURE() AS ErrorProcedure,
    ERROR_LINE( ) AS ErrorLine,
    ERROR_MESSAGE( ) AS ErrorMessage
END CATCH
```

위 구문은, 사용자 정의 트랜잭션을 포함하고 있지 않으므로, 간단한 형식입니다. 트랜잭션을 포함한다면, COMMIT/ROLLBACK 등에 대한 보다 안전하고 세밀한 처리가 요구될 것입니다.

## DDL 트리거

SQL Server 2000에서는 INSTEAD-OF 트리거 방식이 추가되기는 했지만, 여전히 DML 구문에 한정된 작업이었습니다. 많은 DBA 들이 오랜 전부터 DDL 명령에 대한 트리거 기능을 요구해 왔습니다. 주로 감사(Auditing) 기능이나 사용자 실수(혹은 의도적인)에 의한 스키마 변경 등의 이벤트를 감시하고 추적하기 위한 용도였습니다.

SQL Server 2005에서 이러한 요구 사항을 만족할 수 있도록 DDL 트리거를 지원합니다. 두 가지 수준에서 이벤트에 대한 트리거를 생성할 수 있습니다. 하나의 데이터베이스 수준에서 발생하는 이벤트에 대한 트리거(예를 들어, CREATE/ALTER/DROP TABLE, VIEW, USER 등), 나머지 하나는 서버 수준에서 발생하는 이벤트(예를 들어, CREATE/ALTER/DROP LOGIN, CERTIFICATE 등에 대한 트리거입니다.

구문 또한 기존의 DML 구문과 유사합니다. 다만 데이터베이스 수준일 경우에 "ON DATABASE" 절을 지정하거나 서버 수준일 경우 "ON ALL SERVER" 절을 지정하는 것이 중요한 차이점입니다.

또한, 트리거 내에서 참조하는 inserted, deleted 테이블은 생성되지 않습니다. 대신에, EVENTDATA() 라는 함수 호출을 통해서 이벤트에 관련된 정보 들(예를 들어, 이벤트를 발생한 SPID, 발생 시간, 관련 구문 등)을 액세스할 수 있습니다. 특히 EVENTDATA()가 반환하는 데이터 형식이 XML 데이터이므로 전체 결과를 XML 변수 등에 저장하고 처리할 수도 있지만, SQL Server 2005에서부터 지원되는 XQuery 언어의 검색 기능을 활용해서 특정 값을 처리하도록 설계할 수도 있습니다.

### [따라하기] 데이터베이스 수준의 DDL 트리거

```
-- 트리거 관련 데이터를 저장할 테이블
CREATE TABLE DDL_Log (
        ID      int IDENTITY(1, 1)
                        PRIMARY KEY
,       TrData          xml
)
GO

-- 테스트용으로 DROP, ALTER를 수행 할 테이블
CREATE TABLE trTestT1 ( a int )
GO
```

```
-- DDL 트리거
CREATE TRIGGER tr_Test1
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
  DECLARE @trData XML
  -- EVENTDATA( ) 함수 호출 결과를 XML로 저장하거나, XQuery로 검색
  SET @trData = EVENTDATA( )

  -- DDL 트리거는 ROLLBACK 처리가 가능하다
  ROLLBACK

  -- Log 기록
  SET NOCOUNT ON
  INSERT INTO DDL_Log (TrData) VALUES (@trData)
GO

/*
  DDL 트리거 호출 테스트
*/

-- 1) DROP TABLE 이벤트는 직접 수행
DROP TABLE trTestT1
SELECT * FROM DDL_Log
GO

-- 2) ALTER TABLE 이벤트는 TRY ... CATCH 내에서 수행해 봄
BEGIN TRY
```

```
        ALTER TABLE trTestT1 ADD b int
END TRY
BEGIN CATCH
        SELECT ERROR_NUMBER( ), ERROR_MESSAGE( )
END CATCH
GO
SELECT * FROM DDL_Log
```

예제에서 수행된 결과로 DDL_Log에 기록된 데이터를 보면, EVENTDATA( ) 함수의 출력 결과를 확인할 수 있습니다.

### [따라하기] 데이터베이스 수준의 DDL 트리거

```
<EVENT_INSTANCE>
  <EventType>DROP_TABLE</EventType>
  <PostTime>2005-11-03T20:19:01.590</PostTime>
  <SPID>52</SPID>
  <ServerName>YUKON</ServerName>
  <LoginName>YUKON\Administrator</LoginName>
  <UserName>dbo</UserName>
  <DatabaseName>tempdb</DatabaseName>
  <SchemaName>dbo</SchemaName>
  <ObjectName>trTestT1</ObjectName>
  <ObjectType>TABLE</ObjectType>
  <TSQLCommand>
    <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT="ON"
ANSI_PADDING="ON" QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
    <CommandText>DROP TABLE trTestT1
```

```
〈/CommandText〉
 〈/TSQLCommand〉
〈/EVENT_INSTANCE〉
```

다음은 서버 수준의 DDL 트리거 예제입니다.

### [따라하기] 서버 수준의 DDL 트리거

```
-- CREATE LOGIN 에 대한 DDL 트리거
CREATE TRIGGER tr_Test2
ON ALL SERVER
FOR CREATE_LOGIN
AS
  SET NOCOUNT ON

  -- XQuery 의 value( ) 메서드를 사용, 특정 요소 값을 검색하고 비교
  IF (EVENTDATA( ).value( '(/EVENT_INSTANCE/LoginType)[1]' ,  'varchar(100)' )
                                        =  'SQL Login' )
  BEGIN
       ROLLBACK
       PRINT  'SQL 표준 사용자는 생성할 수 없습니다!'
  END
  ELSE
  BEGIN
       PRINT  'Windows 사용자는 생성 가능합니다!'
  END
GO
```

```
-- SQL 표준 사용자 생성 테스트 => 실패
CREATE LOGIN SqlUser WITH PASSWORD = 'p@ssw0rd'
GO

-- Windows 표준 사용자 생성 테스트
CREATE LOGIN [yukonfan\Guest] FROM WINDOWS
GO
DROP LOGIN [yukonfan\Guest]
GO
```

SQL Server 2005에서는 이벤트 그룹이라는 개념으로, 관련된 이벤트에 대한 그룹 단위의 트리거 처리를 지원합니다. 예를 들어, CREATE/ALTER/DROP TABLE 개별 이벤트를 지정하는 대신, DDL_TABLE_EVENTS 라는 이벤트 그룹을 지정할 수도 있습니다.

다음은 트리거에 관련된 메타데이터 검색이 필요할 때 접근할 수 있는 카탈로그 뷰들입니다.

### [따라하기] 트리거 관련 카탈로그 뷰

```
SELECT * FROM sys.triggers
SELECT * FROM sys.server_triggers
SELECT * FROM sys.server_trigger_events
```

DDL과 DML 트리거는 동기적으로 작동합니다. 즉, 트리거의 코드가 완료될 때까지 트리거를 발생시킨 작업으로 제어권을 넘기지 않는다는 것입니다.

SQL Server 2005에서는 비 동기적인 이벤트 처리를 지원하기 위한 새로운 방법으로 이벤트 알림을 도입했습니다. 여러 개의 서로 다른 어플리케이션에서 이벤트가 발생할 때 알림을 주도록 가입을 하기 되면, 해당 이벤트가 발생했을 때, 가입된 모든 어플리케이션들이 자신들의 동작이 종료될 때까지 기다릴 필요 없이 바로 그 이벤트에 대한 코드를 실행할 수 있습니다.

## 이벤트 알림(Event Notifications)

이벤트 알림은 앞서 소개한 DDL 트리거의 한계점을 해결할 수 있는 또 다른 접근 방법을 제공합니다. 서버나 데이터베이스에서 DDL 이벤트가 발생 시 이를 비 동기적으로 처리할 수 있으며, 이벤트 알림을 원격 서버에서 처리할 수도 있으므로, 확장성과 유연성을 제공됩니다. 또한 이벤트 알림은 기존의 SQL 프로필러나 SQLTrace를 통해서나 추적할 수 있었던 SQL Trace 이벤트에 대해서도 알림을 발생하고 처리할 수 있습니다.

어떻게 이벤트에 대한 비동기적이 처리가 가능할까요? 그것은 역시 SQL Server 2005에 새로 도입된 서비스 브로커(Service Broker) 서비스의 기능입니다. DDL 혹은 SQL Trace 이벤트가 발생하면 연관된 서비스 브로커 큐에 저장되고 이를 비 동기적으로 처리하는 것입니다. 서비스 브로커 서비스는 데이터베이스 엔진에서 신뢰할 수 있는 비동기적 메시징 플랫폼을 제공하는 새로운 기술입니다.

■ DDL 트리거나 이벤트 알림의 차이점 요약

- 트리거는 그 발생 원인이 되는 트랜잭션의 범위 내에서 동기적으로 실행되므로 필요한 경우 ROLLBACK 이 가능합니다. 반면에 이벤트 알림은, 트랜잭션 범위 내에서 실행되지 않으므로 ROLLBACK 이 안되며, 서비스 브로커 서비스를 통해서 비 동기적으로 처리됩니다.
- 서버 나 데이터베이스의 DDL 이벤트뿐만 아니라 SQL trace 이벤트에도 응답할 수 있습니다.
- 이벤트 알림은 원격 서버에서도 처리될 수 있습니다.

따라서, 이벤트 알림에 대한 기술은 서비스 브로커 서비스에 대한 이해를 요구합니다. 자세한 내용은 서비스 브로커 서비스 소개를 통해서 살펴봅니다.

이벤트 알림은 CREATE EVENT NOTIFICATION 구문을 사용해서 정의합니다. 이 때, 관련된 큐(Queue)와 브로커 서비스에 대한 지정이 필요하므로 미리 정의되어 있어야 합니다.

### [따라하기] 이벤트 알림

```
USE AdventureWorks

-- 이벤트 정보를 기록할 테이블
CREATE TABLE dbo.evtAudit (
  evtData              xml              NULL
)

-- 이벤트 처리용 프로시저 생성
CREATE PROC dbo.up_Audit
AS
  DECLARE @evtData varbinary(max)

  -- 관련된 큐에서 첫 번째 메시지를 추출
  -- RECEIVE 구문의앞에; 이와야한다
  ; RECEIVE TOP (1) @evtData = message_body
  FROM AdventureWorks.dbo.AuditQueue

  IF CAST(@evtData AS xml) IS NOT NULL
  BEGIN
  INSERT INTO dbo.evtAudit (evtData)
       VALUES (CAST(@evtData AS xml))
  END
```

```
GO

-- 관련 큐 생성
CREATE QUEUE dbo.AuditQueue
WITH STATUS = ON
, ACTIVATION (
  -- 여기서 프로시저를 연계한다
  PROCEDURE_NAME = AdventureWorks.dbo.up_Audit
  , MAX_QUEUE_READERS = 1
  , EXECUTE AS SELF
)
ON [DEFAULT]
GO

-- 서비스 브로커 서비스 생성
CREATE SERVICE dbo.AuditService ON QUEUE dbo.AuditQueue
  ([http://schemas.microsoft.com/SQL/Notifications/PostEventNotification])
GO

-- 이벤트 알림 생성, 관련 서비스 브로커 연동
CREATE EVENT NOTIFICATION en_Audit
ON SERVER
FOR CREATE_DATABASE, DROP_DATABASE
TO SERVICE 'AuditService' , 'current database'
GO

-- 테스트
CREATE DATABASE TestDB1
```

```
DROP DATABASE TestDB1
GO

SELECT * FROM dbo.evtAudit
GO
```

위 스크립트를 수행하면, evtAudit 테이블에서 2건의 이벤트에 대해 저장된 XML 데이터를 확인할 수 있습니다. 이 XML 데이터는 앞서 DDL 이벤트에서 제공되는 것과 유사한 요소로 구성되어 있습니다.

다음은 이벤트 알림과 관련된 메타데이터 검색을 위한 카탈로그 뷰들입니다.

**[따라하기] 이벤트 알림 관련 카탈로그 뷰**

```
SELECT * FROM sys.event_notifications
SELECT * FROM sys.server_event_notifications

SELECT * FROM sys.events
SELECT * FROM sys.server_events
SELECT * FROM sys.trace_events
SELECT * FROM sys.event_notification_event_types
```

## 스냅숏 격리(Snapshot Isolation)

SQL Server 2000에서는 READ COMMITTED 격리 수준을 사용하는 경우 데이터 검색 작업과 변경 작업이 동시에 같은 리소스(기본적으로 행 단위 잠금)에 발생할 때, 그 두 작업 간의 차단(Blocking)이 발생했습니다. 이는 데이터의 일관성을 보장하기 위해서이지만 과도한 차단은 서버의 안정적인 서비스의 장애 요소입니다. 많은 사용자들이 이 문제를 해결하기 위해서 SELECT 작업 시의 READ UNCOMMITTED(혹은 NOLOCK) 격리 수준이나 참고(Hint)를 사용해서 처리했습니다. 그러나 이 접근 방법은 Dirty Read(현재 변경 중인 데이터 읽기)를 발생시킴으로 인해서 중요한 업무에 있어서 데이터 일관성에 문제를 일으킬 수 있다는 것입니다.

SQL Server 2005에서는 기존의 차단 문제를 피하면서 Dirty Read와 같은 데이터 일관성 문제도 지킬 수 있는 또 다른 옵션으로 스냅숏 격리 수준(Snapshot Isolation Level)이라는 것을 지원합니다. (여기서 데이터 일관성 문제를 지킨다는 것에는 약간의 견해 차이가 있을 수 있습니다.)

스냅숏 격리 수준은, 데이터 검색 시의 변경 중인 데이터(Dirty)를 읽도록 허용하는 것이 아니라 변경되기 이전의 정보를 읽을 수 있는 기반 구조를 제공하는 것입니다. 즉 행이 변경될 때 그 행의 이전 버전을 별도로 저장해 둔 뒤에 해당 행의 검색 시의 제공을 하는 것입니다. 바로 이러한 기반 구조를 SQL Server 2005에서 행 버전 관리(Row Versioning)라고 합니다.
행 버전 관리를 위해 사용되는 데이터는 바로 tempdb 입니다. 행 버전 관리는 SQL Server 2005에서 아주 중요한 기반 구조로 제공되며 따라서 tempdb는 이전과 비교할 수 없을 정도로 중요한 시스템 데이터베이스로 자리잡게 될 것으로 예상됩니다. 행 버전 관리에 대해서도 별도로 살펴봅니다.

■ 스냅숏 격리는 두 가지 방법으로 보다 상세한 제어가 가능합니다
- READ_COMMITTED_SNAPSHOT 데이터베이스 옵션
  READ COMMITTED 격리 수준에서 Dirty Read와 Non-Repeatable Read를 피하기 위해서 잠 금을 사용하는 대신, 행 버전 관리를 사용하도록 합니다.

- ALLOW_SNAPSHOT_ISOLATION 데이터베이스 옵션
  이 옵션을 설정한 뒤, SET TRANSACTION ISOLATION LEVEL SNAPSHOT 세션 옵션을 설정하면 해당 세션에서 스냅숏 격리 수준이 적용됩니다.

## 순위 함수들

SQL Server 2005에 새로 도입된 아주 반가운 기능 중의 하나가 바로 순위 함수입니다. 관계형 데이터베이스에서 복잡한 형식의 결과 집합을 처리하는 경우 해당 결과를 위한 쿼리의 형식과 원본 데이터 집합에서 순번(Sequence Number) 혹은 행 번호(Row Number)라고 불리는 열이 필요한 경우가 많이 있습니다. 또는 순위에 해당하는 열이 반드시 있어야만 원하는 결과 집합을 산출하는 쿼리를 작성할 수 있는 경우도 있습니다.

그러나 SQL Server 2000에서는 순번 혹은 순위 값을 처리하기 위한 기능으로 IDENTITY 속성을 지정한 임시 테이블 혹은 테이블 변수를 사용하거나 IDENTITY( ) 함수를 SELECT ... INTO 문과 함께 적용하는 방법, 심지어는 상관 하위 쿼리 등을 사용해서 순번 혹은 순위 값을 가진 중간 결과 집합을 만들고 이를 이용해서 원하는 쿼리 형식을 만들어야만 했습니다. 이러한 방법은 쿼리 작성을 힘들게 할 뿐만 아니라 그 성능 또한 문제를 일으키는 것이 일반적이었습니다.

SQL Server 2005에서는 ROW_NUMBER( ), RANK( ), DENSE_RANK( ), 그리고 NTILE( )라는 4가지 함수를 통해서 아주 쉽게 결과 집합을 구성할 수 있도록 지원합니다. (쿼리 개발자들에게는 정말 반가운 소식입니다!)

순번이나 순위를 결정하기 위해서는, 행 집합이 특정 순서로 정렬이 되거나 혹은 특정 파티션 단위로 정렬되어 있어야 합니다. 이를 위해서 각 함수에는
OVER ([〈partition_by_clause〉] 〈order_by_clause〉) 구문을 제공합니다.

아마도 자세한 설명이 필요 없을 것이라 판단 됩니다.
아래 몇 가지 예문 들을 통해서, SQL Server 2000에서 기본적으로 사용되던 방식과 SQL Server 2005에 도입된 기능을 비교 분석해 보시면 바로 이해가 되실 겁니다.

### [따라하기] 순위 함수

1. ROW_NUMBER 예제

```
USE AdventureWorks

-- 1) SQL Server 2000에서 상관 하위 쿼리를 사용한 방법
SELECT
  -- ProductID/LocationID 별 순번 생성을 위한 하위쿼리 내의 COUNT 작업
  -- 아래에서, COUNT(*)를 사용하는 경우, 조건에서 〈, 〈= 를 사용하는 경우 등에
  -- 따라서 결과에는 차이가 발생할 수 있습니다.
  (SELECT COUNT(ProductID) FROM Production.ProductInventory p2
        WHERE p2.ProductID 〈 p1.ProductID
                    OR (p2.ProductID = p1.ProductID
                                AND p2.LocationID 〈= p1.LocationID)
  ) AS SeqNo
, ProductID
, LocationID
, Quantity
FROM Production.ProductInventory p1
ORDER BY ProductID, LocationID

-- 2) SQL Server 2005에서 ROW_NUMBER( )를 사용한 방법
SELECT
        ROW_NUMBER() OVER (ORDER BY ProductID, LocationID) AS SeqNo
,       ProductID
, LocationID
```

```
, Quantity
FROM Production.ProductInventory
GO

-- 3) ROW_NUMBER()의 결과를 이용한, 일종의 페이징 처리
SELECT *
FROM (SELECT
        ROW_NUMBER( ) OVER (ORDER BY ProductID, LocationID) AS SeqNo
          , ProductID

        , LocationID
        , Quantity
        FROM Production.ProductInventory) p1
WHERE p1.SeqNo BETWEEN 90 AND 100
```

마지막 3번의 경우, 웹 어플리케이션 개발자 분들이 아주 절실히 원하던 방법이죠. 여기서 앞서 소개한 TOP 구문의 향상 기능을 병행한다면 이전 보다 훨씬 좋은 쿼리 형식을 만들 수 있습니다. 물론 대용량 페이징 처리에서는 성능 문제를 추가로 고려해야 합니다.

RANK( ) 와 DENSE_RANK( ) 함수는 말 그 대로 결과 집합에 순위 형식의 열을 사용할 수 있도록 제공됩니다. DENSE_RANK( ) 함수는 복권 추천 결과와 같습니다. 1등이 여러 명 있다고 2, 3등이 없는 건 아니죠?

2. RANK( ), DENSE_RANK( ) 예제

```
SELECT
   RANK() OVER (ORDER BY ProductID) AS Rank
, DENSE_RANK() OVER (ORDER BY ProductID) AS DenseRank
```

```
,*
FROM Production.ProductInventory
GO
```

마지막 4번째 NTILE( ) 함수는 지정된 정렬(혹은 파티션) 조건에 따라서 결과 집합을 지정된 N개의 버킷(Bucket)으로 나눈 결과를 제공합니다. 예를 들어, 한 클래스의 학생 수가 35명일 때 이를 4개의 그룹으로 나누고자 한다면, 한 그룹에 몇 몇씩 포함되면 될까요?

3. NTILE( ) 예제

```
SELECT
   NTILE(3) OVER (ORDER BY Name) AS NBucket
,*
FROM Production.Product
WHERE ProductID <= 316
GO
```

위 결과 집합은 5개의 행으로 구성됩니다. 이를 3개의 버킷으로 나누고자 합니다. 5를 3으로 나누어보시죠? 몫은 1, 나머지는 2가 됩니다. 즉, 전체 3개의 버킷에 1 행씩 포함되고 처음 2 버킷에 추가로 1행씩 포합됩니다.

## 새로운 관계 연산자들: PIVOT, UNPIVOT, 그리고 APPLY

혹시 Microsoft Access를 사용 해 본 적이 있습니까? 그렇다면 Jet Database 엔진에서 제공되는 TRANSFORM 구문에 대해서도 알고 계실 겁니다. 흔히 교차 집계 보고서(CROSSTAB) 또는 PIVOT 테이블이라고 부르는 결과 집합을 만들 수 있는 명령이죠. 한국형 분석 보고서의 대표적인 사례이지만 안타깝게도 SQL Server 2000 버전까지는 같은 기능의 명령이 지원되지 않았던 관계로 데이터베이스 개발자를 힘들게 만드는 또 다른 어려움 중의 하나였습니다.

SQL Server 2000에서는 이를 해결하기 위한 전통적인 방법으로, 주로 CASE, GROUP, 그리고 집계 함수 등을 활용했습니다. 이제 SQL Server 2005에서는 새로 도입된 PIVOT, UNPIVOT 연산자의 도움으로 이러한 결과 집합의 구성을 보다 쉽고 편리하고 만들 수 있습니다.

PIVOT 연산자는 행 집합을 칼럼으로 변환하는데(행->열) 사용될 수 있으며, 반대로 UNPIVOT 을 열을 행으로 변환하는데 사용될 수 있습니다.

다음 따라 하기 예제를 통해서 그 사용 법을 살펴보시죠.

### [따라하기] PIVOT 과 UNPIVOT 연산자 사용 예

```
-- 고객별 / 년도별, SubTotal의 합을 보여주는 교차 집계 보고서 생성
SELECT CustomerID, [2001], [2002], [2003]
FROM
  (SELECT YEAR(OrderDate) SalesYear, CustomerID, SubTotal
    FROM Sales.SalesOrderHeader
    WHERE OrderDate >= '20010101' AND OrderDate < '20040101'
      AND CustomerID <= 10
  ) Sales
PIVOT (SUM(SubTotal) FOR SalesYear
```

```
IN ([2001], [2002], [2003])) SalesPivot
ORDER BY CustomerID
GO

-- 다음은 UNPIVOT 예제를 보여주기 위한 샘플 테이블을 생성
SELECT CustomerID, [2001], [2002], [2003] INTO dbo,SalesPivot
FROM
  (SELECT YEAR(OrderDate) SalesYear, CustomerID, SubTotal
     FROM Sales,SalesOrderHeader
     WHERE OrderDate >= '20010101' AND OrderDate < '20040101'
       AND CustomerID <= 10
  ) Sales
PIVOT (SUM(SubTotal) FOR SalesYear
IN ([2001], [2002], [2003])) PSales
GO

-- SalesPivot 을 사용, 년도별 집계 정보를 행으로 변환
SELECT CustomerID, SalesUnPivot,SalesYear, SalesUnPivot,SubTotal
FROM dbo,SalesPivot
UNPIVOT (SubTotal FOR SalesYear
IN ([2001], [2002], [2003])) SalesUnPivot
ORDER BY CustomerID
GO
```

이번엔 APPLY이 연산자를 살펴보시죠.

SQL Server 2000에서는 테이블 값 함수 형식을 제공했습니다. 함수가 결과 집합을 반환하도록 구성하고 이를 FROM 절 지정해서 함수를 마치 뷰처럼 사용할 있는 인터페이스를 제공했습니다. 즉, T-SQL FROM 절에 지정할 수 있는 대략 5가지 종류의 테이블 원본(Table Source) 중의 하나였습니다. SQL Server 2005에서는 FROM절에 지정된 테이블 원본과

APPLY 연산자를 활용 그 테이블 원본에서 반환되는 각 행에 대해서 테이블 값 함수를 반복 호출하고 다시 함수가 반환하는 결과 집합을 외부 테이블 원본과 결합해서 열 목록에서 사용할 수 있도록 지원합니다. 간단히 정리하면, 특정 테이블 원본의 행 집합에 대해서 테이블 값 함수를 반복 호출하고 그 결과를 조인할 수 있는 쿼리 형식을 제공하는 것입니다.

APPLY는 조인 방식의 따라 다시 두 가지로 나누어집니다. CROSS APPLY 와 OUTER APPLY 입니다. CROSS APPLY는 일종의 내부 조인(INNER JOIN)을 결과를 가집니다. 즉 함수가 결과 집합을 반환하지 않는 경우, 조인 결과에 포함되지 않습니다. 그렇다면 OUTER APPLY는 짐작이 되시겠죠? 네, 외부 조인(OUTER JOIN)의 결과 형식입니다. 함수가 결과 집합을 반환하지 않더라도 외부 테이블의 해당 행은 반환이 되며, 결과가 없는 열 값은 NULL로 표현됩니다.

### [따라하기] PIVOT 과 UNPIVOT 연산자 사용 예

```
-- 예제용 테이블 값 함수
-- 고객ID를 파라미터 값으로, 해당 고객의 주문 정보 n 건을 반환
CREATE FUNCTION uf_CustOrders(
   @custid int
, @n AS int
) RETURNS TABLE
AS
RETURN
  SELECT TOP (@n) * FROM Sales.SalesOrderHeader
  WHERE CustomerID = @custid
  ORDER BY OrderDate DESC, SalesOrderID DESC
GO

-- CROSS APPLY, Customer테이블의 각 고객 별로 함수 호출, 조인된 결과 집합을
반환
-- 참고, ID=13 고객의 결과는 반환되지 않는다.
```

```
SELECT c.CustomerID, c.AccountNumber, o.*
FROM Sales.Customer AS c

  CROSS APPLY uf_custorders (c.CustomerID, 3) As o
WHERE c.CustomerID BETWEEN 13 AND 15
ORDER BY c.CustomerID, o.OrderDate DESC, o.SalesOrderID DESC
GO

-- OUTER APPLY, ID=13 고객도 반환된다.
SELECT c.CustomerID, c.AccountNumber, o.*
FROM Sales.Customer AS c
  OUTER APPLY uf_custorders (c.CustomerID, 3) As o
WHERE c.CustomerID BETWEEN 13 AND 15
ORDER BY c.CustomerID, o.OrderDate DESC, o.SalesOrderID DESC
GO
```

## OUTPUT 절

DELETE, INSERT, 혹은 UPDATE와 같은 DML 구문을 실행한 후에 그 반영된 결과 집합을 다시 재 사용해야 할 상황이 자주 발생합니다. 기존에 다시 그 결과 집합을 SELECT 하거나 혹은 UPDATE 구문의 경우 SET @변수 = 열 = 값 이라는 특수한 구문 형식을 빌어서 그 결과를 처리하기도 했습니다.

SQL Server 2005의 OUTPUT 절은 DELETE, INSERT, UPDATE 구문에서 변경된 행 집합을 저장해 두고 이를 inserted 와 deleted 라는 일종의 가상 테이블(그 내용은 이전 트리거와 동일)을 통해서 재 사용할 수 있도록 구문으로 제공합니다.

### [따라하기] OUTPUT 절

```
-- SELECT ... INTO 구문으로 예제용 테이블 생성
SELECT TOP 50
        SalesOrderID, SalesOrderDetailID, OrderQty
INTO dbo,SalesEx
FROM Sales,SalesOrderDetail
GO

-- DELETE 구문과 OUTPUT 절 예제
DELETE dbo,SalesEx
        OUTPUT deleted.*
WHERE SalesOrderID = 43659
GO

-- UPDATE 구문과 OUTPUT 절 예제
UPDATE dbo,SalesEx

        SET OrderQty = OrderQty * 100
        OUTPUT deleted.*, inserted.*
WHERE SalesOrderID = 43660
GO

/*
  OUTPUT 절의 결과 집합을 테이블 변수에 저장하는 경우
  주의. 테이블 변수를 사용하므로 아래 구문은 모두 동시에(배치 단위)에 실행함.
*/
-- 테이블 변수 선언
DECLARE @SalesExR TABLE (
        SalesOrderID  int
```

```
,       SalesOrderDetailID          int
,       OrderQty        smallint
)

-- UPDATE 구문을 실행한 후, 변경 이전의 이미지를 반환하고 저장
UPDATE dbo.SalesEx
        SET OrderQty = OrderQty + 100
        OUTPUT deleted.*            -- 열의 정의는 일치해야 한다
        INTO @SalesExR
WHERE SalesOrderID = 43660

SELECT * FROM @SalesExR
GO
```

## BULK 행 집합 공급자(BULK Rowset Provider)

대용량 데이터의 대량 입력 시의 BCP.EXE 유틸리티나 BULK INSERT 명령을 사용합니다. 이 경우, 해당 결과 집합을 SELECT 문으로 재 가공하는 작업이 힘들었습니다. SQL Server 2005에서는 OPENROWSET 함수가 BULK 행 집합 공급자를 지원하도록 확장되었습니다.

BULK 행 집합 공급자와 OPENROWSET 함수를 활용, 파일을 읽어서 파일의 내용을 행 집합으로 반환할 수가 있습니다. BCP.EXE 유틸리티와 거의 동일한 옵션들을 지원하며 또한 이전과 동일한 서식 파일이나 혹은 XML 서식 파일을 사용할 수도 있습니다.
(이전의 서식 파일을 써 보신 분들이라면 이것이 얼마나 힘든 작업인지 아실 겁니다! XML 서식 파일의 지원은 정말 반가운 소식입니다!)

뿐만 아니라 SQL Server 2005에서 새로 도입된 큰 값 데이터 형식, 즉 MAX 지시어로 정의된 열에 대해서 파일 내의 저장된 대용량의 문자 혹은 바이너리 형식의 데이터를 하나의 행과 하나의 열로 입력할 수가 있습니다. 정말 편리해졌죠!
이를 위해 세 가지 옵션이 제공됩니다. SINGLE_BLOB, SINGLE_CLOB, SINGLE_NCLOB 입니다.

SINGLE_CLOB와 SINGLE_NCLOB는 각각 varchar(max)와 nvarchar(max) 열에 입력을 할 경우 사용될 수 있습니다. 물론 xml 데이터 형식의 칼럼에 대해서 xml 파일의 내용을 저장하는데 사용될 수 있습니다. 특히, SINGLE_NCLOB 옵션의 경우 해당 파일이 유니코드 형식이어야 합니다. SINGLE_BLOB은 varbinary(max) 열에 파일의 내용을 입력하고자 할 경우입니다.

### [따라하기] OPNEROWSET 함수와 BULK 행 집합 제공자

```
-- 예제용 테이블
CREATE TABLE dbo.BULK_Ex (
  id     int IDENTITY(1, 1) NOT NULL
         PRIMARY KEY
, SetupLog  varchar(max)
)
GO

-- 아래에서는, 임의의 text 형식 파일을 예제로 사용하면 됩니다.
INSERT INTO BULK_Ex (SetupLog)
  SELECT b.SetupLog
   FROM OPENROWSET(BULK N' C:\Windows\setup.log' , SINGLE_CLOB) AS
b(SetupLog)

SELECT * FROM dbo.BULK_Ex
GO
```

## 새로운 선언적 참조 무결성 동작

선언전 참조 무결성(DRI, Declarative Referential Integrity) 동작은 부모 값이 수정 및 삭제되는 경우의 외래 키(참조 키, FK)로 선언 된 자식 값의 동작을 정의 합니다. 이는 외래 키의 선언 시의 ON DELETE/ON UPDATE 절을 통해서 지정하게 됩니다. SQL Server 2000 버전까지는 두 가지 동작을 지원했습니다. 첫 번째는 "NO ACTION" 이라는 동작으로, 바로 부모 값의 수정 및 삭제를 허용하지 않는 것입니다. 두 번째가 "CASCADE" 동작(SQL Server에서는 "연계 참조 무결성" 이라고 소개합니다)으로 부모 값이 수정되는 경우 자식 값도 동일한 값으로 수정됨을 정의합니다. 문제는 바로 DELETE 작업에 대한 "CASCADE" 동작입니다. 부모 값이 삭제되는 경우 자식 값만 삭제되는 것이 아니라, 자식 값을 가진 행 전체가 삭제된다는 것이었습니다.

SQL Server 2005에서는 이제 두 가지 새로운 DRI 동작을 지원합니다. 바로 SET NULL 과 SET DEFAULT 입니다. 즉, 자식 값을 NUL 값이나 DEFAULT 제약 조건의 값으로 적용함을 의미합니다. 따라서 SET NULL의 경우의 FK로 선언된 열에 NULL 허용이 되어야 하며, SET DEFALUT 의 경우 해당 열의 DEFAULT 제약 조건이 선언되어 있거나 NULL 허용이어야 합니다. 이 두 가지 동작의 추가로 인해서 참조 무결성 자동화의 어려운 부분을 해소하는데 큰 도움이 되리라 생각됩니다.

### [따라하기] SET NULL / SET DEFAULT 연계 참조 무결성 동작

```
USE tempdb

-- 부모 테이블의 정의
CREATE TABLE dbo,Products (
  ProductID int NOT NULL
              PRIMARY KEY
, ProductName varchar(20)
)
```

```
-- 자식 테이블-1, SET NULL을 적용한 경우
CREATE TABLE dbo.Orders1 (
  OrderID int           NOT NULL
                        PRIMARY KEY
, ProductID int NULL
                        FOREIGN KEY
                        REFERENCES dbo.Products(ProductID)
                        ON DELETE SET NULL
)

-- 자식 테이블-2, SET DEFAULT를 적용한 경우
CREATE TABLE dbo.Orders2 (
  OrderID int           NOT NULL
                        PRIMARY KEY
, ProductID int NULL
                        DEFAULT (0)  -- DEFAULT가 없으면 NULL로 적용

                        FOREIGN KEY
                        REFERENCES dbo.Products(ProductID)
                        ON DELETE SET DEFAULT
)
GO
-- 임의 데이터 입력
INSERT dbo.Products SELECT 0,  '보류'
INSERT dbo.Products SELECT 1,  'XBOX'
INSERT dbo.Products SELECT 2,  'NOTEBOOK'

INSERT dbo.Orders1 SELECT 100, 1
INSERT dbo.Orders2 SELECT 100, 1
GO
```

```
-- 부모 값을 삭제한 후의 자식 테이블의 결과를 확인
DELETE dbo.Products WHERE ProductID = 1

SELECT * FROM dbo.Orders1
SELECT * FROM dbo.Orders2
```

## 메타데이터 뷰와 동적 관리 뷰(Dynamic Management Views)

SQL Server 2000에서 메타데이터 검색을 위해서 주로 시스템 저장 프로시저, 시스템 함수, 혹은 INFORMATION_SCHEMA 뷰를 사용했습니다. 이러한 방법으로도 원하는 메타데이터 접근이 어려운 경우엔 직접 관련된 시스템 테이블을 검색했습니다.

INFORMATION_SCHEMA 뷰의 경우엔 표준을 만족하기 위한 인터페이스므로 여전히 지원됩니다만, SQL Server 2005에서는 메타데이터 검색을 위한 새로운 인터페이스를 제공합니다. 바로 카탈로그 뷰(Catalog View)입니다. 카탈로그 뷰는 시스템 메타데이터 검색을 위한 일관되고 쉬운 인터페이스와 더불어 기존의 문제가 되었던 메타데이터의 대한 보안 기능을 위한 새로운 인터페이스입니다. 무엇보다 SQL Server 2005의 모든 기능에 대한 메타데이터 접근 방법입니다. 다양한 범주에 따라서 많은 카탈로그 뷰들이 제공되며 이들은 모두 sys 스키마 내에 정의됩니다.

하위 버전 호환성을 위해서 기존의 시스템 테이블들은 같은 이름과 같은 데이터를 반환하도록 지원됩니다만, 호환성 뷰(Compatibility View)라는 이름으로 뷰 집합으로서 제공됩니다.
이 책에서 각각의 주제 영역별로 관련된 카탈로그 뷰들이 소개될 것입니다.

동적 관리 뷰(DMV, Dynamic Management View)는 SQL Server 2005에서 새롭게 제공되는 또 다른 형식의 뷰입니다. 동적 관리 뷰는 SQL Server 2005 시스템의 현재 상태를 나타냅니다. SQL Server의 내부 메모리 구조나 상태에 대한 실시간 이미지 정보를 포함해서, I/O, 인덱스, 쿼리 플랜 캐시, 전체 텍스트 인덱스, 서비스 브로커, 트랜잭션 등 다양한 정보를 제공

합니다 이를 이용해서 서비스 상태를 위한 실시간 모니터링이나 튜닝 및 최적화에 사용될 수 있습니다. 특히 SQL Server 2005에서는 이전 버전과 비교할 수 없을 정도로 많은 분량의 정보를 제공하고 있으며, SQL Server Management Studio(SSMS)에서는 사용자가 중요한 상태 정보들을 보다 쉽게 확인할 수 있도록 관련된 보고서 메뉴를 제공합니다. 동적 관리 뷰 또한 sys 스키마 내에서 정의되며 dm_ 로 시작하는 이름 규칙과 내부 데이터의 범주를 나타내는 이름으로 연결됩니다. 예를 들어, 이전의 sp_who 과 같은 결과를 보고자 할 경우 sys.dm_ exec_sessions를 검색합니다. SQL Server 2005에서는 특히 시스템의 유지 관리를 책임지고 있는 사용자에 있어서 이 동적 관리 뷰의 기능이 중요하게 다루어 질 것입니다.

## 기타

지금까지 소개한 기능들 이외에도 SQL Server 2005에서 새로 도입되거나 향상된 기능들은 많이 있습니다. 지면 상 모두 다루지는 못하지만 독자 여러분 스스로 반드시 살펴보시기를 권장합니다.

■ T-SQL 추가 향상 기능들

- ALTER INDEX 구문 추가
- 문-단위 다시 컴파일(재컴파일, Recompile) 기능 향상
- 새로운 서버 구성 옵션 제공
- EXCEPT 와 INTERSECT 연산자 추가
- SET SHOWPLAN_XML 과 SET STATISTICS XML 세션 옵션 추가

등 입니다.

# ADO.NET 2.0

.NET Framework 2.0 과 함께 소개될 ADO.NET 2.0은 이번 버전과 비교하기 힘들 정도로 많은 추가 기능과 향상을 제공합니다. 그것이 SQL Server 사용자들을 대상으로 한 이 책에서 ADO.NET 2.0을 소개하고자 하는 중요한 이유이기도 합니다. 즉 이 책에서는 ADO.NET의 새로운 기능에 대한 구현 방법을 소개하기 보다는, SQL Server 관리자나 개발자와 같은 데이터베이스 사용자 입장에서 보다 전문적으로 ADO.NET 2.0의 어떤 점이 추가 향상 되었는지 어떠한 기능에 관심을 가져야 하는지 소개하고자 합니다.

***[참고]***
*현재 SQL Server를 연동하는데 사용되는 많은 어플리케이션들이 주로 ADO나 ADO.NET 과 같은 마이크로소프트의 데이터베이스 프로그래밍 API을 사용해서 프로그램 되어 있으며, JDBC를 사용한 자바 프로그램 또한 많이 늘어나고 있습니다. 현장에서 진단분석 및 튜닝 관련 컨설팅을 하면서 중요하게 경험하는 것 중의 하나가 바로 DBMS의 특징을 잘 모른 상태에서 잘못 작성된 코드, 개발 편의성만을 강조해서 만들어진 무성의한 코드, 데이터베이스용 API 기능들을 충분히 활용하지 못하고 있는 코드들이 SQL Server 성능에 상당히 큰 영향을 미치고 있다는 사실입니다. 현업 개발자들 대부분이 바로 그러한 한 점, 잘못 작성된 데이터베이스 관련 코드가 SQL Server에 미치는 성능 문제에 대해서 전혀 깨닫지 못하고 있거나 간과하고 있다는 것입니다.*
*(그도 그럴 것이 많은 개발자들이 안정적이고 성능을 제공할 수 있는 코드를 만드는데 투자할 만한 충분한 시간과 경험을 제공받지 못하고 있습니다)*

ADO.NET 2.0에 향상된 기능들을 데이터베이스 사용자 입장에서 개략적으로 살펴봄으로써 새로 개발되는 데이터베이스용 어플리케이션들이 어떤 기능들을 요구할 수 있는지 이해하는데 도움이 될 것입니다.

## ADO.NET 2.0 주요 기능 리스트

ADO.NET 2.0의 주요 기능들을 간단히 정리하면 다음과 같습니다.

**[표] ADO.NET 2.0 주요 기능**

| | |
|---|---|
| • Provider Factories | • SqlNotificationRequest |
| • Runs w/Partial Trust | • SqlDependency |
| • Server Enumerations | • IsolationLevel Snapshot |
| • Connection String Builder | • Async Commands |
| • Metadata Schemas | • Client Failover |
| • Batch Update Support | • Bulk Import |
| • Provider-Specific Types | • Password Change API |
| • Conflict Detection | • Statistics |
| • Tracing Support | • New Datatypes |
| • Pooling Enhancements | • Promotable Tx |
| • MARS | • AttachDbFileName |

실제로 위에 소개한 기능들 이외에 많은 추가 향상 기능들이 제공됩니다. 다만, 그 내용들은 어플리케이션 개발자 입장에서 보다 중요한 부분이라 생략한 것입니다. 위 내용 중에서 몇 가지 중요한 부분들을 소개합니다.

## 새로운 연결(Connection) 구성 방법

데이터베이스 어플리케이션에서 서버로의 연결을 위한 연결 문자열의 구성은 아주 중요한 부분입니다. ADO.NET 2.0에서는 웹 어플리케이션을 경우 Web.Config 파일 내에서, 윈도우즈 어플리케이션의 경우 App.Config 파일 같은 구성  파일 내의 〈connectionStrings〉 라는 섹션을 도입하고 하부 요소(Element)와 속성(Attribute)을 정의함으로써 기존 연결 문자열(Connection String) 처리를 보다 편리하게 다룰 수 있도록 지원합니다. 이 파일 내의 서버 연결에 대한 정보, 서버 명, 로그인 ID, 암호 등의 모든 정보가 저장되는 것입니다.

C#, VB 어플리케이션 내에서는 해당 *.Config 파일 내의 연결 문자열을 통해서 서버에 접속하고 데이터를 처리하게 됩니다.

다음은 *.Config 파일 내의 연결 문자열에 대한 예제 입니다.

**[예제] connectionStrings 섹션**

```
〈connectionStrings〉
  〈add name=" WinTest.Properties.Settings.NorthwindConnectionString"
      connectionString=" Data Source=YUKON;Initial Catalog=Northwind;User
ID=Bob;Password=p@ssw0rd"
    providerName=" System.Data.SqlClient" /〉
〈/connectionStrings〉
```

## Server Enumeration 기능

이전 버전까지는 네트워크 상에서 어떤 SQL Server가 존재하는지 다른 어떤 DBMS 서버가 존재하는지에 대한 정보를 프로그래밍하는 기능이 기본적으로 제공되지 않았습니다.

ADO.NET 2.0에서는 .NET Framework으로 설치되는 machine.config 파일 내의 .NET Data Provider들(예를 들어, SQLClient Data Provider 등)을 이용해서 네트워크에서 해당 공급자(Provider)에 해당하는 모든 서버 목록과 관련된 정보를 쉽게 구성할 수 있도록 DbProviderFactories 클래스를 제공합니다.
특히 SQL Server에 대해서만 원할 경우, SqlDataSourceEnumerator 클래스를 통해서 보다 쉽게 처리할 수 있습니다.

다음은 몇 가지 관련된 예제 코드입니다.

**[예제] Server Enumeration 예제**

```
// 1) 전체 Data Provider에 대한 목록 얻기
DataTable tblFactories = DbProviderFactories.GetFactoryClasses( );

// 2) 특정 Data Provider만 얻기
DbProviderFactory f = DbProviderFactories.GetFactory(r);
// 혹은
DbProviderFactory f = DbProviderFactories.GetFactory("System.Data.SqlClient");

// 3) 해당 Data Provider에 대해 네트워크 상의 모든 서버 정보 얻기
DbDataSourceEnumerator en = f.CreateDataSourceEnumerator( );
DataTable tblDataSources = en.GetDataSources( );

// 4) SQL Server 정보만 직접 구하는 경우
DataTable                    source                    =
System.Data.Sql.SqlDataSourceEnumerator.Instance.GetDataSources( );
```

## 연결 풀링(Connection Pooling) 향상

서버 측에서뿐만 아니라, 클라이언트 측의 데이터베이스 프로그래밍 API에서도 연결 풀링(Conneciton Pooling)을 지원하고 있으며 이는 데이터베이스 어플리케이션의 성능에 아주 중요한 한 요소입니다. 잘못된 프로그램으로 인해 연결 풀링을 활용하지 못하는 경우도 종종 경험할 수 있습니다.

연결 풀링은 말 그대로, 한 번 구성되고 사용된 연결 개체를 연결이 끊어진 이후에도 풀에 남겨두고 다시 연결될 때 이를 재 사용함으로써 연결을 구성하고 초기화하는데 들어가는 비용과 시간을 줄이기 위한 방법입니다. 이를 통해서 확장성과 성능 향상을 얻을 수가 있습니다.

ADO.NET 이전 버전에서는 연결 풀 내의 연결 개체가 제대로 제거되지 않거나 잘못된 연결 개체가 남아있어서 어플리케이션 수행 시의 문제를 일으킬 수가 있었습니다. ADO.NET 2.0에서는 코드 상에서 이러한 연결 풀의 제거를 직접 수행할 수 있도록 두 가지 메서드를 제공합니다.

■ SqlConnection.ClearPool( )
- 지정된 특정 연결 개체만 제거합니다.

■ SqlConneciton.ClearAllPools( )
- 연결 풀 내의 모든 연결 개체를 제거합니다.

연결 풀링을 모니터링 하기 위한 성능 개체와 카운터도 변경되었습니다.
ADO.NET 이전 버전에서는 .NET CLR Data 개체에서 관련된 연결 정보를 모니터링 할 수 있었지만, ADO.NET 2.0에서는 .NET Data Provider for SqlServer 라는 별도의 성능 개체와 그 관련된 카운터 값들을 제공합니다.

> ***[참고]***
> *실제로 성능 카운터를 사용 시에는 해당 .NET 어플리케이션이 구동 중이어야 합니다.*

## 일괄처리 업데이트(Batch Update)

ADO.NET의 DataTable 혹은 DataSet에서 변경된 데이터를 서버에 반영할 경우, 한 번의 호출에서 다중 행을 반영하도록 지정하는 일괄처리 업데이트가 지원됩니다. 즉 일괄처리(Batch)내에 여러 개의 명령을 하나의 일괄처리 단위로 서버에 전송을 하는 방법입니다.

실제 예제 코드는 다음과 같습니다.

**[예제] Batch Update**

```
// UpdateBatchSize 〉= 0 값으로 설정
sqlDataAdapter1.UpdateBatchSize = int.Parse(textBox1.Text);
sqlDataAdapter1.Update(dsCustomers1,Customers);
```

위 코드는 텍스트박스 컨트롤(textBox1)에 입력된 값을 BatchSize로 지정합니다. 이는 한 일괄처리 내의 행 개수를 의미합니다. 그리고 SqlDataAdapter 개체의 Update 메서드를 호출해서 dsCustomers1 이라는 DataSet 내의 변경된 행을 서버에 적용하도록 호출하게 됩니다. 이 때 실제 서버에 호출되는 일괄처리의 수는 바로 BatchSize에 지정된 행 개수와 DataSet 내에서 변경된 전체 행 수를 나눈 값 만큼 호출되게 됩니다.

■ 일괄처리 업데이트를 적용한 결과로 얻을 수 있는 이득은 다음과 같습니다.

- 네트워크 작업으로 인한 추가 지연을 방지
- 플랜 캐시(Plan Cache)와 재사용에 대한 도움
- 성능 역효과 또한 중요하게 고려

마지막에 성능 역효과를 고려하라는 부분은 무슨 의미일까요?
그것은 실제로 일괄처리 업데이트가 수행되는 내부 방식을 SQL 프로필러 통해서 추적을 해보시면 쉽게 판단하실 수 있습니다. 그 결과를 보시면, 각각의 명령(INSERT, UPDATE, DELETE) 단위로 sp_executesql로 변환되고 이를 서버에 전송해서 실행되는 것을 확인할 수 있습니다. 그 결과로만 보자면 일괄 처리 개념이 적용되지 않는 것 같지만, ADO.NET 팀에 소개에 의하면 SQL Server 내부 수준에서는 배치 단위로 처리가 된다고 합니다.

***[참고]***
*ADO.NET 은 Bata1 과 그 이전 버전, 최근 버전에 이르기까지 계속해서 내용이 변화가 많았습니다. SQL Server 측면에서도 버전 별로 그 내부 동작의 차이가 많이 있습니다. 일괄 처리 업데이트 기능은 또한 Beta2와 CTP 6월 버전이 내부 동작의 차이가 있습니다.*

## 비 동기적인 명령 실행

ADO 에서는 지원이 되었지만, ADO.NET 에서는 지원되지 않았던 아쉬운 부분 중의 하나가 바로 비동기적으로 명령을 실행하는 것입니다. 기본적으로 얻을 수 있는 이득이 바로 클라이언트 측에서 결과 집합의 반환을 대기하면서 발생하는 사용자 인터페이스 상의 차단 현상을 줄일 수 있다는 것입니다. 그 때는 모래 시계가 왠지 싫죠!

ADO.NET 2.0에서 다시 비 동기 처리가 지원됩니다. .NET 코드는 .NET의 비동기 처리 매커니즘과 코딩 패턴을 그대로 사용하므로, 기존 개발자들은 어렵지 않게 사용할 수 있습니다.

우선 연결 문자열에 비 동기 처리를 위한 속성을 지정해야 합니다. (아래 예제 참고)
최종적으로 결과를 반환 받기 위한 접근 방법에 따라서, 전용의 비 동기 호출 메서드를 사용하는 방법, 폴링이나 차단을 위해서 IAsyncResult를 사용하는 방법, 그리고 Callback 메서드(이것이 일반적인 방법)를 사용하는 방법 등을 고려할 수 있는 특히 Callback 메서드를 이용하는 경우엔 서로 다른 쓰레드로 호출되기 때문에, 별도의 동기화 처리를 필요하게 됩니다.

**[예제] 비 동기적인 명령 실행**

1. 연결 문자열 속성 지정

```
"Asynchronous Processing = true;"  혹은 "Async=true;
```

2. 완료될 때까지 폴링하며 대기하는 경우의 코드 예제

```
// 완료에 대한 Polling 수행 코드
IAsyncResult ar = cmd.BeginExecuteReader( );
while(!ar.IsCompleted) {
        // 작업 수행
}
SqlDataReader reader = cmd.EndExecuteReader(ar);
```

3. Callback 메서드를 사용하는 경우의 코드 예제

```
// Callback 사용
IAsyncResult ar = cmd.BeginExecuteReader(
        new AsyncCallback(ExecCallback), null, CommandBehavior.
CloseConnection );

// 다른 작업 수행…


// 필요한 경우 작업 대기(옵션)
ar.AsyncWaitHandle.WaitOne( );

// 작업이 완료되는 이 콜백 메서드가 호출되며, 결과를 처리할 수 있다.
private void ExecCallback( IAsyncResult ar ) {
        SqlDataReader rdr = _cmdAsync.EndExecuteReader(ar);
}
```

## 대량 입력(Bulk Import)

SQL Server에서 대량 로드를 위해서 사용하는 BCP.EXE 혹은 BULK INSERT 명령처럼 ADO.NET 에서 고 성능의 데이터 로드 작업을 위해 SqlBulkCopy 클래스를 제공합니다. 데이터 원본으로 DataRow[ ], DataTable, 그리고 IDataReader 인터페이스를 지원합니다. BatchSize 와 같은 속성 값을 지정해서 일괄처리의 단위, 트랜잭션 단위를 조정하거나 기타 옵션들을 제어할 수 있습니다.

**[예제] SqlBulkCopy 를 사용한 대량 데이터 로드 예제**

```
//destConn은 미리 정의된 목적지 데이터 원본에 대한 연결
SqlBulkCopy bcp = new SqlBulkCopy(destConn, SqlBulkCopyOptions.Default);
bcp.DestinationTableName = "Customers";

// 컬럼명이 일치한다면, 연관을 지을 필요는 없다.
// bcp.ColumnAssociators.Add("customerid", "customerid");// 이름vs. 이름
// bcp.ColumnAssociators.Add(2, 2);          // 컬럼번호 vs. 컬럼번호
bcp.ColumnAssociators.Add(0, 0);             // 위 방법 중 한가지를 선택해서 사용
bcp.ColumnAssociators.Add(1, 1);

/* Batch 크기를 지정할 수 있다. 디폴트는 0 (한 Batch로 모두 복사)
bcp.BatchSize = 50;
bcp.WriteToServer(tblCustomers);

bcp.Close( );
```

ADO.NET 2.0에서 향상된 기능 들 중의 SQL Server 사용자가 관심 있게 볼 만한 몇 가지 주제들을 살펴보았습니다. 이제 이전의 ODBC, OLEDB API에 추가로 SQL Server 2005에 추가된 전용 기능들을 제공하기 위해서 새롭게 개발된 SQL Native Client(SNAC) API와 함께 SQL Server 2005의 전용 향상 기능들과 그 관련된 ADO.NET 2.0의 내용들을 살펴보겠습니다.

## SQL Native Client

윈도우즈 플랫폼에서 데이터베이스 어플리케이션들은 주로 MDAC(Microsoft Data Access Components) 으로 제공되는 ODBC, OLE DB API 혹은 프로그램 가능한 개체 형식으로 제공되는 ADO 등을 사용했습니다. SQL Server 2000은 MDAC 2.6 버전으로 업데이트가 되고, SP3의 경우 MDAC 2.7 버전으로 업데이트가 되었죠.

MDAC은 현재 윈도우즈(혹은 서비스 팩)를 통해서만 배포가 가능하도록 변경된 상태입니다. 따라서, SQL Server 2005와 향후 추가 기능들을 별로도 배포하는 문제가 발생하게 될 것입니다. 이에 대해 SQL Server는 새로운 기능에 대한 지원을 MDAC에서 별도로 분리할 필요가 발생하게 되고 결과적으로 만들어진 새로운 API 가 바로 SQL Native Client 이며, 이는 단일 DLL(\Windir%₩System32₩SQLNCLI.dll)로 존재합니다.

즉, SQL Server 2005의 전용 기능들을 위해서는 SNAC를 사용하게 됩니다. 성능 부분에 있어서도 내부 테스트의 결과 OLE DB와 유사한 성능을 제공하며, 기존 ODBC 에 대해서 20%까지의 성능 향상을 언급하고 있습니다. 따라서 기존 어플리케이션의 경우에도 적합한 경우 SNAC를 사용하도록 조정을 할 수 있을 겁니다. 간단히 연결 문자열만 변경을 하면 됩니다.

■ SQL Native Client를 통해서 지원되는 기능들은
- 데이터베이스 미러링
- 새로운 데이터 형식
- 다중 결과 집합 (MARS)
- 쿼리 알림, 혹은 동적 웹 캐싱
- 암호 변경과 만료
- 스냅숏 격리 수준

등 입니다. 그럼, 각 항목별로 살펴보시죠.

## 데이터베이스 미러링 - 클라이언트 장애 조치

데이터베이스 미러링은 기존 장애조치 클러스터링(Fail-over Clustering)의 자동으로 수행되는 장애조치 기능과 로그 전달(Log Shipping)과 같은 데이터베이스 이중화 기능을 모두 제공하는 데이터 가용성과 확장성을 위한 SQL Server 2005의 새로운 기능입니다.

ADO.NET 2.0에서는 데이터베이스 미러링에 대해서 클라이언트 측의 자동 장애조치 기능을 지원합니다. 즉, 초기 연결 시의 데이터베이스 미러링을 자동으로 인식하고 서비스 장애

발생 시 자동으로 대기 서버(미러 서버)로 연결을 재 시도하는 기능입니다.

■ 클라이언트 장애조치 방법은
- 데이터베이스 미러링을 자동 인식하고 장애조치 준비
- 필요할 경우, 연결 문자열에 장애조치 Partner를 명시적으로 지정하는 것도 가능하며, 이 경우 첫 번째 연결에서 장애조치가 발생하는 경우에도 처리가 가능
- 대기 서버에 대한 PartnerID를 클라이언트 캐시에 저장
- 오류 시 연결을 해제하고 Partner로 재 연결 시도, 재 연결은 투명하게 수행
  해당 트랜잭션은 손실

즉, ADO.NET 프로그램 상에서 별도로 할 작업은 없습니다. 다만, 명시적인 Partner 지정이 필요한 경우에 다음과 같은 연결 문자열을 추가합니다.

**[예제] 명시적인 Partner 지정 시 연결 문자열 추가 내용**

```
Failover Partner=YukonPortal;
```

## 다중 결과 집합 (MARS, Multiple Active Result Sets)

데이터베이스 어플리케이션에서 쿼리를 실행하고 반환된 결과 집합을 처리하기 위해서 커서 형식(CursorType)과 커서 위치(CursorLocation)를 지정하게 됩니다. ADO에서는 디폴트가 서버 측 커서였으며(adUseServer), ADO.NET에선 서버 커서는 지원되지 않고 대신 클라이언트 측 커서만 지원을 했습니다.

클라이언트 측 커서(커서 형식과도 관련이 됩니다)로 선언된 후 쿼리를 실행하는 경우 SQL Server 측에서는 해당 결과 집합을 기본 결과 집합(Default Result Sets) 혹은 일명 Firehose 커서라고 불리는 형식으로 클라이언트에 전송을 합니다. 이 경우 클라이언트는 모든 결과 집합이 반환될 때까지 대기 상태에 있게 되며, 해당 연결(Connection)에서 또 다른 명령을 실행할 수가 없게 됩니다. 두 개 이상의 결과 집합(혹은 행 집합)을 처리하고자 하는 경우에

도 첫 번째 결과 집합을 닫아야만(Resultset.Close) 그 다음 결과 집합 처리가 가능합니다.

만일 두 개 이상의 명령을 유사하게 처리하거나 두 개 이상의 결과 집합을 동시에 열어 놓고 작업을 하고자 한다면, 다음과 같은 방법들을 생각해 볼 수 있습니다.

■ 서버 측 커서 사용
■ 필요한 명령 수 만큼 다중 연결(Connection)을 선언해서 개별적으로 사용

그러나 위와 같은 방법들은 리소스의 낭비, 네트워크 병목과 같은 또 다른 문제들을 불러올 수 있었습니다.

SQL Server 2005에서는 이제 하나의 연결에서 여러 개의 기본 결과 집합을 가질 수 있습니다. 즉, 여러 개의 결과 집합을 열어 놓고 이들을 오가며 데이터를 처리할 수 있고 또한 기본 결과 집합을 처리하면서 INSERT, UPDATE, DELETE 혹은 저장 프로시저 호출과 같은 다른 명령들을 수행할 수 있습니다.

■ 다중 결과 집합으로 인한 성능과 확장성의 이득은
- 물리적인 연결 당 다중 세션을 포함
- 클라이언트와 서버 자원 사용 감소
- 연결 초기화/풀링(Pooling) 로직의 생략
- 비 동기 처리와 조합을 통해서 최대 성능 제공 가능

기존 어플리케이션 개발자들이 데이터 처리에 있어서 불편함을 겪었던 부분들이 상당히 해소될 수 있을 것으로 기대합니다. 특히, 서버 측 커서를 주로 사용하던 다른 플랫폼 개발자들이 ADO.NET을 사용하는 경우의 불편함을 덜 수 있을 겁니다. 물론 이러한 기능이 필요할 때에 제한적으로 사용하는 것이 무엇보다 우선될 것입니다.

**[예제] MARS 예제**

```
//이전 코드 생략
SqlCommand cmdCust = new SqlCommand("select customerid, companyname
                                    from dbo.customers order by customerid",
cnn);
SqlCommand cmdOrd = new SqlCommand("select customerid, orderid
                                   from dbo.orders order by customerid, orderid",
cnn);

cnn.Open( );

SqlDataReader                          rdrCust                          =
cmdCust.ExecuteReader(CommandBehavior.CloseConnection);
SqlDataReader                          rdrOrd                           =
cmdOrd.ExecuteReader(CommandBehavior.CloseConnection);

while (rdrCust.Read( ))
{
        listBox1.Items.Add(rdrCust.GetString(0) + ", " + rdrCust.GetString(1));
}

// rdrCust 가 열려 있는 상태에서 rdrOrd 를 처리
while (rdrOrd.Read( ))
{
        listBox2.Items.Add(rdrOrd.GetString(0) + ", " + rdrOrd.GetInt32(1));
}
//이후 코드 생략
```

## 쿼리 알림(Query Notification)

웹 어플리케이션이나 웹 기반 서비스의 사용이 보편화되면서, 대형 웹 사이트의 경우 동시의 수 많은 서비스 요청을 처리해야만 하는 요구가 발생합니다. 캐시는 하드웨어적으로 혹은 프로그램 적으로 웹 서비스의 동시 처리 능력을 향상시키는데 아주 중요한 역할을 합니다. 그러나 동적으로 임의 시점의 변경될 수 있는 데이터베이스의 경우 일정 간격으로 밖에 처리되는 않는 데이터 동기화 매커니즘을 가진 기존 캐시 처리가 적합하지 않게 됩니다. 물론 파일의 날짜 혹은 버전이 갱신되는 시점에 캐시를 갱신할 수 있는 기능과 테이블의 트리거 등의 개념을 활용하면 데이터베이스가 변경된 시점에서 캐시가 갱신되도록 구성할 수는 있습니다만, 여기에는 또한 여러 가지 어려움이 있습니다.

SQL Server 2005에서는 특히 기존의 웹 어플리케이션에서 아주 중요하게 요구되던 동적 캐시 처리 기능을 지원합니다. 바로 쿼리 알림(Query Notification)이라는 기능입니다. 물론 이 기능은 윈도우즈 어플리케이션에서 훌륭하게 적용될 수 있습니다. 다만 웹 기반에서 웹의 특성 상 별도의 구성을 가지고 있을 뿐입니다.

쿼리 알림은 쿼리에서 반환된 결과 집합에 대해 클라이언트의 로컬(혹은 웹 서버)에서 캐시에 저장을 해 두고, 이후로는 서버의 연결과 데이터 처리가 필요 없이 캐시의 데이터만을 재사용하게 됩니다. 데이터베이스에서 해당 결과 집합의 변경이 발생하게 되면, 클라이언트에게 캐시가 더 이상 유효하지 않음을 알리게 됩니다. 윈도우즈 어플리케이션의 경우 알림을 받기 위한 해당 이벤트가 호출되어서 캐시에 대한 갱신 작업을 수행할 수 있으며, 웹 어플리케이션의 경우 서버로의 재 연결과 쿼리 실행 그리고 캐시 재 설정 작업이 자동으로 수행되며 이후 작업이 반복이 됩니다. 실제로 이러한 내부 동작은 SQL Server 2005의 서비스 브로커 서비스(Service Broker Service) 아키텍처와 결합되어서 지원이 됩니다.

어플리케이션의 유형별, 버전 별 구현 방법의 차이가 있으며 간단히 정리하면 다음과 같습니다.

■ 윈도우즈 어플리케이션의 경우
- System.Data.SqlClient.SqlDependency
  클래스를 사용해서 간단하게 구현할 수 있습니다.

- System.Data.SqlClient.SqlNotificationRequest
  클래스를 사용하는 경우, 저-수준에서 보다 세밀한 제어가 가능합니다.

■ ASP.NET은 자체 구현을 포함하고 있습니다.
- System.Web.Caching.SqlCacheDependency

■ ASP.NET + SQL Server 2000의 경우는 폴링을 통해서 하위 호환성 용으로 제공됩니다.
다음과 같은 별도의 구성 작업이 필요합니다.
- aspnet_regsql 유틸리티를 사용, Cache Notification을 활성화
- OutputCache 선언
- Web.Config 파일 변경

ASP.NET + SQL Server 2000의 경우 데이터베이스 구조의 변경, Polling 간격에 따른 서버 부하와 Cache 처리 등의 주의를 기울여야 합니다.

## 암호 변경/만료

SQL Server 2005에서는 보안 기능에 많은 변화를 경험할 수 있습니다. 그 중에 하나는 SQL Server 인증에 대해서도 윈도우즈 인증과 같은 수준의 보안 기능을 제공하는 것입니다. 예를 들어, 암호 만료 정책이나 윈도우즈 암호 정책을 적용할 수 있는 부분입니다.

따라서 SQL Server 인증의 경우에도 로그인 시 암호를 변경하거나 만료 시 안호를 변경하는 작업 등이 필요하게 됩니다. ADO.NET 2.0에서는 SqlConnection.ChangePassword( ) 정적 메서드(VB 경우 공유 메서드)를 통해서 연결 시 암호를 변경할 수 있도록 지원이 됩니다.

**[예제] SqlConnection.ChangePassword 메서드**

```
// 기존 연결 정보
string pwd = string.Format("data source=yukon;initial catalog=northwind;
                            User ID={0};Password={1};",
                            textBox1.Text.Trim(), textBox2.Text.Trim( ));

// 기존 연결에 대해서 textBox3.Text 의 값으로 암호 변경
SqlConnection.ChangePassword(pwd, textBox3.Text.Trim( ));
```

## 스냅숏 격리(Snapshot Isolation)

SQL Server 2005에 추가된 새로운 격리 수준인 스냅숏 격리 수준을 ADO.NET 2.0에서 트랜잭션 개체 선언 시의 IsolationLevel 옵션을 통해 어플리케이션 측에서 조정을 할 수 있습니다.

**[예제] IsolationLevel.Snapshot**

```
// 트랜잭션 선언 시 격리 수준의 조정
SqlTransaction Tx3 = cnn3.BeginTransaction(IsolationLevel.Snapshot);
sqlCmd3.Transaction = Tx3;
```

# SQL Server Service Broker

서비스지향기술구조(SOA)는 대규모 분산 어플리케이션을 설계하기 위해 중요한 개념으로 자리를 확고하고 있습니다. SOA의 중심부에는 서비스간에 통신을 위한 신뢰할 수 있는 메시지 기반 매커니즘이 위치하고 있습니다. SQL Server 2005에서 서비스 지향 데이터베이스 솔루션을 개발하기 위한 메시지 기반 플랫폼의 역할을 하는 Service Broker에 대해서 소개합니다. 데이터베이스 어플리케이션 간의 메시지교환을 위한 Service Broker의 기능을 살펴보기에 앞서, SOA의 전체적인 개념에 대해서 먼저 알아두어야 합니다.

## SOA란?

최종 사용자에 대한 지연현상을 감소시키거나 확장성을 증대하기 위해서, 대부분의 대용량 시스템은 전통적인 통신 접근방법과 다른 방식의 통신 접근방법을 필요로 하고 있습니다. SOA는 대용량 시스템의 이러한 요구사항을 충족시켜 줄 수 있는 통신 접근방법의 역할을 합니다.

### ■ 정의

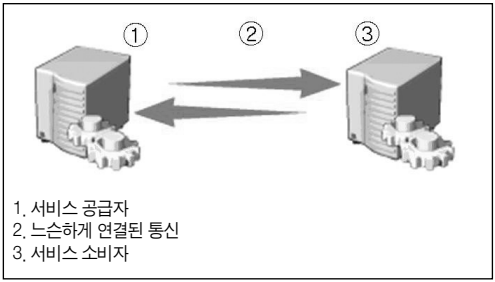
SOA는 소프트웨어 서비스간에 느슨하게 연결된 통신을 보장하는 기술구조의 유형입니다. 서비스는 비즈니스 업무에 대한 소프트웨어적인 구현으로, 통신을 위한 인터페이스와 계약을 제공합니다. 느슨하게 연결된 통신이란 클라이언트와 서비스가 서로에게 비교적 낮은 수준의 의존성을 갖는다는 의미입니다. 예를 들어, 서비스내부에 존재하는 로직은 클라이언트에 영향을 주지 않고 수정이 가능하며, 클라이언트도 서비스내부에 존재하는 로직과 상관없이 수정될 수 있습니다.

### ■ 웹 서비스 사례

소프트웨어는 웹 서비스를 사용하여 인터넷을 통해 통신이 가능합니다. 웹 서비스는 클라이언트가 서비스 공급자가 노출시킨 인터페이스를 통해 서비스를 호출하기 때문에, SOA의 한 사례가 될 수 있습니다. 클라이언트에게 노출된 인터페이스가 웹 서비스 내부의 로직을 호출하며, 서비스 공급자는 클라이언트에 영향을 주지 않고, 내부 로직을 수정할 수 있습니다. 웹 서비스는 XML과 SOAP을 사용하여 서로간에 통신을 수행합니다.

### ■ 메시징 사례

서비스는 Windows NT 4.0 이후에서 제공되는 기능인, 메시지 큐와 같은, 메시징 기술을 사용하여 통신하게 됩니다. 메시징 기술은 서비스 소비자와 서비스 공급자간의 비동기 통신을 가능하게 합니다.




*〈그림. 서비스 지향 기술구조〉*

## SOA 용어

다음 표에는 SOA 관련 용어에 대한 설명이 나타나 있습니다.

| 항목 | 설명 |
|---|---|
| 서비스 | 특정 작업 또는 관련된 작업의 집합으로 구현된 소프트웨어 |
| 서비스공급자 | 계약에 의해 정의된 서비스 규약을 구현한 소프트웨어. 서비스 공급자는 서비스 소비자로서도 동작할 수 있습니다. |
| 서비스소비자 | 서비스 공급자를 호출하는 소프트웨어. 서비스 소비자는 또한 서비스 공급자로서도 동작할 수 있습니다. |
| 메시지 | 서비스간에 통신의 단위 인스턴스. 서비스 공급자와 서비스 소비자는 메시지를 사용하여 통신을 수행합니다.<br>메시지의 양식은 SOA의 구현에 따라 다양한 구조를 가질 수 있습니다. 하지만, 메시지 양식은 플랫폼에 독립적이기 때문에, 잠재 서비스 소비자에게 제한없이 제공될 수 있습니다. |
| 계약 | 두 개의 서비스간에 체결된 각 서비스에서 받아들일 수 있는 메시지에 대한 규약. 계약은 외부 개발자가 프로그래밍 관점에서 서비스가 제공하는 기능을 이해하도록 도와줍니다. |
| 대화(conversation) | 관련된 메시지의 교환. 때로 다수의 메시지가 하나의 작업을 완료하기 위해 필요하기도 합니다. SOA에서는 대화안에 메시지를 연결할 수 있는 매커니즘을 제공해야 합니다. |
| 큐 | 메시지가 처리될 때까지 메시지가 대기하는 저장소. 비동기 통신을 처리하기 위해, 서비스가 해당 메시지를 처리할 때까지, 큐에는 대기중인 메시지가 임시로 저장됩니다. |

## Service Broker의 기능

SQL Server 2005 Service Broker는 데이터베이스 어플리케이션에 메시징 기능을 제공하는 SOA의 구현의 역할을 수행합니다. Service Broker를 통해, SOA 기반 솔루션에서 제공하는, 다음과 같은, 강력한 기능을 사용할 수 있습니다.

| 항목 | 설명 |
|---|---|
| 메시지를 한번에 순서에 따라 수신 | Service Broker는 보내진 순서에 따라 순서대로 한 번에 하나씩 메시지를 수신하기 위해 식별자를 사용합니다. 서비스 공급자와 서비스 소비자의 역할을 하는, 두 개의 Service Broker 엔드포인트간에 지속적인 대화의 일부로 처리됩니다. 서비스 어플리케이션에서는 대화 식별자를 사용하여 별도의 코드를 작성하지 않고서도 정확하게 순서에 따라 하나씩 메시지를 처리할 수 있습니다. 대화 식별자를 통해, 메시지 처리를 위한 정확성을 보장하기 위해서 필요한 코드 작성 노력을 절감할 수 있습니다. |
| 대화을 사용하여 관련 메시지를 조정 | 관련된 메시지를 처리할 때 일관성을 보장하기 위해서, 한 번에 하나의 서비스 프로그램 인스턴스가 대화에 대한 메시지를 큐에서 가져와서 처리하게 할 수 있습니다. 물론, 다른 서비스 프로그램 인스턴스가 해당 대화와 관련없는 메시지를 큐에서 가져올 수는 있습니다. |
| 비동기 전달 | Service Broker는 클라이언트 어플리케이션에서 메시지를 보낸 다음, 서비스 공급자가 결과값을 반환할 때까지 기다리지 않기 위해서, 비동기식 메시지 기반 접근방법을 사용하여 통신합니다. 사실, 서비스 공급자는 클라이언트가 메시지를 보내는 시점에 반드시 운영중일 필요도 없습니다. 클라이언트에서 보내진 메시지는 서비스 공급자가 해당 메시지를 처리할 수 있을 때까지, 큐에 저장되어 대기하게 됩니다. |

| | |
|---|---|
| 스케줄링과<br>처리절차의 유연성 | Service Broker는 현재 작업부하에 따라 메시지를 처리하기 편리한 시점에 메시지를 큐에서 가져와서 서비스 처리절차를 수행합니다. 서버가 너무 과부하 상태인 경우에는 야간시간을 통해 메시지를 배치 처리하게 되어, 주간 업무시간대의 서버의 작업부하를 절감할 수 있습니다. 클라이언트 어플리케이션에서는 메시지를 비동식으로 보내기 때문에, 이러한 변경사항에서 대해서 인식하지 못합니다. |
| 자동 서비스<br>프로그램 활성화 | Service Broker는 큐에 메시지가 도착했을 때 자동으로 서비스 프로그램을 시작하도록 설정할 수 있습니다. 만약, 하나의 서비스 프로그램 인스턴스로 현재 수신된 메시지의 양을 처리할 수 없다면, 구성된 최대값까지, 서비스 프로그램 인스턴스를 추가로 실행합니다. 메시지 처리율이 낮아지면, Service Broker는 필요 없는 서비스 프로그램을 종료시킵니다. 시스템이 재시작되면, Service Broker는 필요한 만큼의 서비스 프로그램만 실행합니다. |
| 데이터베이스와 통합 | Service Broker 는 메시지, 큐, 기타 관련 항목을 모두 데이터베이스로 통합하는 역할을 하기 때문에, 트랜잭션 메시징 기능을 제공합니다. 서비스 프로그램(메시지에 대한 수신, 처리, 클라이언트에 대한 응답을 수행)의 각 반복 작업을 단일 데이터베이스 트랜잭션으로 처리할 수 있습니다. 트랜잭션이 실패하면, 해당 메시지는 추후에 서비스 프로그램이 다시 메시지를 처리할 수 있도록 큐에 재저장됩니다. 관리자 측면에서도, Service Broker와 데이터베이스가 통합됨으로써 많은 관리작업의 부담을 줄일 수 있습니다. 관리자는 별도의 구성작업없이 표준 SQL Server 데이터베이스 백업전략의 일부로 Service Broker 컴포넌트를 포함시켜, 표준 백업절차를 보조하도록 설정할 수 있습니다. |
| 보안이 보장되는 통신 | Service Broker는 네트워크 연결간에 발생하는 대화를 암호화하기 위해 디지털 인증서를 사용합니다. 대화에 참석하는 대상에서는 Windows 사용자 신임장이나 디지털 인증을 사용하여 인증을 받을 수 있습니다. |

## Service Broker의 시스템 기술구조

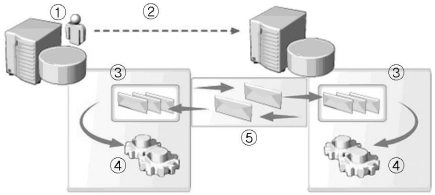
Service Broker 어플리케이션을 개발하기 전에, 먼저 Service Broker 기술구조에 포함된 다섯 가지 SQL Server 개체에 대해서 알아두어야 합니다.

| 개체 | 설명 |
|---|---|
| 메시지유형 | 서비스 소비자는 서비스 공급자에게 메시지를 송신합니다. 서비스 소비자와 서비스 공급자는 각각 서로 교환할 메시지의 유형과 메시지에 대한 유효성 검사 방법을 합의해야 합니다. 소비자 데이터베이스와 공급자 데이터 베이스에 식별가능한 메시지 유형을 생성해야 합니다. |
| 계약 | Service Broker에서는 좀 더 포괄적인 SOA 계약을 구현하기 위해 contract 개체를 제공합니다. Contract 개체는 작업을 완료하기 위해 포함되어야 하는 메시지 유형과 각 메시지 유형을 송신하는 주체에 대해 정의합니다. Contract 개체는 다음과 같이 메시지 유형을 송신하는 주체를 제한합니다.<br>• Initiator Only : 시작자(initiator) 엔드포인트만 지정된 메시지 유형을 보낼 수 있습니다. 시작자 엔드포인트란 대화를 시작하는 엔드포인트를 의미합니다.<br>• Target Only : 대상 엔드포인트에서만 지정된 메시지 유형을 보낼 수 있습니다. 대상 엔드포인트란 시작자 엔드포인트로부터 전송된 메시지를 수신하는 엔드포인트를 의미합니다.<br>• Any Both : 각 엔드포인트에서 지정된 메시지 유형을 보낼 수 있습니다.<br>각 contract 개체를 소비자 데이터베이스와 공급자 데이터베이스에 모두 생성해야 합니다. |

| | |
|---|---|
| 큐 | Service Broker 는 서비스에 메시지를 송신하고, 서비스는 해당 메시지를 큐에 저장합니다. 서비스 소비자는 서비스 공급자 서버가 가용한 상태가 아닌 경우, 해당 메시지를 로컬 큐에 저장합니다. Service Broker는 메시지를 재전송하고, 해당 메시지가 서비스 공급자의 큐에 정상적으로 도착한 경우, 해당 메시지를 로컬 큐에서 삭제합니다. 소비자와 공급자가 동일한 SQL Server 인스턴스에 설치되어 있는 경우에는, 큐가 하나만 사용됩니다.Service Broker는 queue 개체를 메시지를 개별 행으로 보유하고 있는 테이블로 구현합니다. 각 행에는 대화 식별자, 계약 정보 등과 같은 기타 정보도 포함됩니다. Service Broker는 메시지를 수작업으로 조회하는 경우, 큐로부터 메시지를 제거합니다. 또한 큐는 메시지가 도착한 경우 자동으로 해당 메시지를 처리하기 위한 서비스 프로그램을 실행합니다. Service Broker는 큐의 구성정보에 따라, 처리되어야 하는 메시지의 요청량이 많아지게 되는 경우, 해당 메시지를 처리하기 위해서, 하나 이상의 서비스 프로그램 인스턴스를 생성할 수 있습니다. |
| 서비스 프로그램 | 서비스 프로그램은 메시지를 처리하고, 서비스의 실제 처리 로직을 제공합니다. Service Broker는 메시지가 도착했을 때, 자동으로 서비스 프로그램을 활성화시킵니다. 또는 일정을 설정하여 특정 시점에 서비스 프로그램을 실행할 수도 있고, 수작업으로 실행시킬 수도 있습니다. 서비스 프로그램은 작업을 완료하기 위해 대화를 시작한 시작자 엔드포인트에 응답메시지를 보내주어야 합니다. 이러한 응답 메시지도 초기 시작자 엔드포인트에서 정확한 응답메시지를 수신할 수 있도록 하기 위해서 동일한 대화의 일부로 처리됩니다. |
| 서비스 | service 개체는 서비스를 제공하는 접근 가능한 엔드포인트로 표현됩니다. Service 개체는 해당 서비스에 대한 메시지를 저장할 큐의 명칭을 정의하고, 해당 Service 개체가 어떤 계약에 대해서 공급자의 역할을 수행하는지를 정의합니다. 서비스에 계약이 지정되지 않으면, 서비스는 대화를 시작할 수만 있고, 서비스의 공급자가 될 수는 없습니다. |

## Service Broker 대화 기술구조

Service Broker 통신은 메시지를 교환하기 위해 대화를 사용합니다. 대화는 신뢰할 수 있는 메시지를 순서에 따라 보내기 위한 구조로 되어 있습니다. Service Broker 기반 어플리케이션을 개발하기 위해서는, Service Broker의 대화 기술구조에 대해서 알고 있어야 합니다.

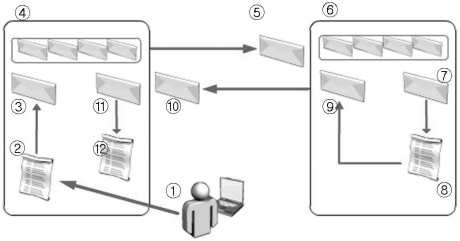



1. 원격서비스바인딩이 원격서비스로 로컬 데이터베이스 사용자 계정과 연결
2. Service Broker가 서로 다른 SQL Server 인스턴스간에 대화가 발생할 때, 메시지를 라우팅
3. 아웃바운드 메시지는 대상에 전달될 때까지 큐에 저장되며, 대상에 전달된 메시지는 처리될 때까지 큐에 저장
4. 서비스 프로그램은 일반적으로 저장 프로시저의 형태이며, 메시지를 읽고 처리하는 역할을 수행
5. 메시지는 서비스간에 통신을 위한 매개체 역할을 수행

*[Service Broker 대화기술구조]*

### ■ Service Broker의 대화처리절차

Service Broker 어플리케이션에서는 단순하게 서비스 공급자에게 개별 메시지를 보내는 기능만 있는 것이 아니라, 부가적인 기능이 좀 더 포함되어 있습니다. 서비스 소비자는 작업이 완료되었을 때, 응답 메시지를 받기를 희망합니다. 하나의 작업을 완료하기 위해 여러 번의 대화가 이루어 질 수 있고, 수 시간에서 수일까지 시간이 소요될 수 있습니다.




1. 직원이 비용정보를 조회하기 위해 클라이언트 어플리케이션을 사용하여 SubmitExpense 저장프로시저를 호출
2. SubmitExpense 저장프로시저가 두 개의 서비스 간에 대화를 시작
3. SubmitExpense 서비스가 대상 엔드포인트에 ExpenseClaim 메시지를 비동기식으로 송신
4. SubmitExpense 서비스 큐
5. ProcessExpense 서비스는 ExpenseClaim 메시지를 수신한 다음, 서비스 인스턴스 식별자를 생성하고, Expense 큐에 해당 메시지를 저장
6. ProcessExpense 서비스 큐
7. Expense 큐에 메시지가 도착하면, ProcessExpense 저장 프로시저 호출
8. ProcessExpense 저장 프로시저에서 ExpenseClaim 메시지 처리
9. ProcessExpense 저장 프로시저는 ClaimResponse 메시지와 EndDialog 메시지를 송신
10. SubmitExpense 서비스는 ClaimResponse 메시지를 수신하여 Expense 큐에 저장
11. ClaimResponse 메시지가 큐에 도착하면, ExpenseResult 저장 프로시저 호출
12. ExpenseResult 저장 프로시저에서 ClaimResponse 메시지의 내용을 점검하고, EndDialog 메시지를 찾으면, 대화를 종료

*[Service Broker 대화처리절차]*

## Service Broker 보안 기술구조

웹 서비스에서 네트워크 연결을 통해 대화를 수행하는 경우, 교환되는 메시지에 대한 보안이 보장되어야 합니다. Service Broker에서는 전송 보안과 대화 보안을 모두 지원합니다.

### ■ 전송보안

Service Broker는 대화를 하기 위해 계약된 두 개의 서비스 간에 보안이 보장되고, 인증된 연결을 설정하기 위해 전송 보안을 사용합니다. 서비스는 원격서버의 HTTP 엔드포인트에 메시지를 송신하기 위해 다음과 같은 인증 옵션을 사용할 수 있습니다.

- None - 서비스에 익명 접근을 허용합니다.
- Enabled - 서비스에 익명 접근을 허용하며, 호출하는 서비스에서 제공하거나, 호출하는 서비스가 요청하는 경우에만 인증을 사용합니다.
- Required - 인증을 필수로 사용합니다.

### ■ 인증

사용되는 인증의 유형은 호출자 어플리케이션의 master 데이터베이스에 존재하는 dbo 사용자가 디지털 인증서를 보유하고 있는지 여부에 따라 결정됩니다. 디지털 인증서가 존재하는 경우, 인증서 기반 인증이 사용됩니다. 디지털 인증서가 없는 경우, Windows 인증이 사용됩니다.

인증서 기반 인증을 사용하기 위해서는, 각 서비스의 관리자별로 master 데이터베이스내에 dbo 사용자에 대한 디지털 인증서를 생성한 다음, 해당 인증서에 대한 공개키를 내보내기 하고, 원격 서비스 관리자에게 해당 공개키를 전달합니다. 각 관리자는 원격 서비스내에 dbo 사용자를 대표하기 위해서, master 데이터베이스에 사용자를 생성한 다음, 새로 생성한 사용자에 원격 관리자에 의해 제공된 공개키를 연결해야 합니다.

Windows 인증을 사용하기 위해서는, 각 서비스의 관리자는 원격 서비스의 서비스 계정으로 사용할 Windows 로그온을 위한 사용자를 master 데이터베이스에 생성해야 합니다.

### ■ 대화(Dialog) 보안

Service Broker 는 원격 서비스간의 메시지를 암호화하고, 원격 사용자에 대한 인증절차를 처리하기 위해 대화 보안을 사용합니다. 대화 보안은 높은수준의(full) 대화 보안과 익명 대화 보안으로 구현될 수 있습니다.

높은 수준의 대화 보안의 경우, 대화를 시작하는 서비스를 호스팅하는 데이터베이스와 호출된 서비스를 호스팅하고 있는 데이터베이스 양쪽 모두에 두 가지 사용자를 포함하고 있어야 합니다. 하나는 원격 서비스에 메시지를 보내는 로컬 사용자이고, 또 하나는 원격 서비스를 호스팅하고 있는 원격 데이터베이스내의 사용자입니다. 대화에 포함되는 각 서비스에는 원격 서비스와 원격 데이터베이스 사용자에 대한 로컬 사용자 계정을 매핑하는 원격 서비스 바인딩 정보가 포함되어야 합니다. 디지털 인증서는 메시지를 수신자의 공개키로 암호화한 다음, 그에 상응하는 개인키로 복호화할 수 있도록, 반드시 각 사용자와 연결되어 있어야 합니다.

익명 대화 보안은 대부분 전체 대화 보안과 동일한 방식으로 동작하지만, 대상 서비스에서 원격 사용자에 대한 접근을 명시적으로 허용할 필요가 없다는 차이가 있습니다. 익명 대화 보안을 사용하면, 대상 서비스는 guest 계정으로 접근을 허용하며, 세션키를 생성하여, 서비스간에 교환되는 메시지를 암호화하기 위해 사용합니다.

## 데이터베이스에서 Service Broker 활성화

Service Broker 서비스는 데이터베이스가 생성될 때, 기본값으로 활성화되지 않습니다. T-SQL 문장을 사용하여, Service Broker 서비스가 활성화되어 있는지 여부를 확인하고, 활성화/비활성화 시킬 수 있습니다. Service Broker가 비활성화되어 있는 경우, 다시 활성화될 때까지 모든 메시지는 전송 큐에 저장되게 됩니다.

### ■ Service Broker 상태 체크

데이터베이스의 Service Broker의 상태를 확인하기 위해서는, 다음 예제와 같이, sys.databases 카탈로그 뷰에서 is_broker_enabled 컬럼을 조회하면 됩니다.

```
SELECT is_broker_enabled
FROM sys.databases
WHERE database_id = db_id( )
```

### ■ Service Broker 활성화

데이터베이스의 Service Broker를 활성화/비활성화 시키기 위해서는, 다음 예제와 같이, ALTER DATABASE T-SQL 명령을 사용합니다.

```
ALTER DATABASE AdventureWorks
SET ENABLE_BROKER
```

## Service Broker 구현

### ■ 계약 생성

Service Broker 어플리케이션을 개발하기 위해서 데이터베이스에 생성해야 하는 개체는 계약과 메시지 유형이 있습니다. 계약은 서비스 간에 교환되는 메시지 유형의 목록을 정의합니다. 계약을 생성하기 위해서는, 다음과 같은 단계의 작업을 수행해야 합니다.

1 단계. 메시지 유형 생성

메시지 유형은 서비스간에 전송되는 데이터의 유형을 정의합니다. VALIDATION 절에는 메시지에 포함될 데이터의 유형을 지정합니다. 메시지에 포함된 데이터의 유형에는 다음과 같은 항목 중 하나를 선택하여 지정합니다.

- NONE - 메시지에 대한 유효성검사를 수행하지 않습니다.
- EMPTY - 메시지에 데이터를 포함하지 않습니다.
- WELL_FORMED_XML - 메시지에 잘 구성된 XML 문자열만 포함되도록 설정합니다. XML이 잘 구성된상태가 아니라면, Service Broker에서 오류를 발생시킵니다.
- VALID_XML - 지정된 XML 스키마 컬렉션에 포함된 스키마를 기준으로 검증된 XML 데이터만 메시지에 포함됩니다.

AUTHORIZATION 절에 메시지 유형의 소유자로 사용할, 특정 데이터베이스 사용자나 역할을 설정할 수 있습니다.

***알림***

*Service Broker 어플리케이션에 메시지 적어도 하나의 메시지 유형은 반드시 포함되어야 하며, 대화에 포함될 각 데이터베이스별로 메시지 유형을 재생성해야 합니다.*

다음과 같은 구문으로 메시지 유형을 생성할 수 있습니다.

```
CREATE MESSAGE TYPE message_type_name
[AUTHORIZATION owner_name]
[VALIDATION =
    {NONE | EMPTY | WELL_FORMED_XML |
    VALID_XML WITH SCHEMA COLLECTION schema_collection_name}]
```

### [따라하기] 메시지 유형 만들기

```
CREATE MESSAGE TYPE
[//Adventure-Works.com/Expenses/ExpenseClaim]
VALIDATION = WELL_FORMED_XML

CREATE MESSAGE TYPE
[//Adventure-Works.com/Expenses/ClaimResponse]
VALIDATION = VALID_XML WITH SCHEMA COLLECTION awschemas
```

**2. 계약 생성**

계약에는 대화에서 사용하게 될 메시지 유형을 정의해야 하고, 메시지 유형이 전송되는 방향(direction)을 정의해야 합니다.

사용할 메시지 유형을 생성한 다음, 다음과 같은 구문으로, 계약을 생성할 수 있습니다.

```
CREATE CONTRACT contract_name
[ AUTHORIZATION owner_name ]
( message_type_name SENT BY { INITIATOR | TARGET | ANY }
[ ,...n] )
```

위의 구문에서 message_type_name 는 사전에 정의된 메시지 유형 이름을 지정합니다. SENT BY는 대화에서 메시지 유형을 전달할 수 있는 방향을 지정합니다. owner_name 에는 계약의 소유자로 설정할 데이터베이스 사용자나 역할을 지정합니다.

[따라하기] 계약 생성

```
CREATE CONTRACT
[//Adventure-Works.com/Expenses/ExpenseSubmission]
( [//Adventure-Works.com/Expenses/ExpenseClaim]
    SENT BY INITIATOR,
    [//Adventure-Works.com/Expenses/ClaimResponse]
    SENT BY TARGET )
```

앞의 예제에서는, 시작자(intiator) 서비스에서만 ExpenseClaim 메시지를 보낼 수 있으며, 대상 서비스에서만 ClaimResponse 메시지를 보낼 수 있습니다.

## ■ 큐 생성

큐는 서비스 프로그램이 메시지를 처리할 수 있는 상태까지, Service Broker가 메시지를 저장할 장소를 정의합니다. 큐를 생성하기 위해서는 다음과 같은 작업절차를 수행해야 합니다.

1. 큐의 이름을 생성합니다..
2. 큐의 가용량을 정의합니다.
3. 활성화 조건 매개변수를 지정합니다.

큐를 생성하기 위한 구문은 다음과 같습니다.

```
CREATE QUEUE queue_name
[ WITH
[ STATUS = { ON | OFF }  [ , ] ]
[ RETENTION = { ON | OFF }  [ , ] ]
[ ACTIVATION (
[ STATUS = { ON | OFF } , ]
```

```
PROCEDURE_NAME = stored_procedure_name ,
MAX_QUEUE_READERS = max_readers ,
EXECUTE AS { SELF | 'user_name' | OWNER } ) ] ]
[ ON { filegroup | [ DEFAULT ] } ]
```

### ■ 서비스 생성

서비스는 서비스 소비자와 서비스 공급자를 연결하는 역할을 합니다. 서비스에는 대화가 시작된 다음, 대화가 종료될 때까지의 범위가 포함됩니다.

서비스를 생성하기 위해서는 다음과 같은 작업을 수행해야 합니다.

- 서비스명을 생성합니다.
- 서비스 소비자가 보낸 메시지를 저장하기 위한 큐를 지정합니다.
- 서비스 공급자의 계약 목록을 지정합니다.
- 대화가 진행되는 동안 메시지를 계속해서 유지할 것인지 여부를 지정합니다.

다음 표에는 CREATE SERVICE 문장에 포함되는 각 구문에 대한 설명이 나타나 있습니다.

| 매개변수 | Description |
|---|---|
| service_name | 서비스를 식별하기 위한 서비스명 |
| AUTHORIZATION | 서비스의 소유자로 지정할 데이터베이스 사용자 또는 역할 |
| ON QUEUE | 메시지를 지정하기 위한 큐를 지정합니다. |
| contract_name | 서비스 공급자가 제공하는 계약의 목록을 지정합니다. 계약이 하나도 지정되지 않으면, 서비스는 대화를 시작하는 기능만 수행합니다. |

[따라하기] 서비스 생성하기-서로 다른 큐를 사용하는 두 개의 서비스

```
CREATE SERVICE [//Adventure-Works.com/SubmitExpense]
ON QUEUE ExpensesInitiator
( [//Adventure-Works.com/Expenses/ProcessExpense] )

CREATE SERVICE [//Adventure-Works.com/ProcessExpense]
ON QUEUE ExpensesTarget
( [//Adventure-Works.com/Expenses/ProcessExpense] )
```

## ■ 메시지 송신

Service Broker 어플리케이션에서 수행해야 하는 가장 기본적인 작업 중 하나는 서비스 소비자로부터 서비스 공급자에게로 메시지를 보내는 것입니다. 메시지를 보내기 위해서는, 먼저 계약, 큐, 서비스와 같은, 관련 Service Broker 개체가 먼저 생성되어 있어야 합니다.

메시지를 보내기 위해서는, 다음과 같은 작업절차를 수행해야 합니다.

### 1. 대화를 관리하기 위한 식별자 변수 선언

메시지를 보내기 전에, 대화를 관리하기 위한 식별자 변수를 선언해야 합니다. 대화를 관리하기 위한 변수는, 다음 예제와 같이, 반드시 uniqueidentifier 데이터형을 사용해서 선언해야 합니다.

```
DECLARE @dialog_handle uniqueidentifier
```

### 2. 대화 시작

BEGIN DIALOG CONVERSATION 명령을 사용하여 대화를 시작할 수 있습니다.

BEGIN DIALOG CONVERSATION 명령에 대한 구문은 다음과 같습니다.

```
BEGIN DIALOG [CONVERSATION] dialog_handle_identifier
    FROM SERVICE service_name
    TO SERVICE 'service_name' [ , broker_instance ]
    ON CONTRACT contract_name
    [ WITH
    [ { RELATED_CONVERSATION = conversation_handle
        | RELATED_CONVERSATION_GROUP = conversation_group_id } ]
[ [ , ] LIFETIME = dialog_lifetime ]
 [ , ] ENCRYPTION = { ON | OFF } ] ]
```

다음 표에는 BEGIN DIALOG 구문에서 사용하는 매개변수 목록이 나타나 있습니다.

| 매개변수 | 설명 |
|---|---|
| dialog_handle _identifier | 새로 생성된 대화 식별자 핸들을 반환합니다. 새 식별자를 저장하기 위해 대화핸들변수가 사용됩니다. |
| FROM SERVICE | 메시지를 보내는 서비스를 지정합니다. 서비스를 초기화할 때 사용할 서비스명을 지정해야 합니다. 서비스에서 지정한 큐로 대상 서비스에서 반환한, 오류나 대화종료 메시지와 같은, 모든 메시지가 수신됩니다. |
| TO SERVICE | 대화를 시작하려고 하는 대상 서비스의 이름을 지정합니다. Service_name은 문자열 상수값이거나, 암시적으로 문자열로 형변환할 수 있는 변수로 지정해야 합니다. 또한, 대소문자를 구별하는 것에 유의해야 합니다. |
| broker_instance | 대상 서비스가 하나 이상의 데이터베이스에서 호스팅되고 있는 경우, 대상 데이터베이스를 지정하기 위해 사용할 수 있는 유일한 식별자입니다. |

| | |
|---|---|
| ON CONTRACT | 서비스에서 통신하기 위해서 사용하려는 계약명을 지정합니다. |
| WITH | RELATED_CONVERSATION 식별자나 RELATED_ CONVERSATION _GROUP 식별자를 사용하여 기존 대화에 새 대화를 연결하기 위해서 사용합니다. 또한, LIFETIME 옵션을 사용하여 대화가 오픈된 상태로 유지되는 최대지속시간을 초 단위로 지정할 수 있습니다. LIFETIME 옵션이 지정되지 않으면, 양쪽 엔드포인트에서 명시적으로 대화를 종료해 주어야 합니다. |
| ENCRYPTION | 대화에서 주고 받는 모든 메시지를 암호화할 것인지여부를 지정합니다. 기본 옵션은 서로 다른 SQL Server 인스턴스간에는 메시지를 암호화하는 것입니다. 동일한 인스턴스에 설치된 서비스 간의 대화는 암호화되지 않습니다. |

**[따라하기] 대화시작**

```
BEGIN DIALOG CONVERSATION @dialog_handle
    FROM SERVICE [//Adventure-Works.com/SubmitExpense]
    TO SERVICE '//Adventure-Works.com/ProcessExpense'
    ON CONTRACT [//Adventure-Works.com/Expenses/ProcessExpense]
```

3. 메시지 송신

대화 핸들 식별자가 설정되면, SEND 명령을 사용하여 메시지를 송신하게 됩니다. SEND 명령에 대한 구문은 다음과 같습니다.

```
SEND ON CONVERSATION conversation_handle
    MESSAGE TYPE message_type_name
    [ ( message_body_expression ) ]
```

다음 표는 SEND 명령에서 사용하는 매개변수의 목록을 나타냅니다.

| 매개변수 | 설명 |
|---|---|
| conversation_handle | 대화 식별자에 대한 핸들값을 지정합니다. BEGIN DIALOG 명령에 의해서 반환된 uniqueidentifier 데이터형 핸들값을 지정합니다. 단순히 대화의 대상 측에서 대화를 시작한 측으로 응답메시지를 회신하기 위해서라면, 대화를 시작한 측에서 보내온 메시지 내에 설정된 대화 식별자 핸들값을 지정합니다. |
| MESSAGE TYPE | 계약에서 정의된 현재 엔드포인트에서 전송할 수 있는 메시지 유형을 지정해야 합니다. 시작자측 엔드포인트에서는 SENT BY INITIATOR, SENT BY ANY로 표시된 메시지 유형을 지정할 수 있습니다. 또한, 대상측 엔드포인트에서는, SENT BY TARGET, SENT BY ANY로 표시된 메시지 유형을 지정할 수 있습니다. |
| message_body_expression | 송신할 메시지의 본문을 지정합니다. 메시지 본문은 적절한 메시지 유형에 일치하는 형식으로 지정되어야 합니다. |

### [따라하기] 메시지송신

```
DECLARE @msgString NVARCHAR(MAX)
SET @msgString = NCHAR(0xFEFF) + N' <root> xml data </root> '
;SEND ON CONVERSATION @dialog_handle
MESSAGE TYPE [//Adventure-Works.com/Expenses/ExpenseClaim]
(@msgString)
```

앞의 예제는 메시지가 XML 본문으로 구성되었다고 가정합니다. XML 형식 문자열에 대해서 메시지 송신 작업을 하기 위해서는, XML 데이터 앞에 바이트 순서 표시를 포함한 문자열인지를 확인해야 합니다. NCHAR(0xFEFF) 값이 바이트 순서 표시로 사용됩니다.

**팁**

*SEND 문장이 배치나 저장 프로시저의 첫 문장이 아닌 경우에는, 반드시 세미콜론(;) T-SQL 종결자를 사용해야 합니다.*

## ■ 메시지 수신

메시지가 보내지면, 서비스 프로그램은 큐로부터 보내진 메시지를 읽어 들여, 메시지 본문에 대한 처리를 수행합니다. 서비스 프로그램이 메시지를 읽는 방법은 큐를 설정할 때 선택한 방법에 따라 결정됩니다. 물론, 공통적인 메시지 수신 절차는 변경되지 않습니다.

메시지를 수신하기 위해서는 다음과 같은 작업절차가 수행됩니다.

**1 단계. 메시지 상세정보를 저장하기 위한 변수선언**

메시지의 개별 컬럼을 저장하기 위한 로컬 변수를 생성해야 합니다. 작업을 위해 공통적으로 필요한 컬럼에는 conversation_handle, message_type_name, message_body 등이 있습니다.

**2 단계. RECEIVE 명령 호출**

RECEIVE 명령은 큐에 저장된 메시지를 처리하여, 테이블 기반 결과값과 같은 결과를 만들어 내는 작업을 수행하기 위해, 지정된 큐로부터 하나 또는 그 이상의 메시지를 가져오는 역할을 합니다. RECEIVE 명령에 별도로 TOP 매개변수를 지정하지 않으면 동일한 서비스 인스턴스 식별자에 소속된 모든 메시지를 큐에서 제거하여 가져옵니다. RECEVIE 명령이 큐에서 메시지를 제거하는 동안에는, 다른 서비스 프로그램 인스턴스가 RECEIVE 명령의 대상이 되는 메시지에 대한 작업을 수행할 수 없으며, 동일한 서비스 인스턴스 식별자가 소유한 메시지에 대한 다른 작업도 수행할 수 없습니다. RECEIVE 명령은 대화에 포함된 각 메시지의 message_sequence_order 값의 순서에 따라 메시지를 큐에서 제거합니다.

RECEIVE 명령에 대한 구문은 다음과 같습니다.

```
[ WAITFOR ( ]
    RECEIVE [ TOP (n) ]
        〈column_specifier〉 [ ,...n ]
        FROM queue_name
        [ INTO table_variable ]
        [ WHERE { conversation_handle = conversation_handle
        | conversation_group_id = conversation_group_id } ]
[ ) ] [ , TIMEOUT timeout ]
```

다음 표는 RECEIVE 명령에서 사용할 수 매개변수 목록을 나타냅니다.

| 매개변수 | 설명 |
|---|---|
| WAITFOR | 메시지가 큐에 도착할 때까지 RECEIVE 명령에 대한 처리를 유보합니다. TIMEOUT 값을 밀리초 단위로 지정하면, RECEIVE 명령은 도착한 메시지가 없는 경우, 빈 결과값을 반환하기 전에 주어진 TIMEOUT 값만큼 대기합니다. TIMEOUT 값을 지정하지 않거나, -1 을 지정하면, RECEIVE 명령은 메시지가 도착할 때까지 대기하게 됩니다. |
| TOP | 큐로부터 몇 개의 메시지를 수신할 것인지를 지정합니다. TOP 매개변수를 지정하지 않으면, 특정 서비스 인스턴스 식별자에 해당하는 모든 메시지를 수신합니다. 일반적으로, 개별 메시지 단위로 작업하기 위해, 1로 지정합니다. |
| column_specifier | 메시지 처리 로직에 사용할 컬럼목록을 지정하기 위해 사용합니다. 완료 목록에 대해서는 SQL Server 2005 온라인 도움말을 참조하십시오. |
| FROM | 메시지를 수신하기 위해서 체크할 큐의 이름을 지정합니다. |

| | |
|---|---|
| INTO | 로컬 변수가 아닌 특정 테이블에 결과값을 지정하기 위해서 사용합니다. 동시에 다수의 메시지를 처리해야 하는 상황에서는 로컬 변수 대신 테이블을 사용하는 방법을 사용해야 합니다. |
| WHERE | 특정 대화 핸들이나 대화 그룹 식별자에 대한 메시지만 반환할 수 있도록 제한하기 위해서 사용합니다. WHERE 절에 다른 컬럼이 사용될 수는 없습니다. |

다음의 예제는 하나의 메시지를 수신하여, 해당 메시지에 포함된 세 개의 컬럼의 정보를 로컬 변수에 저장하는 방법을 나타냅니다.

### [따라하기] 메시지 수신

```
;RECEIVE TOP(1) @conversation = conversation_handle,
    @msgType = message_type_name, @msg = message_body
FROM ExpenseQueue
```

***팁***

*RECEIVE 명령에서 메시지를 발견했는지를 체크하기 위해 @@ROWCOUNT 값을 확인해야 합니다. @@ROWCOUNT 값이 0 인 경우에는, 큐에 메시지가 하나도 저장되어 있지 않은 상태를 의미합니다.*

**3 단계. 메시지 유형 체크 및 메시지 처리**

적절한 메시지 유형을 처리하는 것인지 확인하기 위해, IF 문장을 조합하여, 메시지 결과집합에 포함된 message_type_name 컬럼의 값을 체크합니다. 체크한 결과에 포함될 수 있는 메시지 유형에는 사전에 정의된 메시지 유형,
http://schemas.microsoft.com/SQL/ServiceBroker/Error 메시지 유형의 오류 메시지,

http://schemas.Microsoft.com/SQL/ServiceBroker/DialogTimer 메시지 유형의 대화 타임아웃 메시지,
http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog 메시지 유형의 대화종료 메시지가 있습니다.

**4 단계. 대화를 종료하기 위해 END CONVERSATION 호출**

서비스 공급자가 작업을 완료하면, END CONVERSATION 문장을 사용하여, 대화를 종료할 수 있습니다. END CONVERSATION 명령에 대한 구문은 다음과 같습니다.

```
END CONVERSATION conversation_handle
    [ [ WITH ERROR = failure_code DESCRIPTION = failure_text ]
        | [ WITH CLEANUP ] ]
```

conversation_handle 는 종료하고자 하는 대화에 대한 식별자를 지정합니다. WITH ERROR 절을 사용하여 오류 코드와 설명을 지정할 수 있습니다.

WITH CLEANUP 절은 대화 상대자에게 통지 없이 대화에 포함된 모든 메시지와 메타데이터를 삭제하기 위해서 사용합니다. SQL Server 는 대화를 구성하고 있는 엔드포인트와, 전송 큐에 포함된 대화에 대한 모든 메시지, 서비스 큐에 포함된 대화에 대한 모든 메시지를 삭제합니다.

# Notification Service

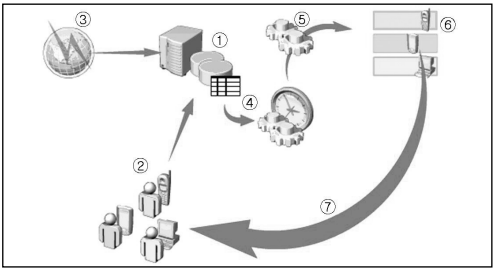
시기적절하게 필요한 핵심 정보를 전달하는 것은 대부분의 비즈니스 솔루션의 핵심 목표입니다. Notification Service는 SQL Server 기반 알림 플랫폼의 역할을 수행합니다. Notification Service를 사용하여 개발자는 알림서비스 어플리케이션을 신속하게 개발할 수 있고 사용자는 알림서비스 어플리케이션을 통해 개인화된 정보를 적시에 제공받을 수 있습니다. 여러 종류의 통신 프로토콜과 장비를 사용하는 구독자에게 자동화된 알림 기능을 제공할 수 있습니다.

## Notificaiton Service 기술구조 소개

Notification Service는 구독관리, 이벤트 수집, 알림 생성, 알림 전달로 구성됩니다. Notification Service 기반 솔루션을 설계하고 구현하기 위해서는 각 구성요소에 대해 상세한 정보를 알고 있어야 합니다.

### ■ SQL Server 2005의 Notification Service 소개

Notification Services 는 특정 이벤트가 발생하였을 때 구독자에게 알림을 전달하기 위한 프레임워크입니다.




*[Notification Service의 구조]*

Notification Services 는 두 가지 종류의 XML 파일을 사용하여 구성됩니다. 하나는 Notification Service 인스턴스에 대한 설정정보를 포함하고 있는 구성 파일이고, 또 다른 하나는 Notification Service 인스턴스에 개별 알림 어플리케이션에 대한 설정정보를 포함하고 있는 어플리케이션 정의 파일(ADF)입니다. 구성 파일과 어플리케이션 정의 파일은 SQL Server 데이터베이스를 정의하기 위해 사용됩니다.

이벤트는 사용자가 알림을 받고자 하는, 주식가격변동, 주문배송정보, 축구경기득점소식 등과 같은 특정 행위입니다. 이벤트는 이벤트 공급자에 의해 Notification Service로 알려지게 되고, 데이터베이스 테이블에 저장됩니다.

구독 관리 어플리케이션을 통해 사용자는 자신을 구독자로 등록합니다. 구독 관리 어플리케이션을 통해, 구독자에 대한 상세정보를 지정하고, 특정 이벤트에 대한 구독 상세 정보를 설정하고, 알림을 받을 디바이스에 대한 정보를 지정합니다. 사용자는 특정 이벤트가 발생하거나 일정 주기의 간격으로 알림이 전송될 수 있도록 구독을 생성할 수 있습니다.

Notification Service는 내부 생성자 컴포넌트를 사용하여 구독에 대한 이벤트가 발생하면 적절한 알림을 생성합니다. 이벤트 정보와 구독 정보가 테이블에 저장되어 있기 때문에, 이벤트 생성자는 집합 기반 SQL 쿼리를 사용하여, 효과적으로 이벤트 정보와 구독 정보간의 비교작업을 수행할 수 있습니다.

생성된 알림 메시지는 내부 배포 컴포넌트로 전달되어, 적절한 형식으로 변환된 다음, 구독자가 지정한 디바이스로 전송됩니다. 알림 메시지는 Email, SMS 문자 메시지, .NET 알림 메시지 등 다양한 형식으로 전송될 수 있습니다.

### ■ Notification Service 기술구조

SQL Server 2005의 Notification Service 솔루션은 하나 또는 그 이상의 인스턴스와 각 인스턴스별로 하나 또는 그 이상 호스팅되는 어플리케이션으로 구성됩니다. 인스턴스와 어플리케이션의 관계를 이해하는 것이 Notification Service 기반 솔루션을 설계하고 계획하는 과정에 도움을 줄 수 있습니다.

**인스턴스**

Notification Service 인스턴스는 XML 구성 파일에 의해서 정의되며, SQL Server 데이터베이스에 데이터를 저장하고 있는 Windows 서비스로 구현됩니다. 인스턴스에는 해당 인스턴스에 호스팅되는 어플리케이션에서 공유하여 사용하는 구독자 정보를 저장하고 있으며, 알림의 생성 및 전달을 통제하는 역할을 수행합니다. 인스턴스에 대한 Windows 서비스는 ns$instance_name 으로 지정되며, 인스턴스에 연결된 데이터베이스는 기본값으로 instance_nameNSMain으로 지정됩니다.

**어플리케이션**

각 인스턴스는 여러 개의 어플리케이션을 통제합니다. 어플리케이션은 어플리케이션 정의 파일(ADF)에 의해서 정의되며, 기본값으로, instance_nameNSapplication_name로 명명되는 데이터베이스로 구현됩니다. 어플리케이션에는 이벤트, 구독, 알림에 대한 정보가 저장됩니다.

## ■ 구독관리

사용자는 개발자가 개발한 구독 관리 어플리케이션을 통해 이벤트를 구독합니다. Notification Service는 구독 관리 어플리케이션을 쉽게 개발할 수 있도록, 구독 관리 개체 라이브러리를 제공합니다. 구독 관리 개체는 Subscriber 개체, SubscriberDevice 개체, Subscription 개체로 구성됩니다. 구독 관리 개체는 각각 필요한 데이터를 Notification Service 데이터베이스에 저장됩니다. 저장된 구독 관리 개체 데이터는 알림을 생성하기 위해 사용됩니다.

## ■ 이벤트 수집

이벤트는 이벤트 공급자에 의해 수집되며, 세 가지 종류의 API 중 하나를 통해, Notification Service로 전달됩니다. 관리되는 .NET API는 Event 개체와 Event Collector 개체를 제공하며, 관리되는 코드를 사용하여 이벤트를 전달할 수 있습니다. COM API는 COM 기반 어플리케이션과 관리되는 API간에 상호운영성을 보장하기 위해 사용됩니다. XML API는 이벤트 원본을 이벤트 데이터를 포함한 XML 파일로 전달한 다음, SQL Server 의 XML 대량 복사 컴포넌트를 사용하여 Notification Service로 전달하기 위해 사용합니다. SQL API는 저장 프

로시저를 통해, 이벤트를 전달하기 위해서 사용됩니다. 어떤 API가 사용되었느냐에 따라, 해당 API에서 전달하는 이벤트 데이터가 Notification Service로 전달되어, 데이터베이스에 저장됩니다.

이벤트는 일반적으로 효율적인 처리를 위해 배치형태로 전달됩니다. 개발자는 각 API를 사용하여 사용자정의 이벤트 공급자를 생성할 수 있습니다. 또한, Notification Service에서는 SQL Server 이벤트 공급자나 FileSystemWatcher 이벤트 공급자 등과 같은 표준 이벤트 공급자를 제공합니다.

### ■ 알림 생성

어플리케이션 정의 파일에 지정된 시간 간격에(퀀텀(Quantum)라고 부름) 따라, 주기적으로 알림 생성자는 어플리케이션 개발자가 지정한 규칙에 따라, 이벤트 데이터와 구독 데이터에 대해 T-SQL 쿼리를 실행합니다. 지정된 규칙에 해당하는 알림이 발견되면, 배치작업으로 처리되어, Notifications 테이블에 저장됩니다.

### ■ 알림 서식 지정 및 배달

알림 생성자에서 알림을 생성한 다음, 배포자는 Notifications 테이블에 저장된 데이터를 사용하여, 알림 메시지에 필요한 서식을 설정하고, 적절한 컨텐츠 포맷터(예를 들어, XSLT 스타일시트)에 맞도록 데이터를 변환합니다. 그 다음, 배포자는 서식이 적용된 알림 메시지를, 구독자가 지정한 디바이스로 전송하기 위해, 적절한 배달 채널에 전달합니다.

배달 채널은 Notification Service 어플리케이션에서 알림 메시지를 전달하기 위해서 사용할, 지정된 프로토콜에 적합하도록 생성할 수 있습니다. Notification Service 는 명시적으로 SMTP, HTTP, 파일 시스템에 알림 메시지를 저장하기 위한 File 프로토콜과 같은 세 가지 프로토콜을 지정합니다. 또한 사용자 정의 프로토콜을 생성할 수도 있습니다.

## Notification Service 프로세스

### ■ 어플리케이션 개발 프로세스

Notification Service 어플리케이션을 개발하기 전에, 솔루션을 개발하기 위해 반드시 수행해야 하는 작업에 대해 알아두어야 합니다. 개발 프로세스에 대한 이해는 개발 프로젝트와 자원을 효율적으로 운용할 수 있게 해 줍니다.

다음은 Notification Service 솔루션을 개발하기 위해 수행해야 하는 작업단계입니다.

| 작업단계 | 설명 |
|---|---|
| Notification Service에 적합한 시나리오 설계 구성정보 및 ADF 파일 생성 | 솔루션 설계자가 Notification Service 솔루션에 대한 계획을 수립하는 단계입니다.<br>구성정보 파일 및 어플리케이션 정의 파일은 XML 형식이기 때문에, XML에 대한 지식을 보유하고 있는 개발자에 의해서 작성되어야 합니다. |
| SQL Server Management Studio, NSControl을 사용하여 인스턴스를 등록하고 어플리케이션 개발 | 개발자 또는 관리자는 Notification Service 솔루션을 개발하기 위해 SQL Server Management Studio 나 NSControl 명령줄 유틸리티와 같은 도구에 대한 지식을 알고 있어야 합니다. |
| 구독 관리 어플리케이션 개발 | Notification Service 솔루션에는 구독 관리 개체를 사용하여 구독 관리 어플리케이션이 포함되어야 하기 때문에, 개발자는 .NET이나 COM 프로그래밍 기술을 보유하고 있어야 합니다. |
| 이벤트 공급자를 사용하여 이벤트 수집 구현 | 이벤트 기반 구독을 활성화하기 위해서는, 이벤트 공급자를 사용하여 이벤트 수집을 구현해야 합니다. Notification Service에 포함된 표준 이벤트 공급자 중에 하나를 사용할 수도 있고, 이벤트 API를 사용하여 사용자정의 이벤트 공급자를 개발할 수도 있습니다. |

## ■ 설계 고려사항

어플리케이션 개발을 시작하기 전에, 먼저 요구사항을 충족하는 솔루션을 설계해야 합니다. Notification Service 솔루션을 설계하기 위한 시간을 투자함으로써, Notification Service 솔루션을 개발하고 배포하는 과정을 좀 더 쉽게 처리할 수 있습니다.

Notification Service 솔루션을 설계할 때 고려해야 하는 고려사항은 다음과 같습니다.

| 고려사항 | 설명 |
|---|---|
| 인스턴스와 어플리케이션 구성 | 필요한 Notification Service 인스턴스 수와 각 인스턴스에 어떤 어플리케이션을 할당할 것인지를 결정합니다. 일반적으로 하나의 인스턴스에는 하나의 어플리케이션을 할당하는 것이 적합합니다. 하지만, 구독자 데이터를 공유할 수 있는 경우, 단일 인스턴스에 여러 개의 어플리케이션을 할당할 수 있습니다. |
| 데이터 구조 설계 | 구독, 이벤트, 알림 테이블에 알림 메시지에 필요한 데이터를 제공하기 위해서 사용자 정의 필드를 추가할 수 있습니다. |
| 매치 규칙 | 매치 규칙은 구독 데이터와 이벤트 데이터로부터 알림을 생성하기 위해 사용합니다. 어플리케이션에서 알림을 생성하기 위해 필요한 T-SQL 명령을 설계해야 합니다. |
| 기록(chronicle) 테이블 | 일부 어플리케이션의 경우, 이벤트, 구독, 알림 데이터의 이력을 기록 테이블에 저장해야 하는 요건이 발생할 수 있습니다. 이력 데이터가 필요한 경우, 이력 데이터를 관리하기 위한 테이블과 규칙을 설계해야 합니다. |
| 알림 형식 | 알림 메시지에 대한 형식도 설계되어야 합니다. 표준 XSLT 포맷터를 사용하는 경우에는, 적절한 XSLT 스타일시트를 설계해야 합니다. 관리되는 코드를 사용하여 컨텐츠 포맷터를 설계할 수도 있습니다. |

| | |
|---|---|
| 알림 전송 | Notification Service 솔루션에서 지원할 프로토콜을 결정해야 합니다. 표준 파일, HTTP, SMTP 프로토콜을 지정할 수 있고, 별도의 사용자정의 프로토콜 공급자를 생성할 수도 있습니다. 지원할 프로토콜을 지정한 다음에는, 지원할 구독자 디바이스도 설계해야 합니다. |
| 물리적 배포 | 테스트나 배포 환경에서는 단일 서버에 Notification Service 솔루션을 구현할 수 있습니다. 하지만, 운영환경에서는 여러 대의 서버를 사용하게 되며, 알림 생성자나 배포자 기능으로부터 Notification Service 데이터 저장소를 분리합니다. |

### ■ 권장사항

Notification Service 어플리케이션을 생성할 때 고려해야 하는 권장사항이 있습니다. 다음 표는 Notification Service 어플리케이션이 최적의 성능과 기능을 제공할 수 있도록 보장하기 위한 가이드라인이 나타나 있습니다.

| 권장사항 | 설명 |
|---|---|
| 이벤트 배치 처리 | 개별로 이벤트를 처리할 수 있다고 하더라도, 집합 기반 작업을 수행하여, 배치단위로 이벤트를 처리하면, Notification Service 을 최적화할 수 있고, 성능을 향상시킬 수 있습니다. |
| 다이제스트 또는 멀티캐스트 배달지정 | 다이제스트 알림을 사용하여, 단일 구독자에 대한 알림을 그룹화하거나, 여러 구독자에 대한 알림을 단일 멀티 캐스트 알림으로 조합하면, Notification Service 시스템에 대한 작업부하를 줄일 수 있으며, 성능도 개선할 수 있습니다. |

| | |
|---|---|
| 이벤트에 구독자에 특화된 데이터 사용금지 | 단일 사용자에 특화된 이벤트를 생성하면, 이벤트의 종류가 매우 많아지기 때문에, 성능에 악영향을 미칩니다. Notification Service 솔루션을 설계할 때는, 각 이벤트를 일반화하여 생성하고, 구독자에 의해서 필터링하는 방법을 사용해야, 좀 더 효율적으로 알림을 생성할 수 있습니다. |
| 기록 테이블에 인덱스 추가 | 기록 테이블에 인덱스를 추가하여 성능을 개선할 수 있습니다. |
| 기간이 경과된 데이터 삭제 | 기록 테이블에서 기간이 경과된 데이터를 압축하기 위한 규칙을 수립하고, 기간이 경과된 이벤트 데이터를 삭제합니다. |

## Notification Service 솔루션 구현

Notification Service 어플리케이션을 구현하는 방법에 대해서 살펴봅니다. 구성 파일과 어플리케이션 정의 파일을 구성하는 방법, SQL Server Management Studio나 NSControl 유틸리티를 사용하는 방법, 이벤트 공급자를 개발하기 위해 사용할 수 있는 옵션과, 구독 관리 어플리케이션을 개발하는 방법을 살펴봅니다.

### ■ 구성 파일

XML 구성 파일은 Notification Service 인스턴스를 정의하기 위해 생성합니다. 필요한 Notification Service 솔루션을 개발하기 위해서는, 구성 파일에 포함되는 핵심 정보에 대해서 알아두어야 합니다.

| 구성요소 | 설명 |
| --- | --- |
| 매개변수 | SQL Server Management Studio나 NSControl 유틸리티를 사용하여 구성 파일에서 사용할 사용자 매개변수를 정의할 수 있습니다. 파일 내부에서 각 매개변수를 퍼센트(%) 문자를 지정하여 참조할 수 있습니다. 환경 변수는 구성파일내에서 자동으로 접근할 수 있습니다. 〈ParameterDefaults〉 엘리먼트를 사용하여 매개변수에 기본값을 할당할 수 있습니다. 〈ParameterDefaults〉 엘리먼트를 사용하면, 매개변수를 명령줄에서 지정하지 않은 경우라고 할지라도, 파일에서 해당 매개변수를 값을 사용할 수 있습니다. |
| 파일 문서화 | 구성 파일 자체에 대한 정보를 문서화하기 위해 파일에 〈Version〉 엘리먼트와 〈History〉 엘리먼트를 사용합니다. 구성 파일에 대한 문서정보를 포함하고 있으면, 솔루션을 좀 더 쉽게 관리할 수 있고, 유지보수성을 향상시킬 수 있습니다. |
| 인스턴스 설정 | 구성파일에는 인스턴스 명칭을 지정하기 위한 〈InstanceName〉 엘리먼트와 인스턴스 데이터베이스를 생성할 SQL Server 인스턴스를 지정하기 위한 〈SqlServerSystem〉 엘리먼트가 반드시 포함되어야 합니다. 필요에 따라 인스턴스 데이터베이스를 생성할 때, 데이터베이스 파일을 정의할 정보를 지정하기 위해 〈Database〉 엘리먼트를 포함시킬 수 있습니다. 〈Database〉 엘리먼트가 생성되면, 인스턴스 데이터베이스는 기존 SQL Server 데이터베이스와 같이, model 데이터베이스를 복사하여 생성합니다. |

| | |
|---|---|
| 어플리케이션 설정 | 인스턴스에 소속된 어플리케이션을 지정하기 위해서 〈Applications〉 엘리먼트를 사용합니다. 〈Applications〉 엘리먼트 내에 〈Application〉 엘리먼트를 사용하여 개별 어플리케이션을 정보를 설정하며, 각 어플리케이션에 대해서는 어플리케이션명, 디렉토리, 어플리케이션 정의 파일, 어플리케이션 정의 파일로 전달할 매개변수를 지정하기 위해, 〈ApplicationName〉, 〈BaseDirectoryPath〉, 〈ApplicationDefinitionFilePath〉, 〈Parameters〉 엘리먼트를 사용합니다. |
| 프로토콜 설정 | 〈Protocols〉 엘리먼트는 〈Protocol〉 엘리먼트의 컬렉션을 포함하기 위해서 사용됩니다. 각 〈Protocol〉 엘리먼트에는 사용자정의 프로토콜에 대한 정보가 포함됩니다. 표준 프로토콜(파일, HTTP, SMTP)을 사용하지 않는 경우에만 지정합니다. |
| 배달채널 | 인스턴스가 사용할 수 있는 배달채널은 〈DeliveryChannels〉 엘리먼트로 정의합니다. 각 배달채널은 〈DeliveryChannel〉 엘리먼트에 정의되며, 배달채널명, 사용할 프로토콜, 프로토콜에서 필요로 하는 인수 (SMTP의 경우, 서버명 등)가 정의에 포함됩니다. |
| 암호화 인수 | 인스턴스 데이터베이스와 어플리케이션 데이터베이스에 포함된 중요 데이터를 암호화하기 위해, 〈EncryptArguments〉 엘리먼트를 사용합니다. |

### ■ 어플리케이션 정의 파일(ADF)

Notification Service 솔루션에 포함된 각 어플리케이션에 대해 어플리케이션 정의 파일을 생성해야 합니다. 구성 파일과 함께, 필요한 요구조건을 충족하는 어플리케이션을 생성하기 위해서, 어플리케이션 정의 파일의 각 항목에 대해서 이해할 필요가 있습니다.

| 구성요소 | 설명 |
| --- | --- |
| 매개변수 | 구성정보 파일에 지정한 매개변수가 ADF 파일로 전달되며, 퍼센트(%)문자안에 매개변수명을 지정하여 참조할 수 있습니다. 구성정보 파일와 같이, 〈ParameterDefaults〉 엘리먼트를 사용하여 매개변수의 기본값을 설정할 수 있습니다. |
| 파일문서화 | 구성정보 파일와 같이, 〈Version〉 엘리먼트와 〈History〉 엘리먼트를 사용하여 ADF의 문서 정보를 지정할 수 있습니다. |
| 데이터베이스 설정 | 어플리케이션 데이터베이스에 대한 데이터베이스 파일의 정보를 커스터마이징해야 한다면, 〈Database〉 엘리먼트에 포함된 파일 정보를 설정할 수 있습니다. |
| 이벤트클래스 | 데이터베이스안에 이벤트 테이블과 뷰를 정의하기 위해서 〈EventClasses〉 엘리먼트에 〈EventClass〉 엘리먼트를 포함시킬 수 있습니다. 각 이벤트 크래스에는 다음과 같은 엘리먼트가 포함될 수 있습니다.<br>• 〈EventClassName〉<br>• 〈Schema〉<br>• 〈FileGroup〉<br>• 〈IndexSqlSchema〉<br>• 〈ChronicleRule〉<br>• 〈Chronicles〉 |
| 구독 클래스 | 〈SubscriptionClasses〉 엘리먼트에는 각 구독 테이블 및 뷰에 대한 정보를 포함하고 있는 〈SubscriptionClass〉 엘리먼트가 포함됩니다. 〈SubscriptionClass〉 엘리먼트에는 다음과 같은 엘리먼트가 포함될 수 있습니다.<br>• 〈SubscriptionClassName〉<br>• 〈Schema〉<br>• 〈FileGroup〉 |

| | |
|---|---|
| | • 〈IndexSqlSchema〉<br>• 〈EventRules〉<br>• 〈ScheduledRules〉<br>• 〈Chronicles〉 |
| 알림 클래스 | Notification 테이블과 뷰는 〈NotificationClass〉 엘리먼트의 컬렉션안에 정의되며, 〈NotificationClass〉 엘리먼트에는, 다음과 같은 엘리먼트가 포함될 수 있습니다.<br><br>• 〈NotificationClassName〉<br>• 〈Schema〉<br>• 〈FileGroup〉<br>• 〈ContentFormatter〉<br>• 〈DigestDelivery〉<br>• 〈MulticastDelivery〉<br>• 〈NotificationBatchSize〉<br>• 〈Protocols〉<br>• 〈ExpirationAge〉 |
| 공급자 설정 | 호스팅된 독립 이벤트 공급자 설정을 〈Providers〉 엘리먼트에 지정합니다. |
| 생성자 설정 | 생성자 역할을 수행할 서버 이름 등과 같은, 생성자 설정을 〈Generator〉 엘리먼트에 지정합니다. |
| 배포자 설정 | 배포자 관련 정보는 〈Distributor〉 엘리먼트에 지정합니다. |
| 어플리케이션 실행 설정 | 〈ApplicationExecutionSettings〉 엘리먼트를 사용하여, 퀀텀 기간, 기간이 경과된 데이터에 대한 보존기간 등과 같은 어플리케이션 실행 설정을 통제할 수 있습니다. 〈Application ExecutionSettings〉 엘리먼트를 지정하지 않으면, 시스템 기본값이 사용됩니다. |

## ■ SQL Server Management Studio 사용하여 Notification Service 어플리케이션 개발

구성정보 파일과 ADF 파일을 사용하여, 인스턴스와 어플리케이션에 대한 정의를 완료한 다음, 인스턴스를 생성하고, 등록하고, 활성화하고, 시작하는 절차를 수행해야 합니다. Notification Service 솔루션을 설정하는 각 단계를 SQL Server Management Studio를 사용하여 수행할 수 있습니다.

### 1 단계. 인스턴스 생성

Notification Service 인스턴스와 어플리케이션을 생성하는 작업을 수행해야 합니다. 인스턴스 생성 작업을 통해, Notification Service 솔루션을 위한 SQL Server 데이터베이스 생성됩니다. SQL Server Management Studio에서는 다음의 절차를 수행하여 인스턴스 생성작업을 수행할 수 있습니다.

1. 개체 탐색기에서, Notification Services 노드에서 오른쪽 클릭한 다음 새 Notification Services 인스턴스 생성을 클릭합니다.
2. 인스턴스에 대한 구성정보 파일을 지정합니다.
3. 구성정보 파일에 정의된 각 매개변수의 값을 할당합니다.
4. 인스턴스가 생성된 다음, 활성화될 수 있도록 설정할 수 있습니다.
5. 확인을 클릭하여 인스턴스를 생성합니다.
6. 인스턴스 생성작업이 완료되면, 닫기를 클릭합니다.

### 2 단계. 인스턴스 등록

인스턴스를 생성한 다음, 인스턴스를 등록하고, 인스턴스를 관리하기 위해서 사용할 Windows 서비스를 정의해야 합니다. SQL Server Management Studio에서는 다음의 절차를 수행하여 인스턴스 등록작업을 수행할 수 있습니다.

1. 개체 탐색기에서, 등록할 Notification Services 인스턴스를 오른쪽 클릭한 다음 작업 옵션에서 인스턴스 등록을 클릭합니다.
2. Windows 서비스에 대한 보안신임장을 지정한 다음, 확인을 클릭합니다.
3. 등록작업이 완료되면, 닫기를 클릭합니다.

3 단계. 인스턴스 활성화

Notification Services 인스턴스를 사용하기 전에, 먼저 반드시 활성화해야 합니다. SQL Server Management Studio에서는 다음의 절차를 수행하여 인스턴스 활성화작업을 수행할 수 있습니다.

1. 개체 탐색기에서, 활성화할 Notification Services 인스턴스를 오른쪽 클릭한 다음, 활성화 옵션을 클릭합니다.
2. 해당 인스턴스를 활성화할 것인지 여부를 묻는 창이 나타나면, 확인을 클릭합니다.

4 단계. 인스턴스 시작

인스턴스를 활성화한 다음, Windows 서비스에 연결된 인스턴스를 시작할 수 있습니다. SQL Server Management Studio에서는 다음의 절차를 수행하여 인스턴스를 시작할 수 있습니다.

1. 개체 탐색기에서 시작할 Notification Services 인스턴스를 오른쪽 클릭한 다음, 시작을 클릭합니다.
2. 해당 인스턴스를 시작할 것인지 여부를 묻는 창이 나타나면, 확인을 클릭합니다.
3. 인스턴스가 정상적으로 시작되면, 닫기를 클릭합니다.

## ■ NCControl 유틸리티 사용방법

1 단계. 인스턴스와 어플리케이션 생성

NSControl 유틸리티의 Create 명령을 사용하여, 인스턴스 데이터베이스와 어플리케이션 데이터베이스를 생성할 수 있습니다. NSControl 유틸리티의 Create 명령에 대한 구문은 다음과 같습니다.

```
nscontrol create
[-help] |
-in configuration_filename
[-sq|username “sql_login” -sq|password “password”]
```

```
[-argumentkey “cryptographic_key”]
[parameter_name=value [,..,n] ]
[-nologo]
```

NSControls 유틸리티의 Create 명령에 대한 매개변수는 다음과 같습니다.

- **help**: CREATE 명령에 대한 구문정보를 표시합니다.
- **in**: 인스턴스에 대한 구성 정보 파일의 경로를 지정합니다.
- **sqlusername**: SQL Server 인증을 사용하는 경우, SQL Server 로그인을 지정합니다.
- **sqlpassword**: SQL Server 로그인에 대한 비밀번호를 지정합니다.
- **argumentkey**: 암호화 인수에서 사용할 키를 지정합니다.(구성정보 파일내의 〈EncryptArguments〉 엘리먼트에서 사용).
- **parameter_name**: 구성정보파일로 전달할 여러 개의 매개변수 명칭과 값 쌍을 지정합니다.
- **nologo**: CREATE 명령이 실행된 다음, 제품 버전에 대한 요약정보를 표시하지 않기 위해 지정합니다.

2 단계. 인스턴스 등록

서비스를 등록하기 위해서는, 다음과 같은 구문으로, NSControl 유틸리티의 Register 명령을 실행합니다.

**[따라하기] 인스턴스 등록**

```
nscontrol register
[-help] |
-name instance_name
[server “sql_server_instance_name”]
[-service
```

```
[-serviceusername "account" -servicepassword "password"]
[-sqlusername "sql_login" -sqlpassword "password"]
[-argumentkey "cryptographic_key"]
[-nologo]
]
```

NSControls 유틸리티의 Register 명령에 대한 매개변수는 다음과 같습니다.

- **help**: Register 명령에 대한 구문정보를 표시합니다.
- **name**: 등록할 인스턴스 이름
- **server**: 데이터베이스를 생성할 SQL Server 인스턴스
- **service**: Windows 서비스가 생성되어야 한다는 것을 지정합니다.
- **serviceusername**: 서비스에 대한 Windows 사용자 계정
- **servicepassword**: 서비스 계정에 대한 비밀번호
- **sqlusername**: SQL Server 인증을 사용하는 경우, 서비스에서 사용할 SQL Server 로그인
- **sqlpassword**: SQL Server 로그인에 대한 비밀번호.
- **argumentkey**: 암호화 인수에서 사용할 키 (구성정보 파일의 〈EncryptArguments〉 엘리먼트에서 사용).
- **nologo**: 명령이 실행된 다음 제품 정보에 대한 정보를 표시하지 않기 위해서 지정합니다.

**3 단계.인스턴스 활성화**

기본 값으로 새로 추가된 인스턴스와 어플리케이션은 비활성화된 상태입니다. 인스턴스에 포함된 구성요소를 활성화하기 위해서는, NSControl 유틸리티의 Enable 명령을 사용해야 합니다. Enable 명령에 대한 구문은 다음과 같습니다.

```
nscontrol enable
[-help] |
-name instance_name
[-component [...n]]
[-server "database_server_name"]
[-application "application_name"]
[-sqlusername "sql_login" -sqlpassword "password"]
[-nologo]
```

NSControls 유틸리티의 Enable 명령에 대한 매개변수는 다음과 같습니다.

- **help**: Enable 명령에 대한 구문정보를 표시합니다.
- **name**: 활성화할 인스턴스의 이름
- **component**: 활성화할 컴포넌트의 이름. 컴포넌트에는 배포자, 이벤트, 생성자, 구독, 구독자 등이 포함될 수 있습니다.
- **server**: SQL Server 인스턴스 이름
- **application**: 활성화할 특정 어플리케이션의 이름
- **sqlusername**: SQL Server 인증을 사용하는 경우, 사용할 SQL Server 로그인
- **sqlpassword**: SQL Server 로그인의 비밀번호
- **nologo**: 명령이 실행된 다음, 제품 버전 정보를 표시하지 않기 위해 지정합니다.

**4 단계.인스턴스 서비스 통제**

인스턴스에 대한 Windows 서비스는 다른 Windows 서비스와 동일한 방법으로 관리됩니다. 서비스 MMC 스냅인을 사용하여, 인스턴스 서비스를 시작하거나 중지할 수 있고, Windows가 시작할 때 자동으로 시작할 수 있도록 구성할 수 있으며, net start 명령이나 net stop 명령을 사용하여 인스턴스 서비스를 통제할 수 있습니다.

## ■ 구독 관리 어플리케이션

사용자가 알림을 구독할 수 있도록 하기 위해, 구독 관리 어플리케이션을 구현해야 합니다.

### 구독 관리 개체

구독 관리 어플리케이션을 개발하기 위해서, 구독 관리 개체 API를 사용합니다. 구독 관리 개체 API 안에 포함된 관리되는 클래스를 사용하여, 구독자 데이터를 Notification Service에 전달하기 위한 .NET 기반 구독 관리 어플리케이션을 개발할 수 있습니다. 또한, COM 상호 운영성 래퍼 클래스를 사용하여, COM 기반 코드로 구독 관리 어플리케이션을 개발할 수 있습니다. 구독 관리 개체 API에는 다음과 같은 Microsoft.SqlServer.NotificationServices 네임스페이스 내의 클래스가 포함됩니다.

- **NSInstance**: Notification Services 인스턴스를 표현.
- **NSApplication**: Notification Services 어플리케이션을 표현.
- **Subscriber**: 특정 인스턴스의 구독자를 표현.
- **SubscriberDevice**: 지정된 인스턴스의 구독자 디바이스를 표현
- **Subscription**: 지정된 어플리케이션내의 구독을 표현

### NSInstance 클래스

Notification Service 인스턴스에 연결하기 위해서, NSInstance 클래스를 사용합니다. NSInstance 클래스 생성자에 인스턴스명을 전달하면 인스턴스를 생성할 수 있습니다. 또는 인스턴스 명을 지정하고, Initialize 메서드를 호출하여 인스턴스를 생성할 수 있습니다.
다음 예제는 Visual Basic.NET 코드 형식으로 생성자에 인스턴스명을 전달하는 방법을 나타냅니다.

```
Dim inst As New NSInstance("AWInstance")
```

또는 다음과 같이, 인스턴스명을 지정하여 initialize 메서드를 호출할 수도 있습니다.

```
Dim inst As New NSInstance
inst.Initialize ("AWInstance")
```

NSApplication 클래스

특정 어플리케이션(예를 들어, 구독 생성)에 연결하기 위해서, NSApplication 클래스를 사용합니다. NSApplication 클래스를 사용하기 위해서는, 먼저 해당 어플리케이션을 호스팅하고 있는 인스턴스를 표현하는 NSInstance 개체와 어플리케이션명을 전달해야 합니다. NSInstance 클래스와 동일하게 인스턴스 개체와 어플리케이션명을 지정하고, Initialize 메서드를 호출하여 NSApplication 클래스의 인스턴스를 생성할 수도 있습니다.

다음 예제는 NSApplication 클래스의 인스턴스를 생성하는 방법을 나타냅니다.

[따라하기] NSApplication 생성

```
Dim app As New NSApplication(inst, "ProductApp")
```

Subscriber 클래스

인스턴스 데이터베이스에 새 구독자를 추가하기 위해서, Subscriber 클래스를 사용합니다. 구독자를 추가하기를 원하는 인스턴스를 지정하여, Subscriber 클래스에 대한 인스턴스를 생성한 다음(생성자를 호출하거나, Initialize 메서드를 사용), SubscriberId 속성(데이터베이스 내에서 해당 구독자를 식별할 수 있도록 유일한 값을 지정)을 설정한 다음, Add 메서드를 호출합니다.

[따라하기] 구독자 생성

```
Dim Subr As New Subscriber(inst)
Subr.SubscriberId = "Bill"
Subr.Add()
```

Subscriber Device 클래스

SubscriberDevice 클래스는 인스턴스 데이터베이스에 새 구독자 디바이스를 추가하기 위해서 사용합니다. 구독자 디바이스를 추가하기를 원하는 인스턴스와 SubscriberId 속성을 유효한 구독자로 지정하여, SubscriberDevice 클래스의 인스턴스를 생성합니다.
그 다음, SubscriberDevice개체에 DeviceTypeName, DeviceAddress, DeviceName, DeliveryChannelName 속성 정보를 설정합니다.

### [따라하기] 구독자 디바이스 생성

```
Dim SubDev As New SubscriberDevice(inst)
SubDev.SubscriberId = "Bill"
SubDev.DeviceTypeName = "SMTP"
SubDev.DeviceAddress = "Bill@adventure-works.com"
SubDev.DeviceName = "EmailDevice"
SubDev.DeliveryChannelName = "SMTPChannel"
SubDev.Add( )
```

Subscription 클래스

어플리케이션 데이터베이스에 구독을 추가하기 위해, Subscription 클래스를 사용합니다. Subscription 클래스의 인스턴스를 생성하기 위해, 구독을 추가하기를 원하는 어플리케이션을 표현하기 위한 NSApplication 개체와, 구독명(어플리케이션내의 구독 뷰의 이름과 일치되어야 함)을 지정해야 합니다. 그 다음, SubscriberId 속성을 설정하고, SetFieldValue 메서드를 사용하여, DeviceNaem 필드와 SubscriberLocale 필드, 사용자정의 필드의 값을 설정합니다.

[따라하기] 구독 생성

```
Dim Subn As New Subscription(app, "NewProductSub")
Subn.SubscriberId = "Bill"
Subn.SetFieldValue("DeviceName", "EmailDevice")
Subn.SetFieldValue("SubscriberLocale", "en-us")
Subn.SetFieldValue("CategoryToMonitor", 1)
Subn.Add()
```

## ■ 이벤트 공급자

이벤트 공급자는 어플리케이션 데이터베이스에 이벤트를 전달합니다. 비즈니스 요구사항을 충족시키는 솔루션을 개발하기 위해서 이벤트 공급자를 구현하기 위한 다양한 옵션에 대해 알아두어야 합니다.

| 항목 | 설명 |
|---|---|
| 표준 이벤트 공급자 | Notification Service에는 두 가지 표준 이벤트 공급자가 포함되어 있습니다.<br>• FileSystemWatcher 이벤트 공급자는 파일 시스템 위치를 모니터링하며, 지정된 XML 파일로부터 이벤트 정보를 적재합니다.<br>• SQL Server 이벤트 공급자는 주기적으로 SQL Server 데이터베이스내의 테이블을 조회하여, 어플리케이션 데이터베이스내의 이벤트 테이블에 데이터를 저장합니다. |

| | |
|---|---|
| 사용자정의<br>이벤트 공급자 | Notification Service에서 제공하는 네 가지 이벤트 API를 중 하나를 사용하여 사용자정의 이벤트 공급자를 생성할 수 있습니다.<br>• 관리되는 이벤트 API. 관리되는 이벤트 API에서는 이벤트 정보를 전달하기 위한 .NET 클래스를 제공합니다.<br>• COM 이벤트 API. COM 이벤트 API에서는 관리되는 클래스와 COM 코드의 상호운영성을 보장하기 위한 래퍼 클래스를 제공합니다.<br>• XML 이벤트 API. XML 이벤트 API에서는 어플리케이션에 대한 이벤트 데이터를 포함하고 있는 XML 문서를 전달하기 위한 EventLoader 라는 관리되는 클래스를 제공합니다. XML 이벤트 API에는 데이터베이스에 존재하는 이벤트 테이블과 XML 문서의 엘리먼트를 매핑하기 위한 XML 스키마가 제공되어야 합니다.<br>• SQL Server 이벤트 API. SQL Server 이벤트 API는 이벤트 정보를 배치 작업으로 전달하기 위한 저장 프로시저로 구성됩니다. |
| 호스팅된 이벤트<br>공급자와<br>독립 이벤트 공급자 | 이벤트 공급자는 IEventProvider 인터페이스나 IScheduledEventProvider 인터페이스를 사용하여 Notification Service 프로세스에 호스팅될 수 있으며, ADF 파일 내에 〈HostedProvider〉 엘리먼트내에 해당 이벤트 공급자를 지정합니다. 호스팅된 이벤트 공급자는 Notification Service와 함께 시작되고 중단되며, Notification Service의 스케줄러를 사용할 수 있습니다. 독립 이벤트 공급자는 Notification Service 프로세스와 독립적으로 실행되며, ADF 파일내에 〈NonHostedProvice〉 엘리먼트내에 해당 이벤트 공급를 지정합니다. 독립 이벤트 공급자는 Notification Service와 별도로 통제할 수 있습니다. |

# 웹서비스와 네이티브 HTTP 지원

서비스 지향 기술구조(SOA)를 기반으로, SQL Server 2005에서는 네이티브 HTTP를 사용하여 직접 SOAP 기반으로 웹서비스를 게시하는 기능을 제공합니다. 웹 서비스의 개념과 SQL Server 2005를 통해 데이터베이스 어플리케이션에서 네이티브 HTTP 기능을 사용하는 방법에 대해서 살펴 봅니다. SQL Server 2005의 네이티브 HTTP 지원기능을 통해 산업표준 통신 메커니즘에 근거하여 최소한의 노력을 투자하여 데이터베이스와 통신할 수 있도록 설정할 수 있습니다.

## 웹서비스와 SOAP이란?

웹 서비스와 SOAP은 분산 어플리케이션 개발 모델의 단점을 보완하는 웹 표준입니다.

### ■ 웹 서비스

웹 서비스는 인터넷 또는 인트라넷을 통해 접근이 가능한 프로그래밍 가능한 로직을 제공하며, 분산 어플리케이션을 개발하기 위한 빌딩블럭의 역할을 수행합니다. 웹 서비스는 블랙박스의 개념으로, 내부에 구현이 캡슐화되며, 웹서비스 간에 통신을 위한 인터페이스를 제공합니다.

### ■ 웹서비스를 사용하는 장점

- 상호운영성. 웹 서비스는 HTTP와 XML을 사용하여 통신하기 때문에, HTTP와 XML을 지원하는 모든 네트워크 노드에서 웹 서비스를 호스팅할 수 있고, 웹 서비스를 사용할 수 있습니다.
- 다중 언어 지원. 개발자는 어떤 개발언어로도 웹 서비스를 개발할 수 있습니다.
- 기존 어플리케이션 재사용. 기존 컴포넌트와 라이브러리를 간단하게 웹서비스로 제공할 수 있습니다. 기존 회사에 보유하고 있는 기존 컴포넌트, 라이브러리, 어플리

케이션은 상당히 많습니다. 이러한 기존 자원을 다시 구현할 필요없이 소프트웨어 자원의 기능을 재사용할 수 있기 때문에 비용을 절약할 수 있습니다.

- 산업표준사용. HTTP, XML, SOAP과 같은, 웹서비스 관련 기술은 거의 모든 벤더에서 지원하고 있습니다. 산업표준을 사용함으로써 이기종 시스템간에 통신을 좀 더 쉽게 처리할 수 있습니다.

### ■ SOAP

SOAP은 분산환경에서 분산되어 있는 정보를 교환하기 위한 프로토콜입니다. SOAP 메시지를 교환하는 방법을 통해, 웹 서비스간에 통신을 처리할 수 있습니다.

SOAP 메시지는 헤더와 본문으로 구성된 XML 문서입니다. 기본적으로 SOAP 메시지는 송신자(sender)에서 수신자(receiver)에게 보내는 단방향 전문입니다. SOAP에는 프로그램 모델, 구현상세정보 등과 같은 어플리케이션 정보에 대해서는 정의하지 않습니다 그러므로, 웹 서비스는 HTTP 요청 및 응답 체계안에 SOAP 메시지를 보내는 요청/응답 모델을 사용합니다.

### ■ 웹 서비스 예제

웹서비스를 기반으로 웹 사이트의 기능을 제공할 수 있습니다. 여행사 웹 사이트 시나리오에서, 사용자가 웹 페이지에 여행지에 대한 도시명을 입력합니다. 해당 페이지에서는 지정된 도시명을 매개변수로 SOAP을 사용하여 웹 서비스를 호출합니다. 웹 서비스에서는 여행지 도시에 대한 기상정보, 환율 등과 같은, 해당 페이지에 필요한 정보를, SOAP 메시지형태로 반환합니다. 웹 페이지에서는 해당 반환값을 필요한 형식으로 전환하여 사용자에게 보여주면 됩니다.

사용자 관점에서는, 필요한 정보를 해당 웹 사이트에서 충분하게 제공받을 수 있다는 장점이 있습니다. 코드를 작성하는 개발자 관점에서는, 웹 사이트를 개발하기 위해 다른 업체에서 제공하는 웹 서비스를 조합하는 적은 노력만을 투자하고, 웹 사이트의 그래픽 인터페이스에 대해서 좀 더 많은 투자를 하여 사용자에게 좀 더 양질의 웹사이트를 제공할 수 있습니다.

## 네이티브 HTTP 지원

SQL Server 2005에서는 네이티브 HTTP 기능을 지원하기 때문에, 웹서비스를 사용하여 프로그래밍 기반으로 인터넷이나 내부 방화벽을 통과하여 데이터베이스에 작업을 수행할 수 있습니다. 네이티브 HTTP 지원기능을 통해, 개발자는 인터넷을 통한 데이터 처리 작업을 개발하기 위한 노력을 절감할 수 있습니다.

### ■ 데이터베이스 개체 노출

특정 데이터베이스 개체를 선택하여 웹서비스로 노출시킬 수 있습니다. 클라이언트 어플리케이션에서는 웹서비스로 노출된 개체에 대해서만 접근이 가능합니다. 다음과 같은 개체를 웹서비스를 노출시킬 수 있습니다.

- 저장 프로시저. 사용자 저장 프로시저(.NET CLR 기반으로 작성한 저장 프로시저 포함)나 확장 저장 프로시저를 웹 서비스로 노출시킬 수 있습니다. 저장 프로시저는 매개변수를 전달받아 결과값을 반환할 수도 있고, 결과정보의 반환없이 특정한 기능을 수행하도록 처리할 수도 있습니다.
- 사용자정의함수. 스칼라 값을 반환하는 사용자정의함수를 웹서비스로 노출시킬 수 있으며, 사용자정의함수에서는 매개변수를 전달받을 수 있습니다.
- T-SQL 배치문장. 임의 T-SQL 배치문장을 웹서비스의 특정 메서드로 노출시킬 수 있습니다. SQL Server 에서는 클라이언트 어플리케이션에 해당 임의 쿼리 문장을 실행할 사용권한만 부여하게 됩니다.

또한, HTTP 엔드포인트는 Service Broker 서비스에 대한 원적 접근을 위해서도 사용됩니다.

### ■ 보안

SQL Server 에서는 웹 서비스로 노출된 저장 프로시저, 사용자정의 함수, T-SQL 배치 문장을 실행하기 전에, 해당 클라이언트 어플리케이션에 대한 인증 신임장을 체크하여 보안 사용권한이 부여되었는지를 체크합니다. SQL Server 내에서 웹 서비스 보안에 대한 관리를 수행할 수 있습니다.

## 네이티브 HTTP를 사용하는 이유

### ■ 이기종 데이터 액세스

비즈니스 환경에서는 동일한 데이터에 서로 다른 다수의 클라이언트 어플리케이션이 접근 상황이 자주 발생합니다. 각 어플리케이션은 Windows Form 기반, Web 기반, 배치 처리를 위한 콘솔 기반 등 여러 가지 형태로 개발됩니다. 또한 각 어플리케이션의 개발 언어도 서로 다를 수 있으며, 데이터에 접근하기 위한 매커니즘도, OLE DB, ODBC, 사용자정의 데이터 접근 매커니즘 등 다양할 수 있습니다.

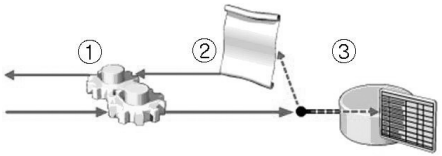
이러한 상황에서 데이터에 접근하기 위한 방법으로 웹 서비스를 사용할 수 잇습니다. 웹 서비스는 프로그래밍 언어나 운영체제 시스템과 독립적이기 때문에, XML 및 HTTP 네트워크 연결을 지원하기만 하면, 어떤 클라이언트 어플리케이션에서도 작업을 수행할 수 있습니다. 모바일 장비부터, 메인 프레임까지, Java, Visual Basic.NET, 또는 다른 프로그래밍 언어로 개발된 모든 어플리케이션에 웹서비스가 사용될 수 있습니다.

### ■ 방화벽 포트 오픈 최소화

웹 어플리케이션에서 SQL Server 로 접근해야 하는 경우, 일반적으로, 관리자가 웹서버에서 데이터베이스로 접근할 수 있도록 하기 위해 내부 방화벽에 추가 포트를 오픈하게 됩니다. 웹 서비스를 사용하여 데이터베이스 접근하면, 표준 HTTP 통신을 허용하기 위한 내부 방화벽에 대한 포트만 오픈하면 됩니다.

## 네이티브 HTTP 기술구조

SQL Server 2005에서 지원하는 네이티브 HTTP 기능을 사용하기 위해서는 먼저 기초적인 기술구조에 대해서 이해해야 합니다. 기술구조를 이해함으으로써, 설치에 대한 요구조건을 확인할 수 있고, 클라이언트와 통신하기 위한 다양한 개체의 역할에 대해서도 이해할 수 있습니다.




1. http.sys 에서 HTTP 요청을 처리합니다. 인터넷정보서비스(IIS)는 필요하지 않습니다.
2. SQL서버가 웹 서비스 메서드를 표현하기 위한 WSDL을 생성합니다.
3. HTTP 엔드포인트가 데이터베이스 개체를 웹서비스의 메서드로 제공합니다.

*[네이티브 HTTP 기술구조]*

## 네이티브 HTTP 지원 구성

네이티브 HTTP 지원 기능을 구성하기 위해서, 반드시 생성해야 하는 SQL Server 개체가 추가되었습니다. 운영 환경에서는 HTTP 엔드포인트를 생성하고, 반드시 보안을 보장할 수 있도록 관리해야 합니다. 또한 URL 네임스페이스를 예약해야 합니다.

### ■ HTTP 엔드포인트 생성

웹 서비스로 SQL Server 2005 에 접근하기 위해서는, 웹 서비스 정보를 지정하기 위한 비스 정보를 지정하기 위한 HTTP 엔드포인트를 생성해야 합니다.

HTTP 엔드포인트를 생성하기 위해서는, 다음과 같은 단계에 따라, CREATE ENDPOINT 명령을 수행하면 됩니다.

**1. 전송 프로토콜 정보 지정**

CREATE ENDPOINT의 처음 부분에서 전송 프로토콜 정보를 지정합니다. 전송 프로토콜 정보에는 수신 포트, 인증 방법 등에 대한 정보가 포함됩니다.

다음의 표는 CREATE ENDPOINT 명령에서 지정할 수 있는 전송 프로토콜 관련 매개변수의 목록입니다.

| 매개변수 | 설명 |
|---|---|
| endPointName | HTTP 엔드포인트를 수정하거나 삭제할 때 사용하기 위한 엔드포인트 명 |
| AUTHORIZATION | 엔드포인트에 대한 소유권할당. 로그온을 지정하지 않으면, 문장을 실행하는 사용자가 소유자가 됩니다. |
| STATE | 엔드포인트를 생성하자마자 요청을 수신할 것인지 여부를 지정합니다. 기본값은 엔드포인트를 생성한 다음 바로 요청을 수신하지 않는 것입니다. |
| AS | 사용할 전송 프로토콜을 지정. 기본값은 HTTP 프로토콜. |
| FOR | 페이로드(payload) 유형을 지정합니다. |
| PATH | SITE 매개변수로 지정된 호스트 컴퓨터에서 엔드포인트의 위치를 지정하기 위한 URL 경로 |
| PORTS | 엔드포인트에 표준 포트로 수신할 것인지, SSL 포트로 수신할 것인지, 양쪽 포트 모두로 수신할 것인지를 지정합니다. |
| SITE | 수신할 호스트 컴퓨터의 이름을 지정. (+) 부호를 사용하여, 컴퓨터에 대한 가능한 모든 호스트명에 대해 수신하도록 지정할 수 있습니다. 또는, (*) 부호를 사용하여, 현재 예약되지 않는 컴퓨터에 대한 모든 가능한 호스트명에 대해서 수신하도록 지정할 수 있습니다. 기본값은 (*) 부호입니다. |
| CLEAR_PORT | 표준 포트 통신을 위한 포트번호를 지정.(기본값은 80) |
| SSL_PORT | SSL 통신을 위한 포트번호를 지정.(기본값은 433). SSL에 필요한 디지털 인증서는 sp_register_ssl certificate_for_http 저장 프로시저를 사용하여, "내(My)" 인증서 저장소에 등록할 수 있습니다. |

| | |
|---|---|
| AUTHENTICATION | 클라이언트 어플리케이션에서 웹 서비스를 호출할 때 사용할 인증방식을 지정합니다. ANONYMOUS, BASIC, DIGEST, INTEGRATED 로 정의할 수 있습니다. |
| AUTH_REALM / DEFAULT _LOGON_DOMAIN | 인증 처리절차에 대한 기본 정보를 지정합니다. DIGEST 인증인 경우 AUTH_REALM를 사용하고, BASIC 인증은 DEFAULT_LOGON_DOMAIN를 사용합니다. |
| RESTRICT_IP and EXCEPT_IP | HTTP 엔드포인트에 접속할 수 있는 IP 목록이나 접속제한할 IP 목록을 지정합니다.<br>기본값은 IP 주소를 제한하지 않는 것입니다. |
| COMPRESSION | HTTP 요청에 GZIP 인코딩 옵션이 포함된 경우, 클라이언트에 응답할 때 압축기능을 사용하도록 지정합니다. 기본값은 DISABLED 입니다. |

**2. 접근가능한 웹 메서드 목록을 포함한 엔드포인트 정보 지정**

CREATE ENDPOINT 명령의 두번째 부분은 엔드포인트 페이로드 정보를 지정하는 부분입니다. 설정하는 정보는 설정된 엔드포인트가 SOAP 기반 웹서비스이냐 Service Broker 기반 네트워크 통신이냐에 따라 다양하게 설정됩니다.

다음은 SOAP 기반 웹서비스 페이로드 정보를 설정하기 위한 매개변수 목록입니다.

| 매개변수 | 설명 |
|---|---|
| WEBMETHOD | SOAP 메시지를 사용하여 클라이언트에서 접속할 수 있는 서버 측 메서드 목록을 정의합니다. WITH 절에 NAMESPACE 인수를 사용하거나, 웹 메서드를 정의의 일부로 네임스페이스 값을 지정할 수 있습니다. MethodAlias 값은 클라이언트에서 웹서비스의 웹메서드에 요청할 때 사용할 수 있습니다. |
| NAME | 웹 메서드의 기능을 구현할 저장 프로시저나 사용자정의 함수의를 지정합니다. NAME 매개변수는 반드시 schema.name 형식으로 지정해야 합니다. |

| | |
|---|---|
| SCHEMA | SOAP 응답내에 웹메서드의 반환값에 대한 인라인 XSD 스키마를 반환할 것인지 여부를 지정합니다.<br>특정 값을 지정하지 않거나, DEFAULT로 지정한 경우에는, WITH 절에 SCHEMA 인수에 설정된 값으로 지정됩니다. |
| FORMAT | 클라이언트에 대한 응답에 포함할 정보를 지정하기 위해 사용합니다. ALL_RESULTS(기본값)은 결과집합, 행 수, 오류 메시지, 경고 등을 반환합니다. ROWSETS_ONLY로 지정하면, 결과 집합만 반환됩니다. |
| LOGIN_TYPE | 엔드포인트에서 SQL Server 인증 모드를 사용하도록 지정할 수 있습니다. 기본값은 WINDOWS 입니다. |
| BATCHES | 웹서비스에서 sqlbatch 메서드를 사용하여 임의T-SQL 쿼리를 실행하도록 지원할 것인지 여부를 지정합니다. 기본값은 Disabled 입니다. |
| WSDL | 엔드포인트에서 WSDL 문서를 자동으로 생성할 것인지를 지정합니다. 기본값은 Disabled입니다. |
| SESSIONS | ENABLED로 설정된 경우 SQL Server는 세션 지원을 허용하여 여러 SOAP 요청/응답 메시지 쌍을 단일 SOAP 세션의 일부로 식별할 수 있습니다. 기본값은 DISABLED입니다. |
| SESSION_TIMEOUT | 더 이상의 요청이 없을 경우 세션이 서버에서 만료되기까지의 대기기간 시간(초)를 지정합니다. |
| DATABASE | 요청한 작업 실행을 위한 대상 데이터베이스를 지정합니다. 데이터베이스 명이 지정되지 않았거나 DEFAULT가 지정된 경우 로그인의 기본 데이터베이스가 사용됩니다. |
| NAMESPACE | 엔드포인트에 대한 네임스페이스를 지정합니다. 네임스페이스가 지정되지 않았거나 DEFAULT로 지정된 경우 http://tempuri.org 네임스페이스가 사용됩니다. |

| | |
|---|---|
| SCHEMA | SOAP 결과를 전송할 때 엔드포인트에서 XSD 스키마를 반환할지 여부를 지정합니다. STANDARD가 기본값이며, 스키마가 기본값으로 반환된다는 의미입니다. |
| CHARACTER_SET | 작업의 결과가 유효하지 않은 XML 형식의 문자를 포함할 때 동작을 정의합니다. |

다음 표는 Service Broker 네트워크 통신 페이로드를 지정하기 위한 매개변수 목록입니다.

| 매개변수 | 설명 |
|---|---|
| AUTHENTICATION | Service Broker 전송 인증모드를 통제. |
| MESSAGE_ FORWARDING | 메시지 포워딩을 활성화하기 위해서 사용 |
| MESSAGE_ FORWARD_SIZE | 포워딩하는 메시지의 최대 저장소 크기(MB 단위)를 지정 |

### ■ 전송 프로토콜 정보 지정

전송 프로토콜 정보를 지정하기 위해서는 다음과 같은 구문을 사용합니다.

```
CREATE ENDPOINT endPointName [AUTHORIZATION login]
STATE = { STARTED | STOPPED | DISABLED }
AS  { TCP | HTTP } (
  PATH =  'url'
 , PORTS = ({CLEAR | SSL} [,... n])
[ SITE = { '*'  |  '+'  |  'webSite'  },]
  [, CLEAR_PORT = clearPort ]
```

```
  [, SSL_PORT = SSLPort ]
        , AUTHENTICATION =({BASIC | DIGEST | INTEGRATED} [,...n])
  [, AUTH_REALM = { 'realm' | NONE } ]
  [, DEFAULT_LOGON_DOMAIN = { 'domain' | NONE } ]
[, RESTRICT_IP = { NONE | ALL } ]
  [, COMPRESSION = { ENABLED | DISABLED } ]
  [,EXCEPT_IP = ({ 〈4-part-ip〉 | 〈4-part-ip〉:〈mask〉 } [,...n])
)
```

### ■ 엔드포인트 페이로드 정보 지정

앞의 전송프로토콜정보 지정 구문 다음에, 다음과 같은 구문을 사용하여, SOAP 기반 웹 서비스 유형의 엔드포인트 페이로드 정보를 지정할 수 있습니다.

```
[ FOR { SOAP     (
    [ { WEBMETHOD [ 'namespace' ,] 'methodalias' (
            NAME = three,part,name
            [, SCHEMA = { NONE | STANDARD | DEFAULT }]
            [, FORMAT = { ALL_RESULTS | ROWSETS_ONLY }]
            [, LOGIN_TYPE = { MIXED | WINDOWS } ] )
        } [,...n] ]
        [,  BATCHES = { ENABLED | DISABLED } ]
        [, WSDL = { NONE | DEFAULT | sp_name } ]

        [ , SESSIONS = { ENABLED | DISABLED } ]
        [ , SESSION_TIMEOUT = int ]
        [, DATABASE = { 'database_name' | DEFAULT } ]
        [, NAMESPACE = { 'namespace' | DEFAULT } ]
```

```
    [, SCHEMA = { NONE | STANDARD } ]
    [, CHARACTER_SET = { SQL | XML }]
```

다음은 Service Broker 엔드포인트를 위한 정보를 지정하기 위한 구문입니다.

```
FOR SERVICE_BROKER (
  [ AUTHENTICATION = ENABLED | REQUIRED* | NONE ]
  [, MESSAGE_FORWARDING = ENABLED | DISABLED*]
  [, MESSAGE_FORWARD_SIZE = forwardsize
)
```

다음은 통합인증 모드를 사용하여 두 개의 웹 메서드를 포함하고 있는 엔드포인트를 생성하는 방법을 나타냅니다.

### [따라하기] 엔드포인트 생성

```
CREATE ENDPOINT sql_AdventureWorks
STATE = STARTED
AS HTTP(
  PATH =  '/sql/AdventureWorks' , AUTHENTICATION = (INTEGRATED),
  PORTS = (CLEAR) )
FOR SOAP (
  WEBMETHOD  'GetProducts'
    (name=' AdventureWorks.Production.GetProducts' ,
     schema=STANDARD),
  WEBMETHOD  'GetProductsNoSchema'
    (name=  'AdventureWorks.Production.GetProducts' ,
     schema=NONE),
```

```
WSDL = DEFAULT,
BATCHES = ENABLED,
DATABASE = 'AdventureWorks',
NAMESPACE = 'http://AdventureWorks/' )
```

***[참고]***
*BATCHES 인수가 활성화되어 있기 때문에, 웹 서비스에는 세번째 메서드가 포함되어 있다고 할 수 있고, WSDL에는 sqlbatch 메서드가 포함됩니다.*
*WSDL 문서를 조회하기 위해서는 브라우저에서*
*http://machinename/sql/AdventureWorks?WSDL을 검색합니다.*

## ■ HTTP 엔드포인트 보안설정

웹 서비스는 항상 적절한 권한이 부여된 사용자만 사용하도록 보안이 유지되어야 합니다. HTTP 엔드포인트에 보안을 설정하기 위해서는, 다음과 같은 작업을 수행합니다.

### 1. 데이터베이스에 접근할 수 있는 사용자 계정이나 역할 생성

다음 예제는 SQL Server 에 Windows 사용자 로그인 계정을 추가하고, 기본 데이터베이스를 지정하고, 기본 데이터베이스에 새로 추가한 로그인 사용자에 대한 데이터베이스 사용자를 추가하는 방법을 나타냅니다.

### [따라하기] 로그인 및 사용자 생성

```
USE master
CREATE LOGIN [Adventure-Works\Peter] FROM WINDOWS
WITH DEFAULT_DATABASE = AdventureWorks
GO

USE AdventureWorks
```

```
CREATE USER Peter FOR LOGIN [Adventure-Works\Peter]
```

2. 웹 서비스로 사용할 저장 프로시저나 사용자 정의 함수에 접근할 수 있도록 적절한 사용 권한을 부여합니다.

다음 예제는 GetContact 저장 프로시저를 실행할 수 있는 사용권한을 부여하는 방법을 나타냅니다.

**[따라하기] 사용권한 부여**

```
USE AdventureWorks
GRANT EXECUTE ON Production.GetContact TO [Peter]
```

3. HTTP 엔드포인트에 연결할 수 있는 권한을 사용자와 역할에 부여합니다.

HTTP 엔드포인트에 연결할 수 있는 사용권한을 사용자에게 부여해야 합니다. Master 데이터베이스에서 GRANT CONNECT ON ENDPOINT 명령을 사용하여, 사용자에게 웹 서비스에 연결할 수 있는 권한을 부여할 수 있습니다.

다음 예제는 HTTP 엔드포인트에 연결하기 위한 사용자 사용권한을 설정하는 방법을 나타냅니다.

**[따라하기] 엔드 포인트 연결 권한 부여**

```
USE master
GRANT CONNECT ON ENDPOINT:: sql_AdventureWorks TO [Adventure-Works\Peter]
```

## ■ URL 네임스페이스 예약

SOAP 메시지를 수신하기 위해서는 Windows HTTP API(HTTP.sys)에 네임스페이스를 예약해야 합니다. 네임스페이스는 암시적으로 예약하는 방법과 명시적으로 예약하는 방법이 있습니다.

**암시적인 예약**

HTTP 엔드포인트가 생성될 때, SQL Server 가 운영중이라면, SQL Server 에서는 자동적으로 네임스페이스를 예약합니다. SQL Server 가 운영 중이 아닌 상태에서는, 다른 어플리케이션에서 해당 네임스페이스를 가져갈 수 있습니다. HTTP.sys내에 네임스페이스를 예약하려고 시도하는 경우, 로컬 Windows 관리자 권한을 가지고 있어야 합니다. 암시적 네임스페이스 예약을 사용하는 경우, CREATE ENDPOINT 권한을 가진 사용자는 반드시 로컬 Windows 관리자 권한을 가지고 있어야 합니다.

다음의 T-SQL 문장의 일부는 HTTP 엔드포인트를 생성하고, 임시 네임스페이스를 예약하는 역할을 합니다. SQL Server 가 운영중인 상태라면,
http://MyServer:80/sql/AdventureWorks 엔드포인트에 대한 모든 HTTP 요청을, HTTP API가 SQL Server 로 전달하게 됩니다.

### [따라하기] 암시적 네임스페이스 예약

```
CREATE ENDPOINT sql_endpoint
    AS HTTP( SITE = 'MyServer' ,
            PATH = '/sql/AdventureWorks'
            ...)
...
```

**명시적인 예약**

sp_reserve_http_namespace 저장 프로시저를 사용하여, HTTP.sys 안에 명시적으로 네임스페이스를 예약할 수 있습니다. 명시적으로 네임스페이스를 예약하면, 다른 어플리케이션에서 해당 네임스페이스를 다시 사용할 수 없습니다. 암시적 네임스페이스 예약인 경우, Windows 관리자 권한을 가지고 있어야 합니다. 하지만, 명시적으로 네임스페이스 예약을 설정하는 경우에는, CREATE ENDPOINT 명령을 실행하는 사용자에게 로컬 Windows 관리자 권한을 부여할 필요가 없습니다.

다음 예제는 명시적인 방법으로 네임스페이스를 예약하는 방법을 나타냅니다. HTTP API는 http://MyServer:80/sql 엔드포인트에 대한 HTTP 요청을 SQL Server 로 전달합니다.

**[따라하기] 명시적 네임스페이스 예약**

```
sp_reserve_http_namespace N' http://MyServer:80/sql'
```

## .NET 기반 HTTP 엔드포인트 클라이언트 개발

가장 강력하면서도 간단하게 구현할 수 있는 SQL Server HTTP 엔드포인트 활용 방법은 웹서비스를 호출하는 .NET 클라이언트 어플리케이션을 사용하는 것입니다. .NET 클라이언트 어플리케이션을 개발하기 위해서는, 몇 가지 단계의 작업을 수행해야 하며, 일부 작은 양의 코드를 작성해야 합니다.

HTTP 엔드포인트 .NET 어플리케이션을 개발하기 위해서는 다음의 단계를 수행합니다.

### [따라하기] HTTP 엔드포인트 개발

1. Visual Studio .NET 프로젝트를 생성합니다.
2. 웹 참조 추가를 사용하여 proxy 클래스를 생성합니다.
   프로젝트를 생성한 다음, 프로젝트 메뉴의 웹 참조 추가 옵션을 사용하여, proxy 클래스를 생성할 수 있습니다. 웹 참조 추가 대화상자에서 proxy 클래스를 생성하기 위해서는 WSDL 문서에 대한 URL을 지정해야 합니다.
   웹 참조 추가 대화상자에서 웹 참조 명칭을 지정할 수 있습니다. 지정한 웹 참조 명칭은 클라이언트 코드내에서 proxy 클래스에 대한 로컬 네임스페이스의 역할을 수행합니다. 웹 서비스가 동일한 컴퓨터에 설치되어 있는 경우라면, 기본값은 localhost로 지정됩니다.
3. proxy 인스턴스를 생성합니다.
   Proxy 클래스의 메서드를 호출하기 위해서는, 먼저 코드에서 proxy 클래스에 대한 인스턴스를 생성해야 합니다. 변수를 선언하고, NEW 키워드를 사용하여, 메모리내에 proxy 클래스의 인스턴스를 생성할 수 있습니다.
4. 보안인증 신임장을 설정합니다.
   SQL Server HTTP 엔드포인트와 통신하기 위해서는, 클라이언트 어플리케이션의 유형과 상관없이, 적절한 보안 신임장을 설정해야 합니다.
5. proxy 인스턴스를 통해 웹 메서드를 호출합니다.
   인증이 되면, Proxy 인스턴스를 통해 적절한 웹 메서드를 호출할 수 있습니다. 웹 메서드에 대한 호출이 완료되면, 메서드에서 반환한 결과값을 추가 작업을 위해서 저장해 두어야 합니다.
6. 웹 서비스 응답 처리
   HTTP 엔드포인트의 FORMAT 인수에 ALL_RESULTS 값을 지정한 경우, 웹 서비스의 응답값에는 결과집합, 행 수, 오류 메시지, 경고 등이 개체배열형태로 반환됩니다.

### ■ proxy 클래스 생성

프로젝트를 생성한 다음, 프로젝트 메뉴의 웹 참조 추가 옵션을 사용하여, proxy 클래스를 생성할 수 있습니다. 웹 참조 추가 대화상자에서 proxy 클래스를 생성하기 위해서는 WSDL 문서에 대한 URL을 지정해야 합니다.

다음 예제는 WSDL URL을 나타냅니다.

```
http://localhost/sql/AdventureWorks?WSDL
```

### ■ proxy 인스턴스 생성

다음 예제와 같이, 변수를 선언하고, NEW 키워드를 사용하여, 메모리에 proxy 클래스에 대한 인스턴스를 생성할 수 있습니다.

**[따라하기] Proxy 생성**

```
'Visual Basic
Dim proxy As New AdventureWorks.sql_AdventureWorks
//Visual C#
AdventureWorks.sql_AdventureWorks proxy =
    new AdventureWorks.sql_AdventureWorks( );
```

앞의 예제는 Visual Basic .NET과, Visual C# .NET을 사용하여 proxy 인스턴스를 생성하는 방법을 나타냅니다. Proxy 클래스는 HTTP 엔드포인트의 명칭은 sql_AdventureWorks으로 호출됩니다. Proxy 클래스에 대한 네임스페이스는 localhost에서 AdventureWorks로 변경됩니다.

### ■ 보안인증 신임장 설정

다음 예제는 Windows 어플리케이션에 로그온한 사용자의 보안 신임장을 사용하는 방법을 나타냅니다.

#### [따라하기]보안인증신임장 설정

```
'Visual Basic
proxy.Credentials = _
    System.Net.CredentialCache.DefaultCredentials
//Visual C#
proxy.Credentials =
    System.Net.CredentialCache.DefaultCredentials;
```

다음 예제와 같이, 로그온명, 비밀번호, 도메인을 지정하여, 특정 Windows 사용자 계정에 대한 보안신임장을 변경할 수 있습니다.

#### [따라하기] 보안신임장 변경

```
'Visual Basic
proxy.Credentials = _
    New System.Net.NetworkCredential("Peter", _
    "P@ssw0rd", "Adventure-Works")
//Visual C#
proxy.Credentials =
    new System.Net.NetworkCredential("Peter",
    "P@ssw0rd", "Adventure-Works");
```

*[참고]*
*위와 동일한 방법을 통해, SQL Server 로그온을 사용할 수 있습니다. 도메인명은 지정하지 않습니다.*

### ■ 웹 메서드 호출

다음 예제는 GetProducts 웹 메서드를 호출하는 방법을 나타냅니다.

**[따라하기] GetProducts 메서드 호출**

```
'Visual Basic
Dim results( ) As Object = proxy.GetProducts
//Visual C#
object[] results = proxy.GetProducts;
```

### ■ 응답 처리

다음 예제는 결과값을 처리하기 위한 간단한 접근방법을 나타냅니다.

**[따라하기] 응답처리**

```
'Visual Basic
If results(0).ToString( ) = "System.Data.DataSet" Then
    Dim ds As DataSet = results(0)
    ' processing of DataSet
End If
If results(1) = "Client.AdventureWorks.SqlRowCount" Then
    Dim rowcount As AdventureWorks.SqlRowCount = results(1)
```

```
        ' processing of rowcount
End If
//Visual C#
if (results[0].ToString( ) == "System.Data.DataSet" )
{
        DataSet ds = (DataSet) results[0];
        //processing of DataSet
}
if (results[1].ToString( ) == "TestHarnessCS.AdventureWorks.SqlRowCount" )
{
        AdventureWorks.SqlRowCount rowcount =
                (AdventureWorks.SqlRowCount) results[1];
        //processing of rowcount
}
```

HTTP 엔드포인트에 FORMAT 인수를 ROWSETS_ONLY로 지정하면, 웹서비스에 대한 응답에 결과값에 해당하는 System.Data. DataSet 개체만 포함됩니다.

다음 예제는 ROWSET_ONLY로 지정한 웹 메서드에 대한 응답을 처리하는 방법을 나타냅니다.

### [따라하기] ROWSET_ONLY 웹 메서드 응답처리

```
'Visual Basic
Dim ds As DataSet = proxy.GetProductsDS
//Visual C#
DataSet ds = proxy.GetProductsDS;
```

# XML 개선기능

XML은 엔터프라이즈 비즈니스 솔루션의 핵심 기술이 되어 왔습니다. SQL Server 2005에서는 SQL Server 2000에서 지원하던 XML 지원 기능에 비해 한층 발전된 XML 기능을 지원하여, 관계형 데이터나 XML 데이터를 기반으로 솔루션을 쉽게 생성하도록 합니다.

SQL Server 2000에서는 SELECT 문장에 FOR XML 절을 추가하여 데이터베이스 엔진에서 쿼리결과를 XML 형식으로 반환하도록 지정하거나, OPENXML 함수를 통해 XML 관련 기능을 지원하였습니다. SQL Server 2005에서는 XML 지원기능이 한층 강화되었습니다.

## FOR XML 절 개선

SQL Server 2005에서는 FOR XML 절의 기능이 개선되었고, SQL Server 데이터베이스에 저장된 데이터를 XML 형식으로 생성하는 절차의 성능이 개선되었습니다.

### ■ RAW 모드 쿼리 내부에 ELEMENTS 지시어 사용

SQL Server 2000에서는 FOR XML 쿼리에 대한 RAW 모드에서는 속성기반 XML만 반환할 수 있었습니다. SQL Server 2005에서는 ELEMENTS 지시어를 사용하여 RAW 모드에서도 엘리먼트기반 XML을 반환하도록 지정할 수 있습니다.

```
SELECT ProductID, Name, ListPrice
FROM Production.Product
FOR XML RAW, ELEMENTS
```

## ■ Null 엘리먼트 지원

Null 컬럼을 생략하지 않고, xsi:nil 속성을 사용하여 빈 엘리먼트로 대체하여 표시하도록 설정할 수 있습니다. AUTO, RAW, PATH 모드 쿼리에서 XSINIL 옵션을 ELEMENTS 지시어와 함께 사용하거나, EXPLICIT 모드 쿼리에 elementxsinil 컬럼 모드를 사용할 수 있습니다.

다음 예제는 null 값을 빈 엘리먼트로 조회하는 방법을 나타냅니다.

```
-- ELEMENTS 지시어와 함께 XSINIL 사용
SELECT ProductID, Name, Color
FROM Production.Product Product
FOR XML AUTO, ELEMENTS XSINIL

-- EXPLICIT 모드에서 elementsxinil 사용
SELECT 1 AS Tag,
NULL AS Parent,
ProductID AS [Product!1!ProductID],

Name AS [Product!1!ProductName!element],
Color AS [Product!1!Color!elementxsinil]
FROM Production.Product
FOR XML EXPLICIT
```

## ■ 인라인 XSD 스키마

SQL Server 2000에서는 인라인 XDR 스키마만 반환할 수 있었습니다. SQL Server 2005에서는 XMLSCHEMA 지시어를 사용하여 인라인 XSD 스키마도 반환할 수 있습니다.

다음의 예제는 인라인 XSD 스키마를 조회하는 방법을 나타냅니다.

```
SELECT ProductID, Name, ListPrice
FROM Production.Product Product
FOR XML AUTO, XMLSCHEMA
```

### ■ XML 데이터형을 위한 TYPE 지시어

SQL Server 2005에는 XML 데이터형이 새로 추가되었습니다. FOR XML 쿼리에서 TYPE 지시어를 지정하면, 결과값을 varchar 문자열이 아닌 XML 데이터로 반환할 수 있습니다. AUTO 모드나 RAW 모드 쿼리에서 여러 레벨의 XML 결과값을 반환하기 위해서는 FOR XML 쿼리를 중첩해서 사용해야 합니다.

다음의 예제는 FOR XML 쿼리를 중첩해서 사용하기 위해서 TYPE 지시어를 사용하는 방법을 나타냅니다.

```
SELECT ProductID, Name, ListPrice,
    (SELECT ReviewerName, Comments
    FROM Production.ProductReview ProductReview
    WHERE ProductReview.ProductID = Product.ProductID
    FOR XML AUTO, ELEMENTS, TYPE)
FROM Production.Product Product
FOR XML AUTO
```

### ■ PATH 모드

SQL Server 2005에서는 FOR XML 쿼리에 대해서 PATH 모드를 새로 지원합니다. PATH 모드를 사용하면, 속성, 엘리먼트, 하위엘리먼트, 텍스트 노드, 데이터값에 값을 매핑하기 위해서 XPath 유사 구문을 사용하여 컬럼을 지정합니다. PATH 모드를 사용하면, EXPLICIT 모드를 사용하지 않고서도, 복잡한 XML 데이터를 생성할 수 있습니다.

다음 예제는 PATH 모드를 사용하는 방법을 나타냅니다.

```
SELECT ProductID AS "@ProductID",
    Name AS "*",
    Size AS "Description/@Size",
    Color AS "Description/text()"
FROM Production.Product
FOR XML PATH
```

### ■ 잘 정의된(well-formed) 결과값을 위한 ROOT 지시어

FOR XML 쿼리는 XML 일부를 반환하고, 잘 정의된(well-formed) XML 문서의 형태로 반환하는 것은 아닙니다. ROOT 지시어를 사용하여, 루트 엘리먼트의 명칭을 지정하면, FOR XML 쿼리결과에 루트 엘리먼트를 추가하여 잘 정의된 XML 문서형태로 반환할 수 있습니다.

다음 예제는 ROOT 지시어를 사용하는 방법을 나타냅니다.

```
SELECT ProductID, Name, ListPrice
FROM Production.Product Product
FOR XML AUTO, ROOT('Products')
```

### ■ RAW 모드와 PATH 모드에서 명명된 엘리먼트 지원

기본값으로, RAW 모드와 PATH 모드 쿼리에서는 쿼리에서 반환하는 각 행을 〈row〉 엘리먼트로 반환합니다. RAW 모드와 PATH 모드에서 반환하는 각 행에 대해 〈row〉 엘리먼트 대신, 별도의 엘리먼트 명칭을 부여할 수 있습니다.

다음 예제는 RAW 모드에서 명명된 엘리먼트를 반환하는 방법을 나타냅니다.

```
SELECT ProductID, Name, ListPrice
FROM Production.Product
FOR XML RAW( 'Product' )
```

## OPENXML 함수 개선기능

SQL Server 2000에서는 XML 문서를 관계형 표형식 행집합으로 표현하기 위해서 OPENXML 함수를 지원하였습니다. SQL Server 2005에서는 OPENXML 함수에 대해 개선된 기능을 제공하며, XML 데이터로부터 표 형식 데이터를 생성하는 솔루션을 생성할 수 있게 도와줍니다.

### ■ XML 문서를 XML 데이터형 값으로 지정

SQL Server 2000은, sp_xml_preparedocument 저장 프로시저를 사용하여, 문서 핸들을 생성하기 위해서 varchar, nvarcahr, text, ntext 변수를 사용하였습니다. SQL Server 2005에서는 XML 변수를 사용할 수 있습니다.

다음 예제는 sp_xml_preparedocument 저장 프로시저와 함께 XML 변수를 사용하는 방법을 나타냅니다.

```
DECLARE @doc xml
SET @doc = '<?xml version="1.0" ?>
<SalesInvoice InvoiceID="1000" CustomerID="123" >
    <Items>
        <Item ProductCode="12" Quantity="2" UnitPrice="12,99" />
        <Item ProductCode="41" Quantity="1" UnitPrice="17,45" />
        <Item ProductCode="2" Quantity="1" UnitPrice="2,99" />
```

```
        〈/Items〉
〈/SalesInvoice〉'

DECLARE @docHandle int
EXEC sp_xml_preparedocument @docHandle OUTPUT, @doc

SELECT * FROM
OPENXML(@docHandle, 'SalesInvoice/Items/Item' , 1)
WITH
(ProductCode int,
Quantity int,
UnitPrice smallmoney)

EXEC sp_xml_removedocument @docHandle
```

### ■ WITH 절에 XML 데이터형 지원

SQL Server 2005에는, OPENXML 함수에 WITH 절을 사용하여, XML 데이터형 컬럼을 반환할 수 있습니다.

다음 예제는 OPENXML 함수에 WITH 절을 사용하여 XML 컬럼을 반환하는 방법을 나타냅니다.

```
SELECT * FROM
OPENXML(@docHandle, 'SalesInvoice' , 1)
WITH
(InvoiceID int,
CustomerID int,
```

```
OrderDate datetime,
Items xml 'Items' )
```

### ■ 배치 수준 영역정의(scoping)

SQL Server 2000에서는 sp_xml_preparedocument 저장프로시저에 의해 반환된 XML 문서 핸들이 세션이 살아있는 동안 유지되었으며, 심지어 문서핸들 변수가 변수가 선언된 영역을 벗어나더라도 사용될 수 있었습니다. SQL Server 2005에서는 XML 문서 핸들이 배치가 종료되면 바로 제거되기 때문에, 서버 자원에 대한 사용률을 감소시켜 줍니다.

### ■ OPENXML 함수 사용방법

다음 예제는 XML 문서에서 관계형 표형식 행집합을 반환하는 방법을 나타냅니다. Xml 변수를 sp_xml_preparedocument 저장 프로시저에 전달할 수 있으며, OPENXML 함수의 WITH 절을 사용하여 XML 컬럼값을 반환할 수 있습니다.

```
DECLARE @doc xml
SET @doc = '<?xml version="1.0" ?>
<SalesInvoice InvoiceID="1000" CustomerID="123" OrderDate="2004-03-07">
    <Items>
        <Item ProductCode="12" Quantity="2" UnitPrice="12.99" />
        <Item ProductCode="41" Quantity="1" UnitPrice="17.45" />
        <Item ProductCode="2" Quantity="1" UnitPrice="2.99" />
    </Items>
</SalesInvoice>'

DECLARE @docHandle int
EXEC sp_xml_preparedocument @docHandle OUTPUT, @doc
```

```
SELECT * FROM
OPENXML(@docHandle, 'SalesInvoice' , 1)
WITH
(InvoiceID int,
CustomerID int,
OrderDate datetime,
Items xml 'Items' )
```

Items 컬럼에 저장된 XML 데이터가 다음과 같이 조회됩니다.

| InvoiceID | CustomerID | OrderDate | Items |
|---|---|---|---|
| 1000 | 123 | 2004-03-07 00:00:00.000 | <Items><br><Item ProductCode="12" Quantity="2" UnitPrice="12.99" /><br><Item ProductCode="41" Quantity="1" UnitPrice="17.45" /><br><Item ProductCode="2" Quantity="1" UnitPrice="2.99" /><br></Items> |

## XML 데이터형 사용

SQL Server 2005에서는 새로운 XML 데이터형을 지원하며, XML 데이터형을 통해, XML 데이터를 SQL Server 2005 데이터베이스 저장하고 조작할 수 있습니다. XML 데이터형을 사용하는 방법과 데이터베이스내에서 XML 스키마를 관리하는 방법에 대해서 설명합니다.

### ■ 데이터베이스에 XML 데이터 저장

SQL Server 2000의 XML 지원기능을 통해, 관계형 데이터를 XML로 변환할 수 있으며, XML 데이터를 관계형 데이터로 변환할 수도 있습니다. 하지만, SQL Server 2000에서 지원하는 XML 기능은 XML 형식의 데이터를 분산 어플리케이션에서 서로 주고받기 위해서 사용할 수는 있지만, 여전히 데이터베이스에 저장될 때는 관계형 저장소에 저장될 수 밖에 없었습니다.

SQL Server 2005에서는 XML 과 관계형 데이터간에 단순한 변환기능도 지원할 뿐만 아니라, XML 데이터형을 통해, 데이터베이스에 XML 데이터를 저장할 수 있습니다.

### ■ 데이터베이스에 XML 데이터를 저장하는 이유

관계형 데이터베이스에 XML 데이터 자체를 저장하는 기능은 어플리케이션 개발자에게 많은 장점을 제공합니다. XML 데이터를 직접 데이터베이스에 저장함으로써 얻을 수 있는 효과는 다음과 같습니다.

- 구조화된 데이터와 반구조화된(semistructured) 데이터를 단일 위치에 저장할 수 있기 때문에, 관리상 편의를 제공합니다.
- 관계형 모델내에 가변적인 컨텐츠를 정의할 수 있습니다.
- 데이터 저장소의 최적화와 쿼리 환경의 효율성을 보장하면서도, 어플리케이션에 특화된 요구사항을 충족시킬 수 있는 좀 더 유연한 데이터 모델을 제공할 수 있습니다.

## ■ 네이티브 XML 기능 지원

XML 데이터형에는 내부적으로 매우 효율적인 형태로 XML 문서의 InfoSet이 저장됩니다. XML 데이터는 불필요한 공백, 속성의 순서, 네임스페이스 전치사, 저장되지 않는 XML 선언부를 제외한 원본 XML 문서가 그대로 저장됩니다. SQL Server 2005의 XML 데이터형에서 지원하는 기능은 다음과 같습니다.

- XML 인덱스. XML 데이터형으로 정의된 컬럼에 XML 인덱스와 전체-텍스트 검색 인덱스를 생성할 수 있습니다. XML 인덱스를 사용하면, XML 데이터에 대한 쿼리의 성능을 개선할 수 있습니다.
- XQuery 기반 데이터 조회 메서드. XML 데이터형에서는 query, value, exist 메서드를 제공합니다. XML 데이터형에서 제공하는 메서드를 통해 XQuery를 사용하여, XML 데이터를 조회할 수 있습니다.
- XQuery 기반 데이터 변경 메서드. XML 데이터에서는 modify 메서드를 제공하며, XML 데이터에 대한 변경작업을 수행하여 XQuery 스펙을 확장하기 위해서 사용합니다.

## ■ 형식화된 XML과 형식화되지 않은 XML

### 형식화된 XML

XML 문서에 포함된 엘리먼트와 속성을 정의하고, 해당 문서에서 사용하는 네임스페이스를 지정하기 위한 XML 스키마가 포함된 XML입니다. XML 데이터형에 형식화된 XML 데이터를 저장하면, SQL Server 에서는 해당 스키마 정보를 기준으로 저장되는 XML 데이터에 대한 유효성 검사를 수행하게 되며, 스키마에 저장된 XML 데이터형을 기초로 하여 적절한 SQL Server 데이터 유형을 할당하기 위해 내부적인 저장소를 최적화합니다.

### 형식화되지 않은 XML

스키마정보가 포함되지 않은 XML입니다. XML 데이터형에 형식화되지 않은 상태로 XML 데이터가 저장되면, SQL Server 에서 해당 XML 데이터에 대해 유효성검사를 수행하지 않습니다. 하지만, 해당 XML 데이터가 잘 구성된(well-formed) 데이터라는 것은 보장할 수 있습니다.

## 형식화되지 않은 XML 사용방법

스키마에 의해 유효성검사를 하지 않아도 되는 XML 데이터를 저장하기 위해서는 형식화되지 않은 XML 데이터를 XML 데이터형 컬럼과 변수에 저장할 수 있습니다. 형식화되지 않은 XML 데이터를 저장하기 위한 어플리케이션을 개발하기 위해서는 XML 데이터형에 형식화되지 않은 XML 데이터를 저장하는 방법에 대해서 이해할 필요가 있습니다.

### ■ XML 컬럼과 변수를 정의

형식화되지 않은 XML 데이터를 저장할 xml 컬럼이나 xml 데이터형을 정의하는 방법은 다른 컬럼이나 SQL Server 데이터형을 정의하는 방법과 동일합니다.
컬럼을 정의하기 위해서는, 다음의 예제와 같이, CREATE TABLE 문장에 xml 데이터형을 지정하면 됩니다.

```
CREATE TABLE Sales.Invoices
(InvoiceID int,
SalesDate datetime,
CustomerID int,
ItemList xml)
```

형식화되지 않은 xml 데이터형 변수를 정의하기 위해서는, 다음의 예제와 같이, DECLARE 문장에 xml 데이터형을 지정하면 됩니다.
DECLARE @itemDoc xml

### ■ 문자열 값의 암시적인 캐스팅

XML 데이터형 변수나 XML 데이터형 컬럼에 형식화되지 않은 XML 데이터를 할당하기 위한 가장 단순한 방법은 잘 정의된(well-formed) 문서나 XML 문서의 일부분을 문자열(varchar, nvarchar, text, ntext) 값으로 할당하는 것입니다. SQL Server 에서는 자동적으로 해당 문자열값을 XML 데이터형으로 암시적으로 변환합니다.

다음 예제는 형식화되지 않은 XML 값을 할당하기 위해 암시적인 형변환을 사용하는 방법을 나타냅니다.

```
DECLARE @itemString nvarchar(2000)
SET @itemString = '<Items>
<Item ProductID="2" Quantity="3" />
<Item ProductID="4" Quantity="1" />
</Items>'

DECLARE @itemDoc xml
SET @itemDoc = @itemString

INSERT INTO Sales.Invoices
VALUES
(1, GetDate(), 2, @itemDoc)
```

동일한 원칙을 사용하여, 다음 예제와 같이, 고정 문자열 표현식으로 XML 데이터형 컬럼에 데이터를 추가할 수 있습니다.

```
INSERT INTO Sales.Invoices
VALUES
(1, GetDate(), 2, '<Items>
<Item ProductID="2" Quantity="3" />
<Item ProductID="4" Quantity="1" />
</Items>' )
```

### ■ 문자열을 명시적으로 XML으로 캐스팅

명시적으로, T-SQL CAST 함수를 사용하여 문자열 값을 XML 데이터형으로 형변환할 수 있습니다.

```
DECLARE @castedDoc xml
SET @castedDoc = CAST(@itemString AS xml)
```

### ■ 문자열을 XML로 명시적으로 형변환

명시적으로, T-SQL CONVERT 함수를 사용하여 문자열 값을 XML 데이터형으로 형변환할 수 있습니다.

```
DECLARE @convertedDoc xml
SET @convertedDoc = CONVERT(xml, @itemString)
```

### ■ 잘 구성된(well-formed) XML 사용

문자열 값을 XML 데이터형으로 형변환하는 방법과 상관없이, XML 데이터형에 저장될 XML 데이터는 반드시 잘 정의된(well-formed) XML 데이터이어야 하며, 잘 정의된(well-formed) XML 데이터가 아닌 경우에는 오류가 발생합니다. XML 데이터형 컬럼 또는 XML 데이터형 변수에 저장되기 위해서는, 단일 루트 엘리먼트를 가지는 잘 정의된 XML 문서이거나, 다수의 잘 정의된 엘리먼트로 구성된 잘 정의된 XML 문서의 일부(fragment)이어야 합니다.

다음 예제는 잘 정의된 XML 데이터와 잘 정의되지 않은 XML 데이터를 나타냅니다.

```
DECLARE @itemXml xml
-- 잘 정의된 XML 문서. 정상적으로 할당됨
SET @itemXml = '<Items>
<Item ProductID="2" Quantity="3" />
<Item ProductID="4" Quantity="1" />
</Items>'

-- 잘 정의된 XML 조각. 정상적으로 할당됨.
SET @itemXml = '<Item ProductID="2" Quantity="3" />
<Item ProductID="4" Quantity="1" />'

-- 잘 정의되지 않은 XML 데이터. 오류 발생.
SET @itemXml = '<Items>
<Item ProductID="2" Quantity="3" />
<Item ProductID="4" Quantity="1" />'
```

## XML 스키마 관리

형식화된 XML 데이터를 사용하기 위해서는, 데이터베이스 XML 스키마를 등록해야 합니다. XML 스키마에는 형식화된 XML 문서에서 사용하는 네임스페이스와, XML 문서에 포함된 엘리먼트와 속성에 대한 정의가 포함되어 있습니다.

### ■ XML 스키마 컬렉션 생성

XML 스키마는 데이터베이스의 XML 스키마 컬렉션 개체에 등록됩니다. 각 XML 스키마 컬렉션에는 하나 또는 그 이상의 XML 스키마가 포함되며, 각 스키마에는 사용할 네임스페이스가 지정됩니다.(XML 스키마의 TargetNamespace 속성에 지정)
XML 스키마 컬렉션을 생성하기 위해서는, CREATE XML SCHEMA COLLECTION 문장을

사용합니다. CREATE XML SCHEMA COLLECTION 문장에 대한 구문은 다음과 같습니다.

```
CREATE XML SCHEMA COLLECTION sql_identifier AS Expression
```

sql_identifier 는 XML 스키마 컬렉션에 대한 유효한 T-SQL 식별자입니다.
Expression 은 하나 또는 그 이상의 XML 스키마 문서를 포함하고 있는 XML 데이터입니다.
다음 예제는 XML 스키마 컬렉션을 생성하는 방법을 나타냅니다.

```
CREATE XML SCHEMA COLLECTION ResumeSchemaCollection
AS
N' 〈?xml version=" 1.0" ?〉
〈xsd:schema targetNamespace=" http://schemas.adventure-
works.com/EmployeeResume"
    xmlns=" http://schemas.adventure-works.com/EmployeeResume"
    elementFormDefault=" qualified"
    attributeFormDefault=" unqualified"
    xmlns:xsd=" http://www.w3.org/2001/XMLSchema" 〉
    〈xsd:element name=" resume" 〉
      〈xsd:complexType〉
        〈xsd:sequence〉
          〈xsd:element name=" name" type=" xsd:string" minOccurs=" 1"
          maxOccurs=" 1" /〉
            〈xsd:element name=" employmentHistory" 〉
              〈xsd:complexType〉
                〈xsd:sequence minOccurs=" 1" maxOccurs=" unbounded" 〉
                  〈xsd:element name=" employer" 〉
                    〈xsd:complexType〉
                      〈xsd:simpleContent〉
                        〈xsd:extension base=" xsd:string" 〉
                          〈xsd:attribute name=" endDate" use=" optional" /〉
```

```
                    </xsd:extension>
                </xsd:simpleContent>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>'
```

### ■ 스키마 정보 조회

다음 예제와 같이, sys.xml_schema_collections 카탈로그 뷰를 사용하여, 데이터베이스에 저장된 스키마 컬렉션에 대한 정보를 조회할 수 있습니다.

```
SELECT * FROM sys.xml_schema_collections
```

다음 예제와 같이, sys.xml_schema_collections 카탈로그 뷰를 사용하여, 데이터베이스 스키마 컬렉션에 정의된 개별 XML 네임스페이스를 조회할 수 있습니다.

```
SELECT * FROM sys.xml_schema_namespaces
```

다음 예제와 같이, sys.xml_components 카탈로그 뷰를 사용하여, 데이터베이스에 정의된 XML 구성요소에 대한 정보를 조회할 수 있습니다.

```
SELECT * FROM sys.xml_components
```

### ■ 스키마 컬렉션 수정

ALTER XML SCHEMA COLLECTION 명령을 사용하여, XML 스키마 컬렉션을 변경할 수 있습니다. ALTER XML SCHEMA COLLECTION 명령을 사용하여, 스키마 컬렉션에 스키마를 추가하거나 제거할 수 있습니다.

### ■ 스키마 컬렉션 삭제

다음 예제와 같이, DROP XML SCHEMA COLLECTION 명령을 사용하여, XML 스키마 컬렉션을 제거할 수 있습니다.

```
DROP XML SCHEMA COLLECTION ResumeSchemaCollection
```

형식화된 XML 컬럼에서 참조하고 있는 스키마를 포함하고 있는 XML 스키마 컬렉션은 삭제할 수 없습니다. 해당 XML 스키마 컬렉션을 삭제하기 위해서는, 사전에 테이블을 변경하거나 삭제하여, XML 스키마에 대한 참조정보를 삭제해야 합니다.

## 형식화된 XML 사용방법

XML 스키마 컬렉션을 등록한 다음, 형식화된 XML 데이터에 대한 유효성 검사를 하기 위해 XML 스키마를 사용할 수 있습니다.

### ■ 형식화된 컬럼 또는 변수 정의

형식화된 XML 데이터를 사용하기 위해서는, XML 컬럼 또는 XML 변수를 선언할 때, XML 데이터에 대한 유효성검사를 하기 위해 사용할 XML 스키마를 포함하고 있는 XML 스키마 컬렉션을 연결해 주어야 합니다.

다음 예제는 ResumeSchemaCollection XML 스키마 컬렉션을 사용하여, XML 데이터형 컬럼을 정의하는 방법을 나타냅니다.

```
CREATE TABLE HumanResources,EmployeeResume
(EmployeeID int,
 Resume xml (ResumeSchemaCollection))
```

동일한 방법으로, 다음 예제와 같이, 형식화된 XML 변수를 선언할 수 있습니다.

```
DECLARE @resumeDoc xml (ResumeSchemaCollection)
```

### ■ 형식화된 값 할당

형식화되지 않은 XML 데이터와 동일한 방법으로, 형식화된 XML 컬럼 또는 형식화된 XML 변수에 XML 데이터를 저장할 수 있습니다. 하지만, 입력되는 XML 문자열이 반드시 해당 컬럼이나 변수에 지정된 XML 스키마 컬렉션에 포함된 XML 스키마에 적합한 형식이어야 하며, 동일 대상 네임스페이스 내에서 선언되어야 합니다.

다음 예제는 형식화된 XML 데이터를 삽입하는 방법을 나타냅니다.

```
INSERT INTO HumanResources,EmployeeResume
VALUES
(1,
```

```
'<?xml version="1.0" ?>
<resume
  xmlns="http://schemas.adventure-works.com/EmployeeResume">
 <name>Guy Gilbert</name>
 <employmentHistory>
  <employer endDate="2000-07-07">Northwind Traders</employer>
  <employer>Adventure Works</employer>
 </employmentHistory>
</resume>' )
```

### ■ CONTENT 및 DOCUMENT 키워드 사용

형식화된 XML 컬럼 또는 변수에 단일 문서만 저장될 수 있도록 제한할 것인지, 여러 개의 문서로 구성된 XML 조각이 저장되도록 허용할 것인지를 지정하기 위해, 해당 XML 컬럼이나 변수를 정의할 때, CONTENT 키워드나 DOCUMENT 키워드를 지정할 수 있습니다. CONTENT 키워드를 지정하면, 유효한 XML 조각을 저장하도록 허용하며, DOCUMENT 키워드를 지정하면, 단일 XML 문서만 저장될 수 있도록 허용합니다.

아무 키워드도 지정되지 않으면, 기본값은 CONTENT로 정의됩니다.

다음 예제는 DOCUMENT 키워드를 사용하여 XML 컬럼에 단일 XML 문서만 저장되도록 하는 방법을 나타냅니다.

```
CREATE TABLE HumanResources.EmployeeResume
(EmployeeID int,
 Resume xml (DOCUMENT ResumeSchemaCollection))
```

## XML 인덱스 관리

### ■ XML 인덱스란?

XML 컬럼에 대한 쿼리 성능을 향상시키기 위해서, XML 인덱스를 생성할 수 있습니다. XML 인덱스를 생성할 때 고려해야 할 고려사항은 다음과 같습니다.

- XML 인덱스를 생성하려고 하는 테이블에 클러스터형 기본키가 반드시 존재해야 합니다. XML 인덱스가 존재하는 경우, 해당 테이블의 클러스터형 기본키를 수정할 수 없습니다. 해당 테이블의 클러스터형 기본키를 수정하기 위해서는, 먼저 모든 XML 인덱스를 삭제해야만 합니다.
- XML 컬럼별로 기본 XML 인덱스는 하나만 생성할 수 있습니다.
- PATH, PROPERTY, VALUE 쿼리를 위해, 부가 XML 인덱스를 생성할 수 있습니다.
- 동일한 테이블에 XML 인덱스와 비-XML 인덱스를 동시에 생성할 수 없습니다.

### ■ 기본 XML 인덱스 생성

CREATE PRIMARY XML INDEX 명령을 사용하여 XML 컬럼에 기본 XML 인덱스를 생성할 수 있습니다.

```
CREATE PRIMARY XML INDEX xidx_Item
ON Sales.Invoices(ItemList)
```

### ■ PATH 인덱스 생성

경로나 데이터값을 지정하여, XML 컬럼으로부터 데이터를 조회하는 쿼리가 존재한다면, 부가 PATH 인덱스를 생성할 것을 고려해야 합니다.
예를 들어, */ItemList/Item[@ProductID="1"]*와 같은 XPath 표현식이 존재하는지 체크해야 하는 쿼리를 실행해야 한다면, PATH 인덱스를 생성할 것을 고려해야 합니다.

다음 예제는 사전에 생성된 기본 XML 인덱스를 사용하여 PATH 인덱스를 생성하는 방법을 나타냅니다.

```
CREATE XML INDEX xidx_ItemPath
ON Sales.Invoices(ItemList)
USING XML INDEX xidx_Item
FOR PATH
```

### ■ PROPERTY 인덱스 생성

경로를 지정하여, XML 컬럼으로부터 노드 값을 조회해야 하는 쿼리를 실행해야 하는 경우에는 부가 PROPERTY 인덱스를 생성할 것을 고려해야 합니다.
예를 들어, *(/ItemList/Item/@ProductID)[1]*와 같은 XPath 표현식에 의해서 지정된 노드의 값을 반환하는 쿼리가 존재한다면, PROPERTY 인덱스를 생성할 것을 고려해야 합니다.

다음 예제는, 사전에 생성된 기본 XML 인덱스를 사용하여 PROPERTY 인덱스를 생성하는 방법을 나타냅니다.

```
CREATE XML INDEX xidx_ItemProp
ON Sales.Invoices(ItemList)
USING XML INDEX xidx_Item
FOR PROPERTY
```

### ■ VALUE 인덱스 생성

부정확한 경로를 지정하여, XML 컬럼으로부터 데이터를 조회하는 쿼리를 실행해야 한다면, 부가 VALUE 인덱스를 생성할 것을 고려해야 합니다.
예를 들어, *(//Item[@ProductID="1"]*와 같은 XPATH 표현식으로 특정 노드의 존재여부를 체크해야하는 쿼리를 실행해야 한다면, VALUE 인덱스를 생성할 것을 고려해야 합니다.

다음 예제는 사전에 생성된 기본 XML 인덱스를 사용하여 VALUE 인덱스를 생성하는 방법을 나타냅니다.

```
CREATE XML INDEX xidx_ItemVal
ON Sales.Invoices(ItemList)
USING XML INDEX xidx_Item
FOR VALUE
```

### ■ XML 인덱스 수정

ALTER INDEX 문장을 사용하여, 다음 예제와 같이, XML 인덱스(기본 또는 부가)를 변경할 수 있습니다.

```
ALTER INDEX xidx_Item ON Sales.Invoices REBUILD
```

### ■ XML 인덱스 삭제

DROP INDEX 문장을 사용하여, 다음 예제와 같이, XML 인덱스를 삭제할 수 있습니다.

```
DROP INDEX xidx_Item ON Sales.Invoices
```

# XQuery 지원기능

XQuery는 XML에 대한 쿼리 언어입니다. XML 데이터를 조회하기 위한 XQuery 표현식 기반 xml 데이터형의 메서드를 살펴보고 XQuery 구문에 대해서 살펴봅니다. XQuery 구문에는 XPath 2.0 표현식이 포함되어 있으며, XQuery를 사용하여 XML 데이터원본에 복잡한 쿼리를 실행할 수 있습니다. SQL Server 에 제공하는 xml 데이터형에서는 XQuery 표현식을 사용하여 xml 데이터를 조회하고 변경하기 위해서 사용할 수 있는 메서드를 제공합니다.
SQL Server 2005의 XQuery 지원기능은 W3C XQuery 1.0 언어 스펙을 기초로 합니다.
( http://www.w3.org/XML/Query 참조)

## XQuery 구문

XQuery는 크게 두가지 부분으로 구성됩니다. 네임스페이스를 선언하고, 스키마를 추가하기 위한 머리글 부분(생략가능)과 xml 데이터를 조회하기 위해서 사용하는 실제 XQuery 표현식이 포함된 본문 부분으로 나눌 수 있습니다. XQuery 표현식은 조회하고자 하는 XML 노드에 대한 간략한 경로 정보일 수도 있고, XML 결과를 생성하기 위한 복잡한 표현식일 수도 있습니다.

XQuery 경로는 XPath 언어를 기반으로 하며, XML 문서상에서 조회하고자 하는 노드의 위치를 나타냅니다. 경로는 절대경로(루트 엘리먼트로부터 XML 트리의 특정 노드의 위치를 표현)로 지정될 수도 있고, 상대경로(이미 알려진 노드로부터 조회하고자 하는 노드의 위치를 상대적으로 표현)로 지정될 수도 있습니다. 다음의 표는 간단한 XQuery 경로 예제를 나타냅니다.

| 예제 경로 | 설명 |
|---|---|
| /InvoiceList/Invoice | 〈InvoiceList〉 루트 엘리먼트에 포함되어 있는 모든 〈Invoice〉 엘리먼트 |
| (/InvoiceList/Invoice) [2] | 〈InvoiceList〉 루트 엘리먼트에 포함되어 있는 두번째 〈Invoice〉 엘리먼트 |
| (InvoiceList/Invoice/@InvoiceNo) [1] | 〈InvoiceList〉 루트 엘리먼트에 포함되어 있는 첫번째 〈Invoice〉 엘리먼트의 InvoiceNo 속성 |
| (InvoiceList/Invoice/Customer/text( ))[1] | 〈InvoiceList〉 루트 엘리먼트에 포함되어 있는 〈Invoice〉 엘리먼트의 하위 첫번째 〈Customer〉 엘리먼트의 텍스트 |
| /InvoiceList/Invoice[@InvoiceNo=1000] | 〈InvoiceList〉 루트 엘리먼트에 포함되어 있는 모든 〈Invoice〉 엘리먼트 중에서 InvoiceNo 속성이 1000인 〈Invoice〉 엘리먼트 |

## FLOWR 문장

XQuery 언어 스펙에 포함되어 있는 for, let, order by, where, return 문장을 통상적으로 FLOWR ("flower" 라고 읽음) 라고 부릅니다. SQL Server 2005에서는 for, where, return 문장을 지원하며, 각각의 사용용도는 다음과 같습니다.

| 문장 | 설명 |
|---|---|
| For | XML 문서의 동일 수준의 노드를 반복하며 처리하기 위해서 사용합니다. |
| Where | 노드를 반복처리할 때 필터링 조건을 지정하기 위해서 사용합니다.<br>XQuery 언어에는 Where 문장과 함께 사용할 수 있는 count( )와 같은 함수가 포함되어 있습니다. |
| Return | 반복구조내에서 반환할 XML을 지정하기 위해서 사용합니다. |

다음의 예제는 각 〈Invoice〉 엘리먼트별로 〈Items〉 자식 엘리먼트에 포함된 하나 이상의 〈Item〉 엘리먼트의 목록을 반환합니다.

```
for $i in /InvoiceList/Invoice
where count($i/Items/Item) > 1
return $i
```

## XQuery 표현식 사용예제

XQuery 표현식 중 가장 일반적으로 사용되는 유형을 예제와 함께 소개합니다.

**원본XML**

```
〈InvoiceList〉
    〈Invoice InvoiceNo="1000" 〉
        〈Customer〉Kim Abercrombie〈/Customer〉
        〈Items〉
            〈Item Product="1" Price="1.99" Quantity="2" /〉
            〈Item Product="3" Price="2.49" Quantity="1" /〉
        〈/Items〉
    〈/Invoice〉
    〈Invoice InvoiceNo="1001" 〉
        〈Customer〉Sean Chai〈/Customer〉
        〈Items〉
            〈Item Product="1" Price="1.99" Quantity="2" /〉
        〈/Items〉
    〈/Invoice〉
〈/InvoiceList〉
```

### ■ 단순한 XQuery 표현식 사용

구문 : /InvoiceList/Invoice

**결과값**

```
〈Invoice InvoiceNo="1000" 〉
    〈Customer〉Kim Abercrombie〈/Customer〉
    〈Items〉
        〈Item Product="1" Price="1.99" Quantity="2" /〉
        〈Item Product="3" Price="2.49" Quantity="1" /〉
    〈/Items〉
〈/Invoice〉
〈Invoice InvoiceNo="1001" 〉
    〈Customer〉Sean Chai〈/Customer〉
    〈Items〉
        〈Item Product="1" Price="1.99" Quantity="2" /〉
    〈/Items〉
〈/Invoice〉
```

### ■ XQuery 조건 사용

구문

/InvoiceList/Invoice[@InvoiceNo=1000]

**결과값**

```
〈Invoice InvoiceNo=" 1000" 〉
    〈Customer〉Kim Abercrombie〈/Customer〉
    〈Items〉
        〈Item Product=" 1" Price=" 1.99" Quantity=" 2" /〉
        〈Item Product=" 3" Price=" 2.49" Quantity=" 1" /〉
    〈/Items〉
〈/Invoice〉
```

### ■ For 문장과 Return 문장 사용

```
구문
for $i in /InvoiceList/Invoice/Items/
Item[../@InvoiceNo=1000]
return $i
```

**결과값**

```
〈Item Product=" 1" Price=" 1.99" Quantity=" 2" /〉
〈Item Product=" 3" Price=" 2.49" Quantity=" 1" /〉
```

■ For 문장과 Return 문장을 사용하여 XML 생성

```
구문
〈OrderedItems〉
    {
        for $i in /InvoiceList/Invoice/Items/Item
        return $i
    }
〈/OrderedItems〉
```

**결과값**

```
〈OrderedItems〉
    〈Item Product="1" Price="1.99" Quantity="2" /〉
    〈Item Product="3" Price="2.49" Quantity="1" /〉
    〈Item Product="1" Price="1.99" Quantity="2" /〉
〈/OrderedItems〉
```

■ For 문장과 Return 문장을 사용하여 속성과 값을 반환

```
구문
〈OrderedItems〉
{
for $i in /InvoiceList/Invoice/Items/Item
return 〈Product〉
{$i/@Quantity}
{string($i/@Product)}
〈/Product〉
}
〈/OrderedItems〉
```

**결과값**

```
〈OrderedItems〉
    〈Product Quantity=" 2" 〉1 〈/Product〉
    〈Product Quantity=" 1" 〉3 〈/Product〉
    〈Product Quantity=" 2" 〉1 〈/Product〉
〈/OrderedItems〉
```

### ■ For, Where, Return 문장 사용

```
구문
〈MultiItemInvoices〉
{
for $i in /InvoiceList/Invoice
where count($i/Items/Item) 〉 1
return $i
}
〈/MultiItemInvoices〉
```

**결과값**

```
〈MultiItemInvoices〉
    〈Invoice InvoiceNo=" 1000" 〉
        〈Customer〉Kim Abercrombie 〈/Customer〉
        〈Items〉
            〈Item Product=" 1"  Price=" 1,99"  Quantity=" 2"  /〉
            〈Item Product=" 3"  Price=" 2,49"  Quantity=" 1"  /〉
        〈/Items〉
    〈/Invoice〉
〈/MultiItemInvoices〉
```

### ■ 머리글에 네임스페이스 지정

```
구문
declare namespace awi =
"http://schemas.adventure-works.com/Invoices";
/awi:InvoiceList/awi:Invoice[@InvoiceNo=1000]
```

**결과값**

```
<awi:Invoice xmlns:awi="http://schemas.adventure-works.com/Invoices"
        InvoiceNo="1000">
    <awi:Customer>Kim Abercrombie</awi:Customer>
    <awi:Items>
        <awi:Item Product="1" Price="1.99" Quantity="2" />
        <awi:Item Product="3" Price="2.49" Quantity="1" />
    </awi:Items>
</awi:Invoice>
```

### ■ 기본 네임스페이스 사용

```
구문
declare default namespace =
"http://schemas.adventure-works.com/Invoices";
/InvoiceList/Invoice[@InvoiceNo=1000]
```

**결과값**

```
<Invoice xmlns="http://schemas.adventure-works.com/Invoices"
        InvoiceNo="1000">
    <Customer>Kim Abercrombie</Customer>
```

```
    <Items>
        <Item Product="1" Price="1.99" Quantity="2" />
        <Item Product="3" Price="2.49" Quantity="1" />
    </Items>
</Invoice>
```

## XML 데이터형에서 제공하는 메서드를 사용하여 쿼리실행

SQL Server 2005 XML 데이터형은 XML 데이터를 쿼리하고 수정하기 위해서 사용할 수 있는 네 가지 메서드를 제공합니다. 각 메서드는 대부분의 개발자에게 익숙한 구문인 데이터형.메서드_명칭으로 호출할 수 있습니다. 각 메서드의 기능에 대해서 이해하면, XML 데이터를 데이터베이스에서 처리하는 어플리케이션을 개발하는데 도움이 될 것입니다.

### ■ Query 메서드

Query 메서드는 XML 데이터형에 저장된 데이터에서 XML을 추출하기 위해서 사용합니다. Query 메서드의 매개변수로 전달되는 XQuery 표현식에 지정된 결과값이 조회됩니다.

```
SELECT xmlCol.query( 'declare default namespace =
 "http://schemas.adventure-works.com/InvoiceList";
<InvoiceNumbers>
{
for $i in /InvoiceList/Invoice
return <InvoiceNo>
{number($i/@InvoiceNo)}
</InvoiceNo>
}
</InvoiceNumbers>' )
```

### ■ Value 메서드

Value 메서드는 XML 문서로부터 단일 값을 반환하기 위해서 사용합니다. value 메서드를 사용하기 위해서는 XQuery 표현식을 XML 데이터에 포함된 단일 노드를 식별할 수 있는 형태로 지정해야 하며, 반환되는 값이 T-SQL 데이터형이 되도록 지정해야 합니다.

```
SELECT xmlCol.value( 'declare default namespace =
 "http://schemas.adventure-works.com/InvoiceList" ;
/InvoiceList/Invoice/@InvoiceNo)[1]' ,  'int' )
```

### ■ Exist 메서드

Exist 메서드는 XML 문서에 지정된 노드가 존재하는지 여부를 판단하기 위해서 사용합니다. Exist 메서드가 1을 반환하면, 지정된 노드가 XML 문서내에서 하나 또는 그 이상 존재한다는 것을 의미하며, 0을 반환하면, 지정된 노드가 존재하지 않는다는 것을 의미합니다.

```
SELECT xmlCol.exist( 'declare default namespace =
 "http://schemas.adventure-works.com/InvoiceList" ;
/InvoiceList/Invoice[@InvoiceNo=1000]' )
```

### ■ 관계형 테이블의 컬럼과 변수의 바인딩

SQL Server 2005에서는 XQuery 언어를 사용하여 xml 데이터형 컬럼을 조회하기 위한 메서드가 포함된 SELECT 문장에서, 관계형 데이터 컬럼을 참조할 수 있도록 지원합니다. XML 데이터형 컬럼으로부터 XML 데이터를 조회하기 위해, XML 데이터형 메서드가 포함된 SELECT 문장에서, sql:column 함수를 사용하여, XML 데이터안에 비-xml 데이터 컬럼값을 포함시킬 수 있습니다. 또한, sql:variable 확장을 사용하여, 저장 프로시저내에서 변수를 참조할 수 있습니다. XML 데이터내부에 비-xml 컬럼 값을 포함시키기 위해, sql:column 함수를 사용하는 예제는 다음과 같습니다.

```
SELECT StoreName, Invoices.query( 'declare default namespace=
"http://schemas.adventure-works.com/Invoices";
〈Invoices〉
〈Store〉{sql:column("StoreName")}〈/Store〉
{
for $i in /InvoiceList/Invoice
return $i
}
〈/Invoices〉' ) InvoicesWithStoreName
FROM Stores
```

## Modify 메서드를 사용하여 XML 데이터 변경

XML 데이터형 컬럼에 저장된 XML 데이터를 변경하기 위해서 Modify 메서드를 사용합니다. Modify 메서드는 XQuery 언어 스펙에 대해 insert, replace, delete 확장기능을 지원합니다. 세 가지 확장기능은 XML DML로서 참조할 수 있습니다.

- INSERT 문장을 사용하여 XML 데이터에 노드를 추가할 수 있습니다.
- REPLACE 문장을 사용하여 XML 데이터를 변경할 수 있습니다.
- DELETE 문장을 사용하여 XML 데이터에서 특정 노드를 삭제할 수 있습니다.

### ■ INSERT 문장

Modify 메서드와 함께 INSERT 문장을 사용하여, XML 데이터형 컬럼이나 변수에 저장된 XML 데이터에 노드를 추가할 수 있습니다. INSERT 문장에 대한 구문은 다음과 같습니다.

```
insert Expression1 (
{as first | as last} into | after | before
Expression2 )
```

INSERT 키워드에 지정할 수 있는 매개변수는 다음과 같습니다.

| 매개변수 | 설명 |
|---|---|
| Expression1 | 추가될 노드를 지정하기 위한 표현식으로, XML 문자열 형식으로 지정해야 합니다. (예를 들어, <Item Product=" 5" Quantity=" 1" />) 또한, 텍스트 노드에 추가할 엘리먼트 표현식을 지정할 수도 있습니다. (예를 들어, element SalesPerson { "Bill" }) 마지막으로, 속성에 추가할 속성 표현식을 지정할 수 있습니다. (예를 들어, attribute discount { "1.50" }) |
| as first | 계층구조의 첫번째 노드에 새로운 XML을 추가하기 위해서 사용합니다. |
| as last | 계층구조의 마지막 노드에 새로운 XML을 추가하기 위해서 사용합니다. |
| Into | Expression2 위치에 Expression1을 추가하기 위해서 사용합니다. |
| After | Expression2 뒤에 Expression1을 추가하기 위해서 사용합니다. |
| Before | Expression2 앞에 Expression1을 추가하기 위해서 사용합니다. |
| Expression2 | XML 문서에 포함된 기존 노드를 지정하기 위한 XQuery 표현식. |

다음 예제는 Modify 메서드와 함께 INSERT Xquery 문장을 사용하는 방법을 나타냅니다.

```
SET @xmlDoc.modify
( ' declare default namespace = " http://schemas.adventure-works.com/InvoiceList" ;
    insert element salesperson { "Bill" }
    as first into (/InvoiceList/Invoice)[1]' )
```

**■ replace 문장**

XML 데이터를 변경하기 위해서 modify 메서드와 함께 REPLACE 문장을 사용합니다. REPLACE 문장에 대한 구문은 다음과 같습니다.

```
replace value of
Expression1
with
Expression2
```

REPLACE 문장에서 사용할 수 있는 매개변수는 다음과 같습니다.

| 매개변수 | 설명 |
|---|---|
| Expression1 | 값을 변경할 노드를 지정하기 위한 XQuery 표현식 |
| Expression2 | 대체할 노드에 새로 지정할 값 |

다음의 예제는 Modify 메서드와 함께 REPLACE 문장을 사용하는 방법을 나타냅니다.

```
SET xmlCol.modify
( 'declare default namespace =" http://schemas.adventure-works.com/InvoiceList" ;
    replace value of (/InvoiceList/Invoice/SalesPerson/text( ))[1]
    with "Ted"' )
```

### ■ DELETE 문장

XML 데이터에서 지정된 노드를 삭제하기 위해서 Modify메서드와 함께 DELETE 문장을 사용합니다. DELETE 문장에 대한 구문은 다음과 같습니다.

```
delete Expression
```

Expression 매개변수는 삭제할 노드를 지정하기 위한 XQuery 표현식입니다.
다음 예제는 modify 메서드와 함께 DELETE 문장을 사용하는 방법을 나타냅니다.

```
SET xmlCol.modify
( 'declare default namespace =" http://schemas.adventure-works.com/InvoiceList" ;
      delete (/InvoiceList/Invoice/SalesPerson)[1]' )
```

## Nodes 메서드를 사용하여 XML 데이터 부분추출

Xml 데이터형에서는 nodes 메서드를 통해, XML 데이터를 관계형 테이블 형식으로 생성할 수 있는 기능을 제공합니다. Nodes 메서드는 XQuery 표현식으로 지정된 각 노드를 행집합 형식으로 반환합니다.

nodes 메서드의 구문은 다음과 같습니다.

```
xmlvalue.nodes (XQuery) [AS] Table(Column)
```

Nodes 메서드에서 사용할 수 있는 매개변수는 다음과 같습니다.

| 매개변수 | 설명 |
|---|---|
| XmlValue | XML 데이터형 변수 또는 컬럼 |
| XQuery | 반환하고자 하는 노드를 지정하기 위한 XQuery 표현식 |
| Table(Column) | 결과값으로 반환할 테이블명과 컬럼명. 결과값 테이블은 순차적으로 수행되는 쿼리에서 데이터를 추출하기 위한 원본으로 사용될 수 있습니다. |

Xml 데이터형 변수나 컬럼에서 nodes 메서드를 사용하여 관계형 테이블 형식의 데이터를 조회할 수 있습니다.

- XML 데이터형 변수에서 관계형 테이블 형식의 데이터를 추출하기 위해서, nodes 메서드에서 반환하는 결과행집합에 대해서, query, value, exist와 같은 메서드를 사용합니다.
- Xml 데이터형 컬럼에서 관계형 테이블 형식의 데이터를 반환하기 위해서, nodes 메서드와 함께 APPLY 연산자를 사용합니다.

### ■ XML 변수에 대해 nodes 메서드 사용법

Xml 데이터형 변수에서 관계형 테이블 형식의 데이터를 추출하기 위해서, nodes 메서드에서 반환하는 행집합에 query, value, exist 메서드를 사용할 수 있습니다. 다음 예제는 xml 변수에서 관계형 테이블 형식의 주문 데이터를 추출하는 방법을 나타냅니다.

```
DECLARE @xmlOrder xml
SET @xmlOrder = '<?xml version="1.0" ?>
    <Order OrderID="1000" OrderDate="2005-06-04">
        <LineItem ProductID="1" Price="2.99" Quantity="3" />
        <LineItem ProductID="2" Price="3.99" Quantity="1" />
    </Order>'
```

```
SELECT nCol.value( '@ProductID' , 'integer' ) ProductID,
       nCol.value( '@Quantity' , 'integer' ) Quantity
FROM   @xmlOrder.nodes( '/Order/LineItem' ) AS nTable(nCol)
```

위의 코드를 실행하면 다음과 같은 결과가 반환됩니다.

| ProductID | Quantity |
|---|---|
| 1 | 3 |
| 2 | 1 |

### ■ XML 컬럼에 대해 nodes 메서드 사용하는 방법

Xml 컬럼에서 관계형 테이블 형식의 데이터를 반환하게 하기 위해서, nodes 메서드와 함께 APPLY 연산자를 사용합니다. 다음 예제는 nodes 메서드를 사용하여 XML 컬럼으로부터 주문 데이터를 추출하는 방법을 나타냅니다.

```
SELECT nCol.value( '../@OrderID[1]' , 'int' ) OrderID,
       nCol.value( '../@OrderDate[1]' , 'datetime' ) OrderDate,
       nCol.value( '@ProductID[1]' , 'int' ) ProductID,
       nCol.value( '@Price[1]' , 'money' ) Price,
       nCol.value( '@Quantity[1]' , 'int' ) Quantity
FROM Orders_X
CROSS APPLY OrderDoc.nodes( '/Order/LineItem' ) AS nTable(nCol)
```

위의 코드는 다음과 같은 결과값을 반환합니다.

| OrderID | OrderDate | ProductID | Price | Quantity |
|---|---|---|---|---|
| 1000 | 2005-06-04 00:00:00,000 | 1 | 2,99 | 1 |
| 1000 | 2005-06-04 00:00:00,000 | 2 | 3,99 | 2 |
| 1001 | 2005-06-04 00:00:00,000 | 2 | 3,99 | 1 |
| 1002 | 2005-06-04 00:00:00,000 | 1 | 3,99 | 1 |

# SMO(SQL Management Objects) 지원

관리자가 수행해야 하는 관리작업을 자동화하면, 관리작업에 대한 오류나 일관성이 없는 관리작업의 발생가능성을 줄여줄 수 있습니다. 특히, 다수의 SQL Server 와 인스턴스가 존재하는 엔터프라이즈 환경에서는 더욱 관리작업의 자동화가 필요합니다. SQL Server 2005에서는 반복적인 공통 관리 작업의 자동화를 지원하기 위한 SQL 관리 개체(SMO) API를 제공합니다.

## SMO 란?

SMO는 관리작업을 위한 프로그램과 스크립트를 생성할 수 있는 기반의 역할을 합니다. SQL Server 개체와 작업을 관리하기 위해서 사용할 프로그램을 개발하기 위한 프로그래밍 개체 집합을 의미합니다.

SMO는 .NET 어셈블리(Microsoft.SqlServer.Smo.dll)로 구현되어 있습니다. SMO는 개체의 계층구조와 고유의 개체 모델을 제공합니다.

SMO를 사용하여 서버 구성 옵션의 조회 및 변경, SQL Server 에이전트 작업 관리, 백업일정계획 관리 등과 같은, 대부분의 관리작업을 수행하기 위한 어플리케이션을 개발할 수 있습니다.

### ■ SMO에서 지원하는 SQL Server 버전

SMO는 SQL Server 7.0, SQL Server 2000, SQL Server 2005가 구동중인 서버에 대한 프로그래밍 기반을 제공합니다. 단, SQL Server 가 호환성 수준 60이나 65로 구동중인 경우에는, 일부 작업에 대해서 오류가 발생하게 됩니다.

## ■ SMO 구현

SQL 관리 개체는 .NET 어셈블리내에 클래스로 구현되어 있습니다. SMO를 사용하여 어플리케이션을 개발하고자 할 때에서는, 기본적으로 다음과 같은 어셈블리를 사용해야 합니다.

- **Microsoft.SqlServer.Smo.dll** - 대부분의 SMO 클래스를 제공합니다.
- **Microsoft.SqlServer.ConnectionInfo.dll** - SQL Server 인스턴스와 연결에 사용되는 클래스를 제공합니다.

각 어셈블리에는 다수의 네임스페이스와 클래스가 포함되어 있습니다.

## ■ 기존 SQL-DMO의 대체기능

SQL 분산관리개체(SQL-DMO)는 이전 버전 SQL Server 에서 지원되던 데이터베이스 관리를 위한 프로그래밍 인터페이스입니다. SMO는 기존 SQL-DMO를 대체하는 역할을 합니다. SMO는 SQL-DMO에 비해서 좀 더 효율적으로 자원을 사용하여, 관리작업을 자동화하기 위해 좀 더 세밀한 부분까지 통제할 수 있습니다. 기존 SQL-DMO에 대한 지식을 보유하고 있는 사용자라면, SMO에 대한 학습에 도움이 될 것입니다.

## ■ SMO의 기능

SQL Server 2005에서 지원하는 SMO의 주요기능은 다음과 같습니다.

- ***최적화된 인스턴스 생성***. SMO는 SQL-DMO에서 사용하는 것처럼 전체 개체에 대한 인스턴스를 생성하지 않고 가능한 범위내에서 필요한 부분에 대한 인스턴스를 생성하기 때문에 성능을 향상시킬 수 있습니다. 코드에서 명시적으로 개체에 대한 참조를 설정하는 경우에만, 전체 개체에 대해서 인스턴스를 생성하게 됩니다.
- ***실행 보류***. SMO는 기본값으로 각 명령을 즉시 수행합니다. 실행 보류 기능을 통해, 데이터베이스에 변경사항을 적용하기 전에 검토하기 위해서, 특정 시점까지 명령의 실행을 보류할 수 있습니다.

- ***WMI 호환***. SMO는 SQL Server WMI 공급자에 대한 래퍼 클래스를 제공하여, SQL Server 에 대한 자동화된 코드의 일관성을 위지하기 위해 WMI 프로그래밍 인터페이스를 제공합니다.
- ***스크립트 작성***. SMO는 Scripter 클래스를 통해, 단일 개체를 기반으로 하여 전체 종속성 트리에 대한 스크립트 작업을 자동화하는 것과 같은, 개선된 스크립트 기능을 제공합니다.
- ***서버옵션구성***. SMO 는 sp_configure 저장 프로시저와 동일한 방법으로 서버 구성 옵션을 조회하고 변경할 수 있는 기능을 제공합니다.

## SMO와 SQL Server 분산 관리 개체(SQL-DMO) 비교

SMO는 이전 버전 SQL Server 에서 제공하던 SQL Server 분산관리개체(SQL-DMO)의 역할을 완전히 대체합니다. SQL Server 2005 데이터베이스 엔진에 상당히 많은 내부구조상의 변경이 있었기 때문에, 이러한 변경사항의 효과를 충분히 활용할 수 있는 SMO를 사용해야 합니다. SMO는 가장 효율적인 자원활용을 할 수 있도록 최적화되어 있으며, 네트워크상에서 개체에 대한 인스턴스를 생성하기 위해 사용하는 메모리 사용량을 절감해 줍니다. SMO를 사용하면, 개발자가 개체에 대한 인스턴스를 생성하는 시점에 대해서 좀 더 세밀하게 통제할 수 있습니다.

> ***[참고]***
> *SQL Server 2005에도 SQL-DMO가 여전히 지원되지만, SQL-DMO는 하위버전호환성을 보장하기 위한 목적으로만 사용해야 합니다. SQL-DMO는 SQL Server 2005 환경에 최적화되어 있지 않습니다. 새로운 개발 프로젝트에서는 반드시 SMO를 사용해야 합니다.*

### ■ SMO and WMI

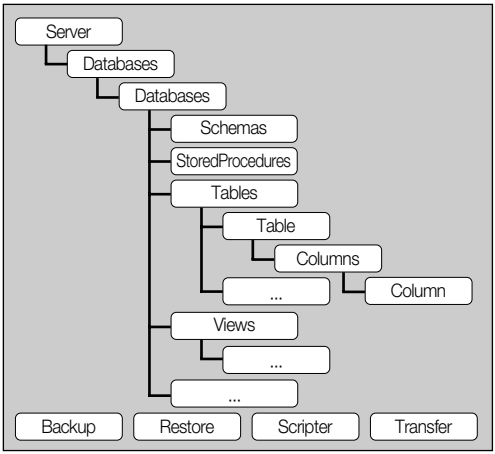
SQL-DMO는 SQL Server 에 대해 프로그래밍 관점에서의 인터페이스를 제공하였습니다. SMO에서는 프로그래밍 기능에 추가하여, WMI 와의 인터페이스를 단순화하는 기능을 통해 좀 더 관리자적인 접근이 가능하도록 지원합니다. SMO와 WMI을 사용하여 관리자는, SQL Server 2005 서버와 인스턴스에 대한 모니터링 및 구성작업을 효율적으로 관리할 수 있습니다.

### ■ SMO Scripting

SMO는 Scripter 클래스를 통해 좀 더 개선된 스크립트 작성 기능을 제공합니다. 예를 들어, Scripter 클래스를 사용하여, 단일 개체을 기반으로 한 전체 종속성 트리에 대한 스크립트 작업을 자동화할 수 있습니다.

## SMO 개체모델

SMO 개체 모델에는 데이터베이스와 데이터베이스 서버를 관리하기 위한 전체 프로그래밍 인터페이스를 제공하는 역할을 하는 여러 개의 네임스페이스와 클래스가 포함되어 있습니다. 일부 클래스는 쉽게 찾을 수 있지만, 일부 클래스는 참조하기 위해 좀 더 자세한 탐색이 필요한 경우도 있습니다. SMO에서는 관리작업을 지원하기 위해, 크게 instance 클래스와 utility 클래스를 지원합니다.




〈그림.SMO 개체 계층구조〉

## ■ 계층구조로 된 Instance 클래스

Instance 클래스는 데이터베이스 서버, 데이터베이스, 테이블, 뷰, 사용자정의함수 등과 같은, SQL Server 개체를 포함하고 있습니다. 각 클래스는 표준 데이터베이스 개체 계층 구조와 직접적으로 매핑되어 있기 때문에 좀 더 쉽게 개발작업을 수행할 수 있습니다. 부모 개체는 다수의 자식 개체(예를 들어, 하나의 테이블에 다수의 컬럼 포함 가능)를 포함할 수 있고, 각 자식개체의 집합은 컬렉션 개체로 표현할 수 있습니다. 자식 개체가 하나만 존재한다면, 해당 자식 개체를 별도의 독립된 개체로 표현할 수 있습니다.

모든 컬렉션 클래스는 컬렉션 클래스에 포함된 개별개체에 접근하기 위한 인터페이스를 제공합니다. 예를 들어, Views 컬렉션 클래스에는 데이터베이스에 따라 하나 또는 그 이상의 View 개체를 포함할 수 있습니다. 또한, Views 컬렉션 클래스에 View 개체가 하나도 포함되지 않을 수도 있습니다.

### ■ Utility 클래스

Utility 클래스는 개별 서버나 데이터베이스 개체에 매핑되지 않고, 부가기능을 제공하기 위해서 존재합니다.

## SMO 개체 참조

SMO를 사용하여 관리작업을 자동화하는 처리절차는 다음과 같습니다.

1. .NET 기반 클라이언트 어플리케이션 생성.
2. SQL SMO 어셈블리 참조.
3. 코드내에서 Imports (Microsoft Visual C# 에서는 using 구문)구문을 사용하여, 필요한 네임스페이스에 대한 참조를 설정

### ■ .NET 기반 클라이언트 어플리케이션 생성

SMO는 모든 .NET 기반 어플리케이션(Windows Form 어플리케이션, 콘솔 어플리케이션, ASP.NET 웹 어플리케이션)에서 사용할 수 있습니다. 어플리케이션의 목적에 따라 개발할 게 될 클라이언트의 유형이 결정됩니다. 예를 들어, 본사에서 전체 데이터베이스를 통합관리해야 하는 상황이라면, 클라이언트 컴퓨터에 별도의 설치가 필요없는 ASP.NET 기반 인트라넷 어플리케이션을 개발하는 것이 바람직합니다. 또한, 전세계 어느 곳에서도 관리작업을 수행할 수 있도록, ASP.NET 인트라넷 어플리케이션에서 보안이 보장된 연결을 사용할 수 있습니다.

### ■ SMO 어셈블리 참조

어플리케이션에서 관리작업을 수행하기 위해서는, 반드시 Microsoft.SqlServer.Smo.dll 어셈블리에 대한 참조를 .NET 프로젝트에 추가해야 합니다. 필요에 따라 추가적인 기능을 위해 다음과 같은, 관련 어셈블리도 추가할 수 있습니다.

| 어셈블리 | 설명 |
|---|---|
| Microsoft.SqlServer.ConnectionInfo.dll | SQL Server 인스턴스에 대한 연결을 명시적으로 관리하기 위한 클래스 제공 |
| Microsoft.SqlServer.SmoEnum | 일부 SMO 개체의 작업을 보조하기 위한 열거형 포함 |

SMO 어셈블리는 SQL Server 2005의 클라이언트 도구 설치 옵션의 일부로 포함되어 전역 어셈블리 캐시(GAC)에 등록됩니다. SMO 관련 어셈블리는 클라이언트 어플리케이션의 설치패키지에 일부로 포함되어 클라이언트 컴퓨터에 배포할 수 있습니다.

Microsoft Visual Studio® .NET 프로젝트에서 참조를 추가하기 위해서는 다음 절차를 수행합니다.

1. 프로젝트 메뉴-참조추가 옵션을 선택합니다.
2. 참조 추가 대화상자의 어셈블리 목록에서 각 어셈블리를 선택한 다음 확인을 클릭합니다.

### ■ 네임스페이스 추가

코드에 대한 가독성을 높이기 위해, 다음 예제와 같이, Visual Basic .NET에서는 Import 구문, Visual C#.NET에서는 using 구문을 사용하여, SMO 네임스페이스를 추가할 수 있습니다. .NET 프로젝트에 네임스페이스를 추가하면, 각 클래스에 대한 전체 이름을 지정하지 않고서도 해당 네임스페이스에 포함된 클래스에 쉽게 접근할 수 있습니다.

```
'Visual Basic
Imports Microsoft.SqlServer.Management.Smo
Imports Microsoft.SqlServer.Management.Common
```

```
//Visual C#
using Microsoft.SqlServer.Management.Smo;
using Microsoft.SqlServer.Management.Common;
```

## 서버속성조회

SMO를 사용하여 서버 및 데이터베이스 개체에 대한 정보를 조회할 수 있습니다. 서버 및 데이터베이스 개체에 대한 정보를 조회하기 위해서는, 다음과 같은 절차를 수행합니다.

1. 특정 서버명을 지정하여, Server 개체의 인스턴스를 생성합니다.
2. Databases 컬렉션이나 서버의 다양한 속성에 대해 필요한 작업을 수행합니다.

### ■ Server 개체 인스턴스 생성

서버에 대한 작업을 수행하기 위해서는, 먼저, Server 개체를 선언하고 초기화해야 합니다. Server 개체는 Server 개체가 초기화될 때, 생성자 클래스내부에서 자동적으로 Windows 인증을 사용하여 SQL Server 에 연결됩니다. 다음 예제는 DBSERVER라는 SQL Server 인스턴스에 연결하는 방법을 나타냅니다.

```
'Visual Basic
Dim svr As New Server("DBSERVER")

//Visual C#
Server svr = new Server("DBSERVER");
```

SQL Server 인증을 사용하거나, 명시적으로 연결정보를 통제해야할 필요가 있는 경우에는, ConnectionInfo 어셈블리에 대한 참조를 설정한 다음, 다음 예제와 같이, ServerConnection 개체를 사용해야 합니다.

```
'Visual Basic
Dim conn As New ServerConnection("DBSERVER", "sa", "P@ssword")
Dim svr As New Server(conn)

//Visual C#
ServerConnection conn = new ServerConnection("DBSERVER", "sa",
"P@ssword");
Server svr = new Server(conn);
```

다음 예제와 같이, SqlConnectionInfo 개체를 사용하여 좀 더 상세한 수준까지 연결정보를 설정할 수 있습니다.

```
'Visual Basic
Dim conInfo As New SqlConnectionInfo("DBSERVER")
With conInfo
    .DatabaseName = "AdventureWorks"
    .EncryptConnection = True
    .ApplicationName = "AWClient"
End With
Dim conn As New ServerConnection(conInfo)
Dim svr As New Server(conn)

//Visual C#
SqlConnectionInfo conInfo = new SqlConnectionInfo("DBSERVER");
conInfo.DatabaseName = "AdventureWorks";
conInfo.EncryptConnection = true;
```

```
conInfo.ApplicationName = "AWClient";
ServerConnection conn = new ServerConnection(conInfo);
Server svr = new Server(conn);
```

### ■ 속성에 대한 작업 수행

SQL Server 인스턴스에 연결을 생성한 다음, 다음 예제와 같이, Information, Settings, Configuration 속성을 사용하여 서버속성 정보를 조회할 수 있습니다.

```
' Visual Basic
Console.WriteLine(svr.Information.VersionString)
Console.WriteLine(svr.Settings.MasterDBPath)
Console.WriteLine(svr.Configuration.NestedTriggers.RunValue.ToString( ))

// Visual C#
Console.WriteLine(svr.Information.VersionString);
Console.WriteLine(svr.Settings.MasterDBPath);
Console.WriteLine(svr.Configuration.NestedTriggers.RunValue.ToString( ));
```

또한, Databases 속성을 사용하여, 설치된 데이터베이스와 데이터베이스 개체에 대한 정보를 조회할 수 있습니다. 다음 예제는, 특정 SQL Server 인스턴스에 포함된 각 데이터베이스에 대한 간단한 정보(데이터베이스명과 데이터베이스 크기)를 조회하는 방법을 나타냅니다.

```
'Visual Basic
For Each db As Database In svr.Databases
    Console.WriteLine(db.Name & ": " & db.Size & "K")
Next
```

```
//Visual C#
foreach (Database db in svr.Databases)
{
    Console.WriteLine(db.Name + ": " + db.Size + "K");
}
```

## SMO 개체 생성

SMO를 사용하여 새 데이터베이스 개체(데이터베이스, 테이블, 저장프로시저, 트리거, 뷰 등)를 생성할 수 있습니다. 데이터베이스 개체를 생성하는 처리절차는 다음과 같습니다.

1. 테이블, 컬럼 등 생성하고자 하는 데이터베이스 개체에 대한 적절한 변수를 생성합니다.
2. 해당 개체에 적절한 속성을 설정합니다.
3. 해당 개체에 대한 부모 개체에 자식 개체를 추가하기 위해서, 부모 개체에서 제공하는 Create 메서드를 호출합니다.

### ■ 변수 생성 및 초기화

속성을 설정하기 전에 추가하고자 하는 새 데이터베이스 개체에 대한 로컬 변수를 생성해야 합니다. 각 개체의 생성자 클래스에서는 개체를 생성하기 위해 다양한 속성값을 지정할 수 있도록 허용합니다. 생성자 클래스는 별도이 코딩없이 속성값을 설정할 수 있는 가장 효율적인 방법입니다. 하지만, 각 클래스별로 다른 생성자 클래스를 제공하기 때문에, 각 개체에 대한 자세한 정보를 확인하기 위해 해당 클래스에 대한 문서자료를 참조해야 합니다.

다음 예제는 데이터베이스 개체를 저장하기 위한 로컬 변수를 생성하고, AdventureWorks 데이터베이스를 참조하기 위해 Databases 컬렉션 속성에 인수로 데이터베이스명을 지정합니다. 그 다음, 해당 데이터베이스에 Table 개체와 두 개의 Column 개체를 추가하는 방법을 나타냅니다.

```
'Visual Basic
Dim AWDBase As Database = Svr.Databases("AdventureWorks")
Dim DiscountsTable As New Table(AWDBase, "Discounts")
Dim DiscountID As New Column(DiscountsTable, _
"DiscountID", DataType.Int)
Dim DiscountName As New Column(DiscountsTable, _
"DiscountName", DataType.NVarChar(40))

//Visual C#
Database AWDBase = svr.Databases["AdventureWorks"];
Table DiscountsTable = new Table(AWDBase, "Discounts");
Column DiscountID = new Column(DiscountsTable,
"DiscountID", DataType.Int);
Column DiscountName = new Column(DiscountsTable,
"DiscountName", DataType.NVarChar(40));
```

### ■ 속성 설정

개체에 대한 변수를 생성한 다음, 데이터베이스에 변경사항을 적용하기 전에 적절한 부가 속성을 지정할 수 있습니다. 다음 예제는 DiscountID 컬럼에 identity 속성을 설정하는 방법을 나타냅니다.

```
'Visual Basic
DiscountID.Identity = True

//Visual C#
DiscountID.Identity = true;
```

### ■ 데이터베이스 서버에 변경사항 적용

데이터베이스 개체를 저장할 변수를 생성하고, 적절한 속성을 설정한 다음, 새로 추가된 개체를 실제로 데이터베이스 서버에 추가하는 작업을 수행해야 합니다. 일반적으로, 다음의 예제와 같이, 부모 개체에 자식 개체를 추가하기 위해서 컬렉션 개체의 Add 메서드를 호출한 다음, 변경사항을 실제 데이터베이스 서버에 반영하기 위해 부모 개체에서 제공하는 Create 메서드를 호출합니다.

```
'Visual Basic
DiscountsTable.Columns.Add(DiscountID)
DiscountsTable.Columns.Add(DiscountName)
DiscountsTable.Create( )

//Visual C#
DiscountsTable.Columns.Add(DiscountID);
DiscountsTable.Columns.Add(DiscountName);
DiscountsTable.Create( );
```

현재 연결이 기본 실행 모드로 운영중인 상태라면, SMO에서는 해당 변경사항을 즉시 데이터베이스 서버에 반영합니다. 현재 연결의 SqlExecutionModes 속성이 CaptureSql 모드로 운영중인 상태라면, 향후 일정 시점이 경과된 이후에 실행될 수 있도록, 기존 T-SQL 문장이 해당 연결의 CapturedSql 속성에 저장됩니다.

## SMO를 사용하여 기존 개체 변경

SMO를 사용하여 기존에 생성되어 있는 데이터베이스 개체(뷰, 테이블, 사용자정의함수, 트리거 등)를 변경할 수 있습니다.

1. 수정대상이 되는 저장 프로시저나 테이블과 같은 적절한 개체를 지정합니다.
2. 해당 개체에 필요한 속성을 수정하거나, 해당 개체의 적절한 메서드를 호출합니다.
3. 변경사항을 데이터베이스 서버에 반영하기 위해 Alter 메서드를 호출합니다.

### ■ 수정할 대상 개체 지정

개체를 변경하기 전에, 적절한 개체 컬렉션에서 수정할 대상이 되는 개체의 위치를 지정해야 합니다. 다음 예제는 데이터베이스에 있는 테이블과 컬럼의 위치를 지정하는 방법을 나타냅니다.

```
'Visual Basic
Dim AWDBase As Database = Svr.Databases("AdventureWorks")
Dim DiscountsTable As Table = AWDBase.Tables("Discounts")
Dim DiscountName As Column = DiscountsTable.Columns("DiscountName")

//Visual C#
Database AWDBase = svr.Databases["AdventureWorks"];
Table DiscountsTable = AWDBase.Tables["Discounts "];
Column DiscountName = DiscountsTable.Columns["DiscountName"];
```

### ■ 속성 변경

수정할 대상이 되는 개체의 위치를 지정한 다음, 해당 개체에 대한 변경사항을 데이터베이스 서버로 반영하기 전에 필요한 속성을 수정합니다. 다음 예제는 컬럼의 Null 허용여부를 변경하는 방법을 나타냅니다.

```
'Visual Basic
DiscountName.Nullable = False

//Visual C#
DiscountName.Nullable = false;
```

또한, 개체의 속성값을 변경하는 대신 해당 개체에서 제공하는 메서드를 호출해야 하는 경우도 존재합니다. 예를 들어, Table 클래스, Column 클래스를 포함한 많은 클래스는 데이터베이스에 해당 개체를 삭제하기 위한 Drop 메서드를 제공합니다. 속성값을 변경하는 경우, 명시적으로 명령을 실행할 때까지 데이터베이스 서버에 변경사항이 적용되지 않는 것에 비해, 메서드를 호출하는 유형의 변경사항은 즉시 데이터베이스 서버에 변경사항이 반영됩니다. 다음 예제는 테이블에서 특정 컬럼을 즉시 삭제하는 방법을 나타냅니다.

```
'Visual Basic
DiscountName.Drop( )

//Visual C#
DiscountName.Drop( );
```

### ■ 서버에 속성값 변경사항을 적용

데이터베이스 개체에 대한 속성을 변경한 다음, 해당 변경사항을 명시적으로 데이터베이스 서버에 반영해야 합니다. 다음의 예제와 같이, 대부분의 SMO 개체는 변경된 속성값을 데이터베이스 서버에 반영하기 위해, Alter 메서드를 제공합니다.

```
'Visual Basic
DiscountName.Alter( )
```

```
//Visual C#
DiscountName.Alter( );
```

## SMO를 사용한 어플리케이션 개발 및 서버정보조회

Visual Studio 2005를 사용하여 SMO 기반 어플리케이션을 개발할 수 있습니다. Visual Studio 2005에서는 비즈니스 요구사항에 따라, Windows Form 어플리케이션, ASP.NET 웹 어플리케이션, 콘솔 어플리케이션을 모두 개발할 수 있습니다. 예를 들어, 본사에서 각 지사에서 위치한 데이터베이스를 관리해야 하는 경우, 클라이언트에 별도의 설치가 필요없는 인트라넷 기반 ASP.NET 어플리케이션을 개발할 수 있습니다.

### ■ .NET 기반 SMO 어플리케이션 개발

다음은 .NET 기반 SMO 어플리케이션을 개발하기 위한 단계를 나타냅니다.

1. Visual Studio 2005 실행.
2. 필요한 비즈니스 요구조건에 따라 적절한 어플리케이션 유형의 프로젝트를 새로 생성합니다. 예를 들어, Visual Basic.NET Windows Form 프로젝트를 생성합니다.
3. SQL SMO 어셈블리에 대한 참조를 설정합니다. SQL Server 에 대한 작업을 수행하기 위해서는 반드시 Microsoft.SqlServer.SMO 어셈블리를 참조해야 합니다. 어플리케이션의 기능상의 요구조건에 따라, Microsoft.SqlServer 네임스페이스에 포함된 다른 어셈블리(예를 들어, Microsoft.SqlServer.ConnectionInfo)도 추가해서 사용할 수 있습니다.
4. Visual Basic.NET의 경우 Import 명령을 사용하고, Visual C#.NET의 경우 using 명령을 사용하여, SMO 네임스페이스를 추가합니다.

5. Server 개체를 생성하고, ConnectionContext 개체를 획득한 다음, 서버명을 지정하고, 해당 서버에 연결하는 과정을 통해, 현재 운영 중인 SQL Server 에 연결을 생성합니다. SMO 어플리케이션을 개발하는 경우, 기본값으로, Windows 인증을 사용하여 서버에 연결합니다.
6. 어플리케이션 코드를 작성합니다.

### ■ 서버 정보 조회

SQL Server 가 운영중인 서버에 연결을 생성한 다음, Server 개체를 사용하여 서버 정보를 조회할 수 있습니다. Server 개체에는 서버 속성중 변경할 수 없는 속성을 포함하고 있는 information 속성이 포함되어 있습니다.

다음 표에는 어플리케이션에서 Information 개체에 포함되어 있는 변경할 수 없는 속성의 목록이 나타나 있습니다.

| 속성 | 설명 |
|---|---|
| Edition | 현재 운영중인 SQL Server 의 에디션 정보를 저장 |
| IsClustered | 현재 서버에 클러스터 환경이 구축되어 있는지 여부를 Boolean 값으로 저장 |
| IsSingleUser | 현재 서버가 읽기전용상태로 운영중인지 여부를 Boolean 값으로 저장 |
| Language | SQL Server 가 운영중인 컴퓨터의 기본 언어 정보를 저장 |
| NetName | 현재 서버의 NETBIOS 명을 저장 |
| OSVersion | 현재 서버의 운영체제 버전 정보를 저장 |
| Parent | 현재 Information 개체의 부모 개체에 해당하는 Server 개체를 저장 |
| PhysicalMemory | 현재 서버의 총 RAM 정보를 저장 |
| Platform | 현재 서버의 하드웨어 플랫폼 정보를 저장 |

| | |
|---|---|
| Processors | 현재 서버에 설치된 CPU 수를 저장 |
| Product | 현재 SQL Server 가 운영 중인 컴퓨터의 제품 타이틀 정보를 저장 |
| ProductLevel | 현재 SQL Server 가 운영 중인 컴퓨터의 제품 레벨 정보를 저장 |
| VersionString | 현재 SQL Server 가 운영 중인 컴퓨터의 버전 정보를 저장 |

## SMO를 사용하여 데이터베이스 백업

SMO를 사용하여 데이터베이스 백업과 같은, 주기적으로 반복되는 관리작업을 수행할 수 있습니다. Backup 개체의 속성을 커스터마이즈하여, 필요한 비즈니스 요구사항을 충족할 수 있는 백업 작업을 수행할 수 있습니다.

### ■ 데이터베이스 백업

다음은 SMO를 사용하여, 데이터베이스를 백업하기 위한 처리절차입니다.

1. SQL Server 가 운영중인 서버에 연결을 생성합니다.
2. Backup 개체를 선언하고 초기화합니다.
3. Backup 개체에 다음과 같은 속성을 설정합니다.
   - Action
   - BackupSetName
   - Database
   - DeviceType
   - 기타 필요한 속성(예를 들어, Incremental, NoRewind)
4. Backup 개체의 Devices.Add 메서드를 호출하고, 백업 디바이스에 대한 경로 정보를 전달합니다. Backup 개체의 SqlBackup 메서드를 호출하고, Server 개체를 전달합니다.

## ■ 백업관련속성

비즈니스 요구조건을 충족시키는 백업 작업을 수행하기 위해, Backup 개체의 속성을 커스터마이즈할 수 있습니다. 다음 표는 Backup 개체의 핵심 속성의 목록이 나타나 있습니다.

| 속성 | 설명 |
|---|---|
| Action | 수행할 백업의 유형 |
| BackupSetDescription | 백업 세트에 대한 텍스트 설명 |
| BackupSetName | 백업 세트를 식별하기 위한 명칭 |
| BlockSize | 백업 테이프를 포맷하기 위한 블록 크기 |
| Checksum | 백업 및 복원 작업동안 유효성검사를 위한 체크섬(checksum)값을 계산할 것인지 지정하기 위한 Booleam 값 |
| Database | 백업 또는 복원 작업을 수행할 대상 데이터베이스 |
| DatabaseFileGroups | 백업 작업의 대상이 되는 SQL Server 파일 그룹 |
| DatabaseFiles | 백업 작업의 대상이 되는 운영체제 파일 |
| Devices | 백업 작업을 수행할 백업 디바이스 |
| DeviceType | 백업 작업을 수행할 백업 디바이스의 디바이스 유형 |
| ExpirationDate | 백업데이터가 더 이상 유효성을 보장할 수 없다고 판단하는, 백업 세트의 최종유효기간을 위한 일자와 시간 정보 |
| FormatMedia | 백업 작업의 첫번째 단계로 백업 테이프를 포맷할 것인지 여부를 지정하기 위한 Boolean 값 |
| Incremental | 차등 백업을 수행할 것인지 여부를 지정하기 위한 Boolean 값 |
| Initialize | 백업 작업의 일부로, 백업작업 수행할 디바이스를 초기화할 것인지 여부를 지정하기 위한 Boolean 값 |

| | |
|---|---|
| LogTruncation | 백업 작업의 일부로 데이터베이스 로그 파일을 삭제하기 위해서 사용할 메서드 명칭 |
| MediaDescription | 백업 세트를 포함하고 있는 백업 미디어에 대한 텍스트 설명 |
| MediaName | 특정 미디어 세트를 식별하기 위한 미디어 명칭 |
| NoRewind | 백업 작업 후에 백업 데이프를 오픈된 상태로 유지할 것인지 여부를 지정하기 위한 Boolean 값 |
| Restart | 백업이 중단된 경우, 해당 백업 작업을 다시 시작할 것인지를 지정하기 위한 Boolean 값 |
| RetainDays | 백업 세트를 덮어쓰기(overwrite)전에 경과해야 하는 기간 일수 |
| SkipTapeHeader | 백업 미디어가 정상적으로 로드되었는지 체크하기 위한 운영 로직을 적용할 것인지 여부를 결정 |
| UnloadTapeAfter | 백업 작업이 완료된 다음, 백업 테이프 미디어를 다시 감아서 언로드할 것인지 여부를 지정하기 위한 Boolean 값 |

## SMO를 사용하여 데이터베이스 목록 생성 및 새 데이터베이스 생성

다음 예제는 SMO 네임스페이스에 대한 참조를 추가하고, 데이터베이스에 연결을 생성하고, 서버로부터 데이터베이스 목록을 조회한 다음, 새 데이터베이스를 추가하는 처리절차를 나타냅니다.

### ■ SMO 네임스페이스 참조

SMO를 사용하여 데이터베이스 관리 솔루션을 개발하는 경우에는, SMO 어셈블리에 대한 참조를 반드시 추가해야 합니다. 또한, 다음의 예제와 같이, SMO 클래스를 사용하기 위해서는 코드 파일에 SMO 네임스페이스를 추가해야 합니다. SMO 네임스페이스를 추가하면, 코드에 대한 가독성을 개선할 수 있습니다.

```
using System.SqlServer.Management.Common;
using System.SqlServer.Management.Smo;
```

### ■ 서버에 연결

다음 예제는 ConnectionContext 관리 개체를 사용하여 서버에 연결을 생성합니다. 서버명은 텍스트 박스 컨트롤에서 지정됩니다.

```
Server myServer = new Server( );
ServerConnection conn = myServer.Connection.Context;
conn.ServerInstance = txtServerName.Text;
conn.Connect( );
```

### ■ 데이터베이스목록생성

다음 예제는 Server 개체의 Databases 컬렉션을 사용하여 특정 서버의 모든 데이터베이스의 목록을 리스트박스 컨트롤에 표시하는 방법을 나타냅니다.

```
lstDatabases.Items.Clear( );
for (int i = 0; i < myServer.Databases.Count; i++)
{
    lstDatabases.Items.Add(myServer.Databases[i].Name);
}
```

### ■ 데이터베이스생성

다음 예제는 Database 개체의 Create 메서드를 사용하여 NewDatabase 라는 새 데이터베이스를 생성하는 방법을 나타냅니다.

```
Database newDb = new Database(myServer, "NewDatabase");
newDb.Create( );
```

# RMO(Replication Management Objects) 지원

SQL Server 2005에서 새로 제공하는, RMO를 사용하여, SQL Server 복제 관련 작업을 자동화할 수 있습니다. RMO를 사용하기 위해서는 RMO 라이브러리에 포함된 Replication 네임스페이스와 관련 클래스에 대해서 살펴보아야 합니다. 또한, RMO를 사용하여 프로그래밍 기반으로 복제를 관리하기 위해 사용하는 RMO 클래스에 대해서도 살펴보아야 합니다.

## RMO 란?

SQL Server Management Studio에서는 새 게시 마법사, 새 구독 마법사와 같은 사용하기 편리한 복제 도구를 지원합니다. 하지만, 일부의 경우에서는 복제 관리 작업을 자동화해야 하는 상황이 발생할 수 있습니다. SQL Server 2005에서는 복제에 대한 프로그래밍 인터페이스를 제공하기 위해 RMO를 지원합니다.

### ■ RMO에서 지원하는 SQL Server 버전 Server versions

SMO와 마찬가지로, RMO도 SQL Server 7.0, SQL Server 2000, SQL Server 2005를 지원합니다.

### ■ RMO 구현

RMO는 Microsoft.SqlServer.Rmo.dll 이라는 .NET 어셈블리내에 클래스로 구현되어 있습니다. Microsoft.SqlServer.Rmo.dll 어셈블리에는 Microsoft.SqlServer.Replication 네임스페이스가 포함되어 있으며, Microsoft.SqlServer.Replication 네임스페이스에는 복제관련 클래스가 포함되어 있습니다.

### ■ SQL-DMO 복제 관련 클래스 대체

RMO는 기존 SQL-DMO에서 지원하던 복제 관련 클래스를 대체하는 역할을 합니다. SQL-DMO 복제 관련 클래스에 대한 지식이 있는 사용자라면, 해당 지식을 RMO에서도 적용할 수 있습니다.

## RMO 서버에 연결

다음의 코드는 ServerConnection 개체에 대한 인스턴스를 생성하는 방법을 나타냅니다.

```
'Visual Basic
Dim conn As New ServerConnection("DBSERVER")

//Visual C#
ServerConnection conn = new ServerConnection("DBSERVER");
```

### ■ ReplicationServer 개체 생성

ServerConnection 개체를 생성한 다음, ReplicationServer 개체에 대한 인스턴스를 생성하기 위해서 사용합니다. ReplicationServer 개체는 복제가 구성되었는지 여부와 상관없이, SQL Server 7.0 또는 그 이후 버전 SQL Server 에 연결할 수 있습니다.

다음 예제는 ReplicationServer 개체에 대한 인스턴스를 생성하는 방법을 나타냅니다.

```
'Visual Basic
Dim rs As New ReplicationServer(conn)

//Visual C#
ReplicationServer rs = new ReplicationServer(conn);
```

## 복제 관리 작업

RMO를 사용하여, 단순하게 복제구성정보를 조회하는 것에부터 게시자, 분배자, 구독자를 생성하고 관리하는 것까지, 모든 복제관련 관리작업을 수행할 수 있습니다.

### ■ 복제구성정보 조회

ReplicationServer 클래스에서는 IsPublisher, IsDistributor, DistributorAvailable, HasRemotePublisher와 같은 속성과 복제 구성 정보를 조회하기 위한 속성을 제공합니다.

### ■ 게시와 배포 구성

RMO에는 게시와 배포를 구성하기 위해서 사용할 수 있는 DistributionDatabase, DistributionPublisher 같은 클래스가 포함되어 있습니다.

### ■ 게시와 아티클 관리

RMO에서는 트랜잭션 게시, 스냅샷 게시, 병합 게시를 생성하기 위해서 사용할 수 있는 TransPublication, MergePublication와 같은 클래스가 포함되어 있습니다. TransArticle 클래스 또는 MergeArticle 클래스를 사용하여, 게시할 아티클을 지정할 수 있습니다.

### ■ 구독 관리

TransSubscription 클래스와 MergeSubscription 클래스를 사용하여 밀어내기 방식 구독을 생성할 수 있습니다. 또한, TransPullSubscription 클래스와 MergePullSubscription 클래스를 사용하여 가져오기 방식 구독을 관리할 수 있습니다.

# 향상된 전체 텍스트 검색(Full Text Search)

2005에서는 이전 버전에서 볼 수 없었던 다양한 기능을 전체 텍스트 검색에 도입했습니다. 향상된 기능은 다음과 같습니다.

1. 전체 텍스트 카탈로그의 백업 및 복원
2. 데이터베이스 연결 및 분리 작업에 전체 텍스트 카탈로그 포함
3. XML 데이터의 전체 텍스트 인덱싱
4. 전체 텍스트 인덱싱 성능 개선 및 업그레이드
5. 병렬 서비스 보안
6. 다양한 상태보고

## 전체 텍스트 카탈로그 백업 및 복원

SQL 2000에서 제공되었던 전체 텍스트 검색 서비스와는 달리 SQL Server 2005 에서는 전체 텍스트 카탈로그에 통합 백업 및 복원 기능을 제공합니다. 2005에서는 데이터베이스 데이터와 함께 백업 및 복원을 하거나 별도로 전체 텍스트 카탈로그를 백업 및 복원할 수 있습니다. 이 기능을 통해 재해 복구에 필요한 시간이 줄어들게 되었고 카탈로그 전체를 다시 채울 필요가 없어졌고 전체 텍스트 카탈로그를 비롯하여 관련 데이터를 한 컴퓨터에서 다른 컴퓨터로 이동하는 작업이 간편해졌습니다. 위와 같은 기능은 아래와 같은 이점을 제공합니다.

1. 다른 데이터와 동일한 방법으로 하나 이상의 전체 텍스트 카탈로그를 백업 및 복원할 수 있습니다.
2. 복원 후 데이터 전체를 다시 채울 필요가 없습니다.
3. 복원 후 전체 텍스트 데이터를 업데이트하면 로그를 롤포워드하여 변경 내용을 반영합니다. 이 기능을 사용하려면 변경 내용 추적을 사용해야 합니다.

## 데이터베이스 연결 및 분리 작업에 전체 텍스트 카탈로그 포함

2005 에서는 관리자가 데이터베이스 연결 및 분리 작업을 수행할 때 전체 텍스트 카탈로그를 유지합니다. 이전 버전에서는 관리자가 전체 텍스트 카탈로그를 삭제하고 다시 작성해야 했습니다.

2005에서는 전체 텍스트 카탈로그를 데이터베이스의 일부로 인식합니다. 관리자는 데이터베이스를 분리하고 모든 데이터베이스 파일을 새 위치에 복사한 다음 데이터베이스를 다시 연결할 수 있습니다. 이 과정에서 전체 텍스트 카탈로그가 그대로 유지됩니다.

## XML 데이터의 전체 텍스트 인덱싱

2005 에는 XML 조각이나 문서를 저장할 수 있는 새로운 XML 데이터 형식이 도입되었습니다. SQL Server 의 전체 텍스트 검색은 이제 XML 데이터 형식을 기반으로 전체 텍스트 인덱스를 생성하는 기능과 XML 데이터 형식에 대해 전체 텍스트 쿼리를 작성하는 기능을 지원합니다.

## 전체 텍스트 인덱싱 성능 개선 및 업그레이드

2005의 전체 텍스트 검색에는 3.0으로 업그레이드된 Microsoft Search (MSSearch) 서비스가 포함되어 있습니다. 이 서비스의 업그레이드된 특징은 다음과 같습니다.

1. 대폭 개선된 전체 텍스트 인덱스 채우기 성능
2. 각 SQL Server 인스턴스에 MSSearch 3.0 인스턴스가 하나씩 제공
3. MSSearch 3.0은 SQL Server 과 동일한 서비스 계정으로 실행

## 병렬 서비스 보안

SQL Server 2005 전체 텍스트 검색은 SQL Server 인스턴스당 하나의 MSFTESQL(SQL용 Microsoft 전체 텍스트 검색 엔진) 서비스 인스턴스를 사용합니다. 각 MSFTESQL 인스턴스는 SQL Server 인스턴스가 사용하는 것과 동일한 서비스 계정으로 시작하고 실행됩니다. 관리자가 특정 인스턴스의 SQL Server 서비스 계정을 변경하면 관련된 MSFTESQL 서비스 계정도 업데이트됩니다.

이전 버전의 SQL Server 에서는 LocalSystem으로 실행되는 MSSearch 인스턴스 하나를 서버의 모든 인스턴스와 응용 프로그램에서 공유했습니다.

## 다양한 상태 보고

SQL Server 2005 의 전체 텍스트 검색은 다양한 상태 보고 기능을 통해 전체 텍스트를 보다 쉽게 구현하고 관리할 수 있도록 해 줍니다. 이러한 기능은 다음과 같습니다.

1. 카탈로그 내의 각 인덱스에 대한 상태 및 인덱싱 오류를 기록하는 채우기 상태 로그가 추가되었습니다.
2. 추가 서비스, 카탈로그 및 인덱스 상태, 구성 옵션을 사용할 수 있습니다.
3. SQL Server 프로필러에서 전체 텍스트 쿼리를 확인하여 문제를 진단하고 성능을 분석할 수 있습니다.

# SQL Server 2005의 향상된 보안기능

2003년 1월 23일 슬래머웜(Slammer Worm)이 전 세계 SQL서버를 강타했습니다. 그후 2년 동안 마이크로소프트는 SQL서버에 있어서 보안에 대한 부분에 많은 시간과 비용을 투자했습니다.

2005에서는 보안에 있어서 새로운 특징을 탑재해서 개발자와 관리자에게 좀더 강하고 편리한 보안 체계를 소개하게 되었습니다. 2005에서는 개체에 대한 계층적인 구조를 사용하여 보안을 향상시켰습니다.

다음의 표에서 향상된 대표적인 보안특징을 정리했습니다.

| 향상된 점 | 설명 |
|---|---|
| 1. SQL 로그인 계정에 대한 암호정책 가능 | 이전 버전에서는 윈도우서버에 정의된 암호화 정책이 SQL Server 로그인계정에 영향을 주지 못했습니다. 2005에서는 윈도우 보안 정책이 SQL Server 로그인계정에 그대로 영향을 주게 되었습니다. |
| 2. 계층적 보안 | SQL Server 2005에서 보안주체, 보안개체, permission에 대한 계층적 배열을 사용해서 서로 다른 영역의 개체를 보호하는 일관된 모델을 제공합니다. 이전 버전에 비해서 2005의 보안의 접근 방법을 통해서 사용자는 보안에 대해서 좀더 수월하게 할 수 있게 되었습니다. |
| 3. 사용자와 스키마의 분리 | 2005이전 버전에서 개체의 네임스페이스는 개체의 소유자를 가리켰습니다. 2005에서는 개체의 네임스페이스는 개체의 소유자와는 독립적으로 스키마를 이용해서 지정이 됩니다. 소유자와 개체 네임스페이스의 분리를 통해서 좀더 유연하고 관리가 용이한 프레임 워크를 제공합니다. |
| 4. 제한적인 메타데이터 접근 | SQL Server 2005는 메타데이터에 대한 허가가 부여된 개체에 대해서만 살펴 볼 수 있는 권한을 주어줍니다. |

| | |
|---|---|
| 5. 모듈의 컨텍스트 실행 | 저장프로시저나 함수와 같은 프로그램적인 모듈에 대해서 수행 문맥을 CREATE 문장을 이용하여 명시적으로 선언할 수 있습니다. |
| 6. 데이터암호화 | SQL Server 자체에서 데이터암호화를 지원하는 함수를 제공합니다. |
| 7. 기타 서버구성 값 | 2005 데이터베이스 엔진의 특징에서 보안에 관련된 구성옵션을 추가했습니다. |

## SQL 로그인 계정에 대한 암호정책 가능

■ 로그인 계정 생성 방법

- CREATE LOGIN 명령어
  로그인계정을 생성하기 위해 2000에서 사용해 왔던 sp_addlogin 과 sp_grantlogin 저장프로시저는 CREATE LOGIN 명령어로 대치 됩니다.

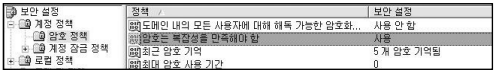
```
CREATE LOGIN [ComputerName\user1]
FROM windows  -- 윈도우계정 이용시
With default_database = adventureWorks
```

- sp_addlogin 과 sp_grantlogin은 과거버전의 호환성을 위해 제공됩니다.
- SQL 계정을 추가할 때도 마찬가지 원리로 생성합니다.

```
CREATE LOGIN [sqluser]
With password=' abc_1234' ,
default_database = adventureWorks
```

■ SQL Server 로그인 암호 정책

- 윈도우 사용자계정은 윈도우 정책에 의해서 암호화 정책이 정의됩니다.
- 윈도우 로컬보안계정에서 '암호는 복잡성을 만족해야 함' 을 사용으로 설정하면 다음과 같은 로그인은 생성되지 않습니다.




〈그림 로그인 암호 정책〉

```
CREATE LOGIN [simple_user]
With password=' 1111' ,
Default_database = adventureWorks
```

- 보안정책을 무시하고 생성을 하기 위해서는 다음과 같이 합니다.

```
CREATE LOGIN [simple_user]
With password=' 1111'
,default_database = adventureWorks
,CHECK_EXPIRATION = OFF
,CHECK_POLICY = OFF
```

- 옵션
- CREATE LOGIN 명령어서 사용할 수 있는 옵션의 일부분입니다.

| 옵션 | 설명 |
| --- | --- |
| HASHED | SQL Server 로그인에만 적용됩니다. PASSWORD 인수 다음에 입력한 암호가 이미 해시되었음을 지정합니다. 이 옵션을 선택하지 않으면 암호로 입력한 문자열이 데이터베이스에 저장되기 전에 해시됩니다. |
| MUST_ CHANGE | SQL Server 로그인에만 적용됩니다. 이 옵션을 지정한 경우 새 로그인을 처음 사용할 때 SQL Server 에서는 새 암호를 묻는 메시지를 표시합니다. 이 옵션이 사용되면 CHECK_EXPIRATION 옵션과 CHECK_POLICY이 같이 설정됩니다. |
| CHECK_ EXPIRATION | SQL Server 로그인에만 적용됩니다. 이 로그인에 암호 만료 정책을 적용할지 여부를 지정합니다. 기본값은 OFF입니다 |
| CHECK _POLICY | SQL Server 로그인에만 적용됩니다. SQL Server 가 실행 중인 컴퓨터의 Windows 암호 정책을 이 로그인에 적용하도록 지정합니다. 기본값은 ON입니다. |

**사용권한**

서버에 대한 ALTER ANY LOGIN 권한이 필요합니다. CREDENTIAL 옵션을 사용하는 경우에는 서버에 대한 ALTER ANY CREDENTIAL 권한도 필요합니다.

### [따라하기]내장함수를 이용해서 계정의 정보 알아내기

2005에서 지원하는 함수를 이용해서 계정의 정보를 알아낼 수 있습니다.

```
CREATE LOGIN Mic  WITH PASSWORD =' 111'
SELECT LOGINPROPERTY( 'Mic' ,  'IsLocked' )
SELECT LOGINPROPERTY( 'Mic' ,  'IsExpired' )
SELECT LOGINPROPERTY( 'Mic' ,  'LastSetTime' )
SELECT LOGINPROPERTY( 'Mic' ,  'IsMustChange' )
```

## 인가 (계층적인 인가 방식)

위의 인증절차를 통해서 들어온 사용자는 SQL Server 자원에 접근할 수 있는 인가를 받아야 합니다. 2005에서 이 인가부분 절차에 새로운 부분이 등장했습니다.

| 명칭 | 설명 |
|---|---|
| 보안주체 (Principal) | 개별적인 윈도우 로그인, SQL서버 로그인, 데이터베이스 사용자, application 역할, 데이터베이스 역할과 같은 SQL Server 내에서 인증과 인가에 사용되는 모든 단위의 일반적인 의미를 가리킵니다. |
| 보안개체 (Securable) | Endpoints, 데이터베이스, 전체 텍스트 카탈로그, 서비스 브로커, 테이블, 뷰 등과 같이 서버, 데이터베이스, 스키마 수준에서 지켜지는 개체들을 말합니다. |
| Grantor | 허가를 허용하는 보안주체를 의미합니다. |
| Grantee | 허가를 받은 보안주체를 의미합니다. |

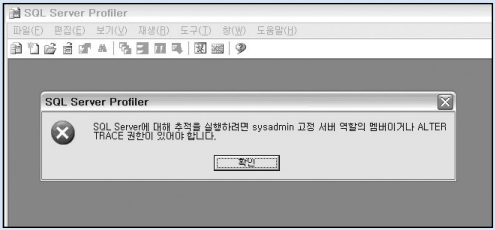
2000에서 어느 사용자에게 프로필러를 이용하기 위해서 sysadmin 권한을 주었다 생각해봅니다. 이 사용자는 프로필러뿐만 아니라 시스템 전체에 대해서 권한을 부여 받게 됩니다. 이런 문제는 웹호스팅 업체에서 충분히 일어날 수 있는 일입니다. A, B, C라는 웹호스팅 서비스를 받는 고객이 있을 때, A라는 고객은 자기 웹사이트에서 주로 사용되는 쿼리를 프로필러로 받고 싶습니다. 그래서, 그 사용자에게 sysadmin권한을 주면 다른 업체의 데이터베이스에 이런 저런 일을 할 수 있게 됩니다. 그럼 프로필러만 돌릴 수 있는 권한을 줄 수 있을까요? 2000까지는 방법이 없습니다.

2005에서는 서버 수준, 데이터베이스 수준, 스키마 수준, 개체 수준, 보안주체 수준에 대해서 보안 개체를 제공합니다. 예를 들면 스키마 수준에서 일정한 허가를 부여하면 특정 스키마내에 존재하는 개체에 대한 허가를 가지게 됩니다.

예를 들어 보기로 합시다.
서버영역에서의 허가는 추적을 돌리고 끝점(EndPoint)을 생성할 수 있습니다.
데이터베이스 영역에서의 허가권은 테이블, 뷰, 저장프로시저, 함수, 서비스브로커,큐 등등과 같은 보안개체를 포함합니다.

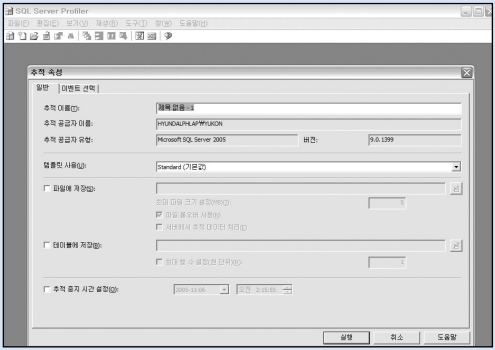
개체영역 허가는 테이블, 뷰, 프로시저 등등에 대한 변경을 포함합니다.
여기서 새로 등장하는 개념 impersonate이 등장합니다. 이것은 login X와 login Y가 있을 때 로그인 X에게 로그인 Y에 대한 IMPERSONATE 권한을 부여할 수 있습니다. 이렇게 되면 로그인 Y는 로그인 X의 역할로 명령어를 수행 할 수 있습니다.

```
USE AdventureWorks
Go
CREATE LOGIN Tom WITH Password =' 1234'
GO
CREATE USER Tom
GO
CREATE LOGIN Jelly WITH Password =' 5678'
GO
CREATE USER Jelly
GO

GRANT EXECUTE    ON          DATABASE::ADVENTUREWORKS TO Jelly
GRANT CONTROL    ON          OBJECT::PERSON.ADDRESS
TO Jelly
GRANT SELECT                 ON          SCHEMA::PERSON
TO Jelly
-- CONTROL 허가를 이용하면 허가를 부여받은 보안주체는 개체를 소유한 것 만큼의 허가를 받게됩니다.
-- 아래의 명령을 수행하기 전에 프로필러에 Jelly로 로그인을 해봅니다.
```




〈그림. 서비스 지향 기술구조〉

```
USE master
Go
-- Jelly에게 추적권한을 허가합니다.
GRANT ALTER TRACE TO Jelly
-- 프로필러에 다시 한번 로그인합니다.
```




〈그림 로그인 성공〉

--〉 로그인이 잘 되었습니다.

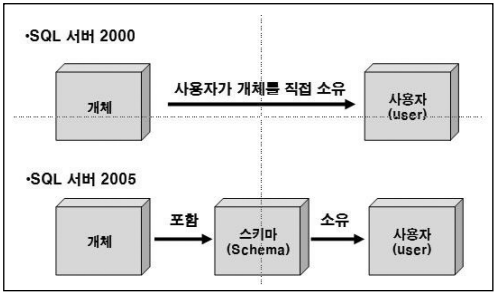
```
USE AdventureWorks;
GRANT IMPERSONATE ON USER::Jelly TO Tom
--Tom으로 로그인 했을 때 어떻게 보안이 수행해 오는지 살펴보자.
-- Tom으로 로그인
USE AdventureWorks
GO
SELECT * FROM person.address
GO
--메시지229, 수준14, 상태5, 줄1
--개체' Address' , 데이터베이스' adventureworks' , 스키마' Person' 에대한SELECT
사용권한이 거부되었습니다.

ALTER TABLE person.address ADD col1 INT null
GO
--메시지1088, 수준16, 상태13, 줄1
--개체"address" 이(가) 없거나 권한이 없으므로 이를 찾을 수 없습니다.

ALTER TABLE person.address DROP column col1
GO
--메시지1088, 수준16, 상태13, 줄1
--개체"address" 이(가) 없거나 권한이 없으므로 이를 찾을 수 없습니다.

EXEC dbo.uspGetEmployeeManagers 1;
GO
--메시지229, 수준14, 상태5, 프로시저uspGetEmployeeManagers, 줄1
--개체' uspGetEmployeeManagers' , 데이터베이스' adventureworks' , 스키마' dbo'
에대한EXECUTE 사용권한이 거부되었습니다.

--Jelly로 은폐변경을해보자.
```

```
EXECUTE AS USER = 'Jelly'
GO
SELECT User_name( )
---------------------------------
Jelly

-- 앞에서 수행한 쿼리를 재수행해봅시다.
SELECT * FROM person.address
GO
ALTER TABLE person.address ADD col1 INT null
GO
ALTER TABLE person.address DROP column col1
GO
EXEC dbo.uspGetEmployeeManagers 1;
GO
-- 모든쿼리가 잘 수행됩니다.
```

## 사용자와 스키마의 분리

2000에서는 사용자는 개체를 생성하면 그 소유권은 생성한 사용자에게 속했습니다. 예를 들어서 Ann이라는 사용자가 TableA를 생성하면 소유권은 Ann이 되었습니다. 이렇게 사용자는 개체를 만들어 내고 소유권도 가지게 됩니다. 만약 사용자 Ann을 지워야 한다면 어떻게 사용자를 지울 수 있을까요? 사용자를 지운다? 바로 지울 수 없습니다. 이유는 사용자를 지우려면 사용자가 가지고 있는 소유물의 소유권을 먼저 변경을 해야 하기 때문입니다.

스키마의 개념은 다음과 같습니다. 사용자가 개체를 만들지만 생성된 개체의 소유권은 스키마가 지니게 됩니다. 즉, Ann.TableA이 됩니다. 모양은 2000과 같지만 앞에 나온 Ann은 소유자가 아닌 스키마를 의미합니다. 이렇듯 스키마는 데이터베이스 개체를 네임스페이스로 구성하게 됩니다.




〈그림 스키마〉

한편 2005에서는 기본적으로 dbo라는 스키마를 가지고 있어서 2000에서 사용하던 dbo 사용자의 개체를 dbo 스키마로 변경하게 됩니다.

사용자를 생성할 때 다음과 같이 생성하며 기본 스키마를 지정합니다. 스키마는 생성되지 않은 스키마를 우선적으로 지정할 수 있습니다.

### ■ 스키마 관리하기

스키마 관리는 SQL Server Management Studio에 있는 Object Explorer나 CREATE, ALTER, DROP SCHEMA를 사용해서 수행합니다.

### ■ 스키마 생성하기

CREATE SCHEMA 문을 이용해서 새로운 스키마를 생성합니다. 간단히 한 줄로 스키마를 작성할 수 있습니다.

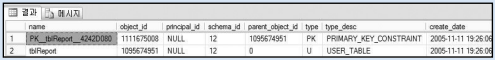
```
CREATE SCHEMA  Schema_A
```

CREATE SCHEMA 문의 문법은 아래와 같습니다.

```
CREATE SCHEMA schema_name_clause
    [ 〈 schema_element 〉 [ , ...n ] ]
〈schema_name_clause 〉 ::=
{    〈schema_name〉
    | AUTHORIZATION 〈 owner_name 〉
    | 〈 schema_name 〉 AUTHORIZATION 〈 owner_name 〉
}
〈schema_element 〉 ::=
{ table_definition | view_definition |
grant_statement| revoke_statement | deny statement }
예)
CREATE SCHEMA sales
    CREATE TABLE orders
    (OrderID INT, SalesPersonID INT, OrderDate DATETIME)
    GRANT SELECT ON orders TO John
GO
```

### [따라하기] 스키마 생성 및 스키마에 개체 생성하기

```
USE AdventureWorks
GO

CREATE SCHEMA Sample
GO

CREATE TABLE Sample.tblReport
(
        id_no INT PRIMARY key      NOT null      identity(1,1)
,       employee_name              varchar(10)
)
GO

INSERT INTO  Sample.tblReport DEFAULT values;
SELECT * FROM  Sample.tblReport
GO

SELECT NAME AS schemaowner FROM sys.database_principals
WHERE principal_id =
(SELECT principal_id FROM sys.schemas
WHERE NAME =  'Sample' )
GO
--결과: dbo

SELECT * FROM sys.all_objects WHERE schema_id =
( SELECT schema_id FROM sys.schemas WHERE NAME =' Sample' )
GO
```



결과 메시지

| | name | object_id | principal_id | schema_id | parent_object_id | type | type_desc | create_date |
|---|---|---|---|---|---|---|---|---|
| 1 | PK__tblReport__4242D080 | 1111675008 | NULL | 12 | 1095674951 | PK | PRIMARY_KEY_CONSTRAINT | 2005-11-11 19:26:06 |
| 2 | tblReport | 1095674951 | NULL | 12 | 0 | U | USER_TABLE | 2005-11-11 19:26:06 |

〈그림. 스키마 개체〉

```
DROP TABLE Sample.tblReport
GO
DROP SCHEMA Sample
GO
```

### ■ 기본 스키마 지정하기

개체를 생성하거나 개체를 수정할 때 기본 스키마를 지정할 수 있습니다. 특별히 스키마를 지정하지 않으면 기본 스키마는 dbo가 됩니다.

**[따라하기] 기본스키마 지정하기 및 스키마 이해하기.**

```
USE adventureWorks
GO
-- 로그인을 생성합니다.
CREATE login loginTemp WITH password ='1234'
-- 사용자를 생성하고 그 사용자의 기본 스키마를 sales로 합니다.
CREATE USER loginTemp WITH default_schema = sales;
-- sales.store 개체의 SELECT명령어 권한을 loginTemp에게 허가합니다.
GRANT SELECT ON object::sales.store TO loginTemp
GO

-- 사용자를 loginTemp로 전환합니다.
```

```
EXECUTE AS USER =' loginTemp'
GO

SELECT * FROM store
GO
SELECT * FROM sales.store
GO
-- 위의 두 개의 쿼리가 성공적으로 수행됩니다.

-- 사용자를 원위치(dbo)로 합니다.
REVERT
GO
-- 사용자 loginTemp의 기본 스키마를 production으로 합니다.
ALTER USER loginTemp WITH default_schema=production
GO

-- 사용자를 loginTemp로 전환합니다.
EXECUTE AS USER =' loginTemp'
GO
-- 아래의 쿼리를 수행하면 실패합니다.
-- 실패 이유는 loginTemp 사용자가 production 스키마에서 store를 찾습니다.
-- 하지만 production 스키마에는 존재하지 않습니다.
SELECT * FROM store
GO
메시지208, 수준16, 상태1, 줄2
개체이름' store' 이(가) 잘못되었습니다.

-- 아래의 쿼리는 성공합니다.
-- 성공이유는 loginTemp 사용자가 sales스키마에서 Store 개체를 찾았기 때문입니다.
```

```
SELECT * FROM sales.store
GO
-- 사용자를 환원합니다.
REVERT
Go
```

■ 스키마 변경하기

스키마의 소유권을 변경하게 되면 스키마에게 부여되었던 모든 개체와 허가는 제거됩니다.

ALTER SCHEMA schema_name TRANSFER object_name

예)

ALTER SCHEMA HumanResources TRANSFER Person.Address

설명: Person스키마에 있는 address 테이블을 HumanResources 스키마로 수정합니다.

■ 스키마 제거하기

DROP SCHEMA 명령어를 이용해서 스키마를 제거합니다. 스키마를 제거하기 전에 스키마가 가지고 있는 모든 개체를 우선적으로 제거해야 합니다.

```
DROP SCHEMA schema_name
```

### [따라하기] 스키마 제거하기

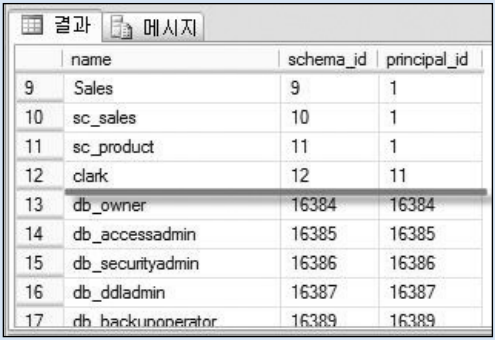
```
--스키마 sales를 제거합니다
DROP SCHEMA sales
```

***알림***

*sp_addlogin 과 sp_adduser 명령어는 앞으로 스크립트에서 사용하지 않을 것을 권장합니다.*
*위의 프로시저는 CREATE LOGIN 과 CREATE USER로 대치되어 사용됩니다.*
*2005에서 sp_addlogin과 sp_adduser는 사용할 수 있으나 기본 스키마를 지정할 수 없습니다.*
*sp_adduser를 이용하여 사용자를 추가하면 기본 스키마는 사용자 이름과 같은 스키마를 가지게 됩니다.*

### [따라하기] 로그인 생성

```
-- 로그인 생성
sp_addlogin 'clark' , 'adventureWorks'

USE AdventureWorks
-- 사용자 계정생성
sp_adduser 'clark'
-- 스키마 테이블에서 확인
select * from sys.schemas
-- 사용자와 같은 스키마로 생성이 되어 있는 것을 볼 수 있음
```



결과 | 메시지

| | name | schema_id | principal_id |
|---|---|---|---|
| 9 | Sales | 9 | 1 |
| 10 | sc_sales | 10 | 1 |
| 11 | sc_product | 11 | 1 |
| 12 | clark | 12 | 11 |
| 13 | db_owner | 16384 | 16384 |
| 14 | db_accessadmin | 16385 | 16385 |
| 15 | db_securityadmin | 16386 | 16386 |
| 16 | db_ddladmin | 16387 | 16387 |
| 17 | db_backupoperator | 16389 | 16389 |

*〈그림 스키마와 로그인〉*

## 메타데이터에 대한 제한적인 접근

2000에서는 모든 사용자는 메타데이터(시스템테이블)에 접근이 가능했습니다. 그리고, 사용자의 권한에 관계 없이 모든 행을 반환했습니다. 2005에서는 메타데이터(시스템테이블)에 대해서 직접적으로 접근을 금지 하며 하나의 뷰 형식으로 그 값을 반환합니다. 사용자(User)에게 owner가 아니거나 메타데이터에 대한 열람 할 수 있는 권한이 없다면 행을 반환하지 않습니다.

2005에서는 새로운 허가 VIEW DEFINITION을 제공합니다. 이것은 사용자에게 메타데이터를 볼 수 있도록 권한을 부여합니다.

[따라하기]

```
USE AdventureWorks
GO

CREATE LOGIN  Ally WITH password=' 12345'
GO

CREATE USER allyU  FOR LOGIN Ally
GO

EXECUTE AS USER =' allyU'
go

EXEC sp_helptext  'dbo.uspLogError'
go

메시지15009, 수준16, 상태1, 프로시저sp_helptext, 줄54
데이터베이스' adventureworks' 에개체' dbo.uspLogError' 이(가) 없거나이작업에적
합하지않습니다.

SELECT object_definition(object_id( 'dbo.uspLogError' ))
go
NULL

SELECT * FROM sys.objects WHERE type =' U'

SELECT * FROM information_schema.tables;
go
```

```
--실행컨텍스트를마지막EXECUTE AS 문의호출자로다시전환합니다.
REVERT;
go

--사용자allyU에게각각의권한을부여합니다.
GRANT VIEW definition ON OBJECT::dbo.uspLogError TO allyU
GRANT VIEW definition ON SCHEMA::HumanResources TO allyU

--위의작업을다시수행합니다.
-- 이전과는달리결과값이잘반환되는것을볼수있습니다.
EXECUTE AS USER =' allyU'
go

EXEC sp_helptext  'dbo.uspLogError'
go

SELECT object_definition(object_id( 'dbo.uspLogError' ))
go

SELECT * FROM sys.objects WHERE type =' U'

SELECT * FROM information_schema.tables;
go

REVERT;
go
```

## 모듈 실행 컨텍스트

SQL Server 2005 에서는 모듈 실행 컨텍스트라는 개념을 도입했습니다. 이것은 저장프로시저, 함수 트리거와 같은 모듈에 대한 실행을 규정합니다.
EXECUTE AS는 함수, 프로시저, 큐 및 트리거와 같은 사용자 정의 모듈의 실행 컨텍스트를 정의하는 데 사용될 수 있습니다. 예를 들어 실행 컨텍스트는 모듈 호출자에서 모듈 소유자로 또는 지정된 사용자로 전환될 수 있습니다. 이전 버전의 SQL Server?에서 이러한 모듈은 항상 모듈 호출자의 컨텍스트에서 실행되었습니다.

모듈이 실행되는 컨텍스트를 지정해서 SQL Server 2005 데이터베이스 엔진 사용자 계정을 제어하면 모듈에서 참조하는 개체에 대한 사용 권한을 검사할 수 있습니다. 이렇게 하면 사용자 정의 모듈 및 해당 모듈에서 참조하는 개체 사이에 있는 개체 체인에서 좀 더 유연하게 사용 권한 관리를 제어할 수 있습니다. 모듈 사용자는 해당 모듈의 실행 권한만 필요하며, 참조되는 개체에 대한 명시적 사용 권한은 필요하지 않습니다. 모듈이 실행 중인 사용자만이 모듈에서 액세스한 개체에 대한 사용 권한을 가져야 합니다.

**[따라하기]**

```
CREATE PROC GetOrders
With EXECUTE AS {CALLER |SELF | OWNER | 'user_name' }
AS
{수행 문장}
```

위의 인수가 의미하는 것은 다음과 같습니다.

| 인수 | 설명 |
|---|---|
| CALLER | 모듈 내부의 문이 모듈 호출자의 컨텍스트에서 실행되도록 지정합니다. 모듈을 실행하는 사용자는 모듈 자체에 대한 알맞은 사용 권한뿐만 아니라 모듈에서 참조하는 모든 데이터베이스 개체에 대한 사용 권한도 갖고 있어야 합니다.<br><br>CALLER는 큐를 제외한 모든 모듈의 기본값이며 이것은 SQL Server 2000 동작과 동일합니다.<br><br>CREATE QUEUE 또는 ALTER QUEUE 문에서는 CALLER를 지정할 수 없습니다. |
| SELF | EXECUTE AS SELF는 EXECUTE AS user_name과 동일합니다. 여기에서 지정된 사용자는 모듈을 만들거나 변경하는 사용자입니다. 모듈을 만들거나 수정하는 사용자의 실제 ID는sys.sql_modules 또는 sys.service_queues 카탈로그 뷰의 execute_as_principal_id 열에 저장됩니다. SELF는 큐의 기본값입니다.<br><br>sys.service_queues 카탈로그 뷰에서 execute_as_principal_id의 사용자 ID를 변경하려면 ALTER QUEUE 문에서 명시적으로 EXECUTE AS를 설정해야 합니다 |
| OWNER | 모듈 내부의 문이 현재 모듈 소유자의 컨텍스트에서 실행되도록 지정합니다. 모듈에 지정된 소유자가 없으면 이 모듈의 스키마 소유자를 사용합니다. DDL 트리거에 대해서는 OWNER를 지정할 수 없습니다. |

**[따라하기] 사용자 Alice에게 실행권한이 있는 프로시저 생성하기**

```
CREATE PROC Getorders
WITH EXECUTE AS 'Alice'
AS
SELECT * FROM sales.orders
```

위의 [따라하기]는 저장 프로시저 Getorders는 Alice가 수행하는 것과 같은 사용권을 가집니다.

**[따라하기] AdventureWorks 데이터베이스에서 소유자 Bob이 sales.orders 라는 테이블을 생성했고 Alice 에게 이 테이블에 대한 SELECT 권한을 부여했습니다. 반면 Bob은 Eve 에게 어떤 권한도 주지 않았습니다. Eve는 다음과 같이 저장 프로시저 sales.getorders을 만들었습니다.**

```
CREATE PROC sales.getorders
WITH EXECUTE AS SELF
AS
SELECT * FROM sales.orders
```

Eve 가 위의 저장프로시저를 수행 할 수 있을까요? 아니면 없을까요?

EXECUTE AS SELF라는 구절은 이 프로시저를 수행하는 사용자에게 권한을 부여한다는 의미입니다. 즉, Eve는 저장프로시저에 테이블에 접근할 수 있는 권한이나 소유권 깨짐 현상이 없어도 수행을 할 수 있게 되었습니다.

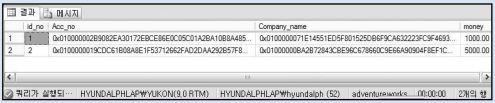
## 암호화 지원

이전 버전에서는 데이터를 암호화하고 암호를 해독하는데 있어서 자체 개발이나 써드파티 벤더의 솔루션을 이용해야 했습니다. 이런 문제점을 해결하기 위해서 2005에서는 SQL Server 자체적으로 지원하기로 했습니다.
EncryptByKey, EncryptByAsmKey, EncryptByCert, EncryptByPassPhase 등의 함수를 이용하여 데이터를 암호화 하고 암호를 풀어냅니다.

### [따라하기] 암호화와 암호화된 데이터 읽기

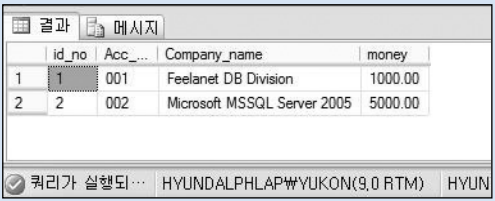
```
USE AdventureWorks
go
CREATE TABLE dbo.tAccount
(
        id_no           int             identity(1,1)   NOT NULL PRIMARY key
,       Acc_no          varbinary(100) NOT NULL
,       Company_name                    varbinary(512) NOT null
,       money Money                                     NOT null
)
go

INSERT INTO dbo.tAccount values
(
        EncryptByPassPhrase( 'asdfg_1' ,  '001' )
,       EncryptByPassPhrase( 'asdfg_2' ,  'Feelanet DB Division' )
,       1000
)
-- 암호화하여데이터삽입하기
INSERT INTO dbo.tAccount values
(

        EncryptByPassPhrase( 'asdfg_1' ,  '002' )

,       EncryptByPassPhrase( 'asdfg_2' ,  'Microsoft MSSQL Server 2005' )
```

```
,       5000
)

-- 데이터삽입한결과
SELECT * FROM dbo.tAccount
GO
```




〈그림. 암호화된 데이터〉

```
-- 데이터를읽어오기결과
SELECT  id_no
,       CONVERT(varchar(15),
        DecryptByPassPhrase( 'asdfg_1' , Acc_no)) AS Acc_no
,       CONVERT(varchar(255),
        DecryptByPassPhrase( 'asdfg_2' , Company_name)) AS Company_name
,       money
FROM dbo.tAccount
GO
```




〈그림. 암호화된 데이터〉

```
DROP TABLE dbo.tAccount
GO
```

## SQL Server 2005 데이터베이스 엔진의 보안 값 설정

2005서버에서 보안에 관련된 서버구성 값이 추가되었습니다.

| Sp_configure의 옵션 | 설명 | 기본값 |
|---|---|---|
| CLR enabled | CLR enabled 옵션을 사용하여 Microsoft SQL Server에서 사용자 어셈블리를 실행할 수 있는지 여부를 지정합니다. | Off |
| Xp_cmdshell | 시스템 관리자가 시스템에서 xp_cmdshell 확장 저장 프로시저를 실행할 수 있는지 여부를 제어할 수 있도록 하는 서버 구성 옵션입니다. | Off |
| Web Assistant Stored Procedures | 서버에서 웹 길잡이 프로시저를 설정할 수 있습니다. | Off |
| Ad hoc Distributed Queries | 이 옵션을 0이 아닌 값으로 설정하면 SQL Server 에서 OLE DB 공급자에 대해 OPENROWSET 및 OPENDATASOURCE 함수를 통한 임의 액세스가 허용되지 않습니다. | Off |
| Database Mail XPs | 서버에서 데이터베이스 메일을 활성화할 수 있습니다 | Off |
| SQL Mail XPs enabled | 서버에서 SQL 메일을 활성화할 수 있습니다. | Off |
| OLE automation procedures | OLE 자동화 개체가 Transact-SQL 일괄 처리 내에서 인스턴스화될 수 있는지 여부를 지정할 수 있습니다. | Off |
| SMO and DMO XPs | 서버에서 SQL 메일을 활성화할 수 있습니다. | On |
| Remote admin connections | 관리자 전용연결(DAC)을 허용합니다. | Off |

| | | |
|---|---|---|
| Agent XPs | 서버에서 SQL Server 에이전트 확장 저장 프로시저를 설정할 수 있습니다. 이 옵션을 해제하면 SQL Server Management Studio 개체 탐색기에서 SQL Server 에이전트 노드를 사용할 수 없습니다. SQL Server 에이전트 서비스를 시작할 때 이 확장 저장 프로시저가 자동으로 설정됩니다. | On |