Fast track to SYBASE IQ

한국사이베이스
Pre-Sales BI Part

ver 2.9
2003/10/09
소병각

SYBASE IQ Overview

Sybase RDBMS

- **Sybase가 제공하는 RDBMS는 각 업무의 규모와 성격에 따라 크게 아래와 같이 3개로 구분되어집니다.**

  ● **Adaptive Server Anywhere (ASA): 개인업무용 최소형 RDBMS이고 ANSI 표준 SQL를 지원하며 검증된 제품입니다. SYBASE IQ의 일부분으로 사용되고 있으며 SYBASE IQ에서는 Catalog, Query parser, Connectivity를 담당하고 있습니다**

  ● **Adaptive Server Enterprise (ASE) : 범용 RDBMS이며 특히 OLTP 업무에 탁월한 효과를 발휘합니다. DW 솔루션에서 OLAP의 Repository 영역으로도 사용 가능합니다.**

  ● **SYBASE IQ : DW/DSS/Data Mart 업무를 위해 고안되어진 전용 RDBMS 이며 ASA가 포함되어 있습니다. 단 여기에 포함된 ASA 영역은 시스템을 위한 영역입니다.**

SYBASE IQ 는……

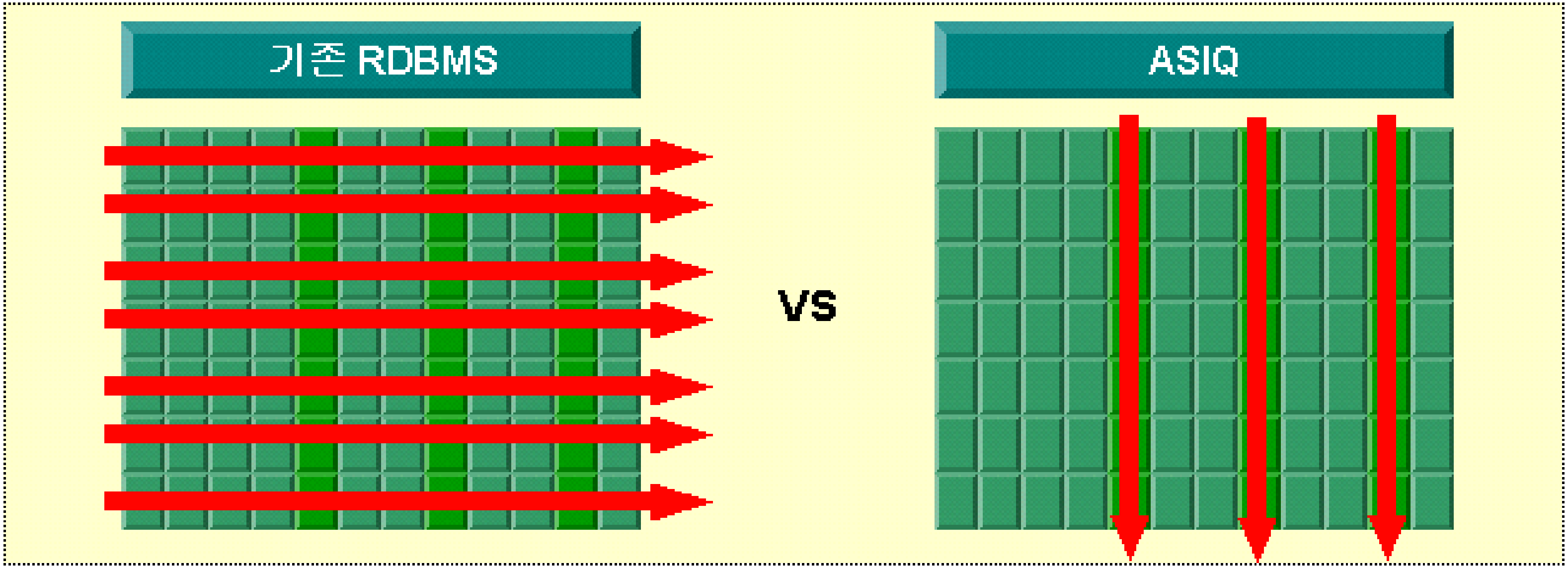
- 기존 **RDBMS**는 **OLTP**와 **DSS**가 혼재된 데이터베이스 엔진인데 반해 **SYBASE IQ** 데이터 웨어하우스 전용 **RDBMS**로 인덱싱이나 액세스 방식에서 데이터 웨어하우스에 맞는 새로운 기법으로 구현된 **RDBMS**로 성능**,** 유연성**,** 경제성 측면에서 가장 탁월한 데이터베이스 입니다

- 기존 **RDBMS**는 대량의 데이터에 대한 질의 특히 조인에 대한 성능을 높이기 위해 데이터베이스 스키마를 비정규화 즉 **Star-Schema** 또는 **Snowflake-Schema**로 디자인 하기를 권장 하기도 하고 파티션 기법의 데이터 분산 디자인에 의해 **ER** 모델을 권장하기도 하지만 **SYBASE IQ**의 새로운 기술은 데이터베이스 스키마에 대해 어떠한 제약도 두지 않습니다

- **RDBMS**가 처리 능력을 높이기 위해 단일 질의 작업을 **MPP**와 같은 병렬 노드에 분담시키는 방법으로 대용량 데이터를 처리 하는 기법을 이용하는데 이는 비용에 비해 성능에 문제가 있습니다

SYBASE IQ 특징-column-wise data 구조

- 기존 RDBMS는 데이터 액세스 방식이 레코드 레벨로 되어 있고 DW는 일정 주기로 데이터가 로드 되며 로드된 후에 액세스 되는 형태로 이용되어 집니다. 이때 DW에 사용되는 질의는 레코드의 모든 항목이 필요한 것이 아니라 특정 몇 개의 컬럼만 필요로 합니다. 따라서 레코드 레벨의 데이터 액세스 방식은 불필요한 I/O를 유발 시키므로 DW를 효과적으로 지원할 수 없습니다 이와 같은 I/O의 증가를 피하기 위해 MPP와 같은 병렬 시스템을 이용 하는데 이는 여러 시스템에 I/O를 분산시켜 성능을 향상 시키기 위한 것인데 결국 I/O의 총량은 변화가 없으며 비용 대비 성능이 탁월하지 않은 것으로 나타났습니다. SYBASE IQ는 I/O 자체를 줄일 수 있는 새로운 데이터 액세스 기법을 제공 합니다. 필요한 컬럼만 액세스 하여 한번 I/O에 대단히 많은 양의 데이터를 읽어 들일 수 있습니다

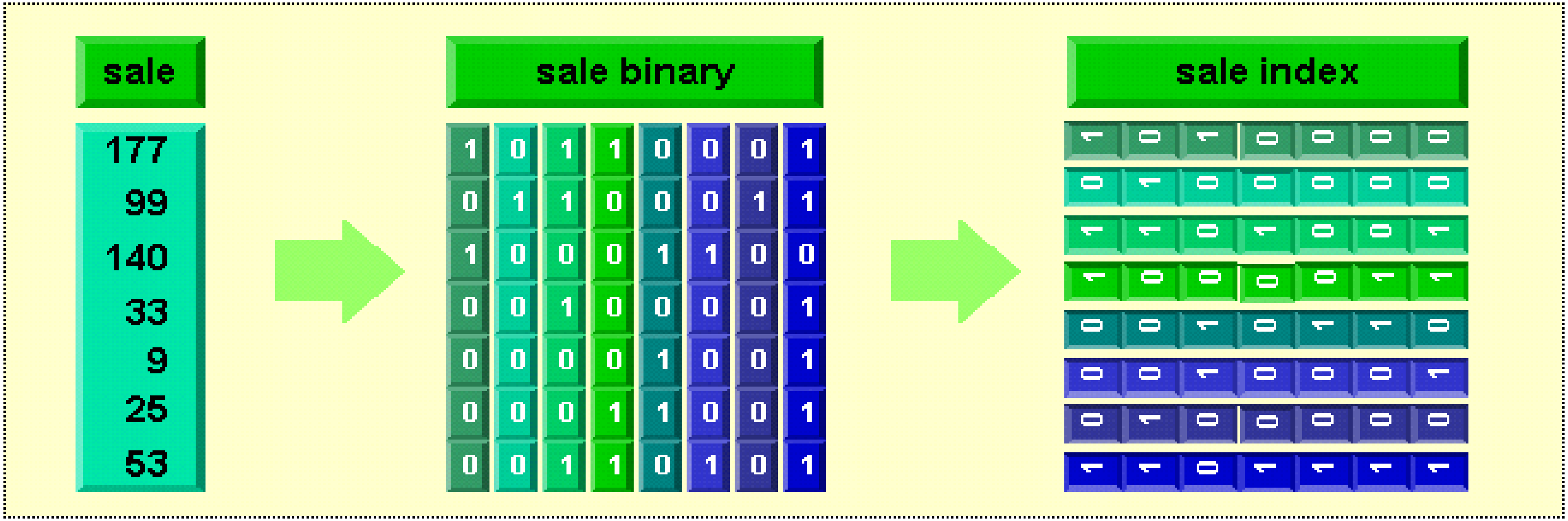


SYBASE IQ 특징-bit-wise index 구조

- 칼럼의 카디날리티에 해당하는 수를 **bit mask** 형태로 표현하며 이렇게 표현된 각 **bit**는 개별적으로 따로 저장되어 인덱스가 됩니다

- 이렇게 **bit** 단위로 저장된 각각의 인덱스는 서로 다른 압축저장 기법을 사용하며 또한 서로 다른 압축률을 제공합니다.

- 압축률은 데이터의 성격 및 데이터베이스를 생성할 때 사용하는 인자인 페이지 크기에 따라 달라집니다.



sale
177
99
140
33
9
25
53

sale binary							
1	0	1	1	0	0	0	1
0	1	1	0	0	0	1	1
1	0	0	0	1	1	0	0
0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	1
0	0	1	1	0	1	0	1

sale index						
1	0	1	0	0	0	0
0	1	0	0	0	0	0
1	1	0	1	0	0	1
1	0	0	0	0	1	1
0	0	1	0	1	1	0
0	0	1	0	0	0	1
0	1	0	0	0	0	0
1	1	0	1	1	1	1

SYBASE IQ를 이런 용도로는……

- **SYBASE IQ**의 **Lock,** 트랜잭션 메카니즘등 모든 구조가 **DW** 시스템에 적합하도록 디자인이 되어 있으므로 다음과 같은 업무를 위해서 사용하지 마십시오**.** 아래에 언급되고 있는 업무의 특성들은 소량의 데이터를 기반으로 한건 한건의 데이터 처리를 위주로 하는 업무입니다**.**
  - **OLTP**
  - **Real time update**
- 위와 같은 필요성이 있는 업무는 같이 제공되는 **ASE**라는 **RDBMS**로 이동하여 사용하시기 바랍니다**.**

SYBASE IQ Client 설정하기

SYBASE IQ Client

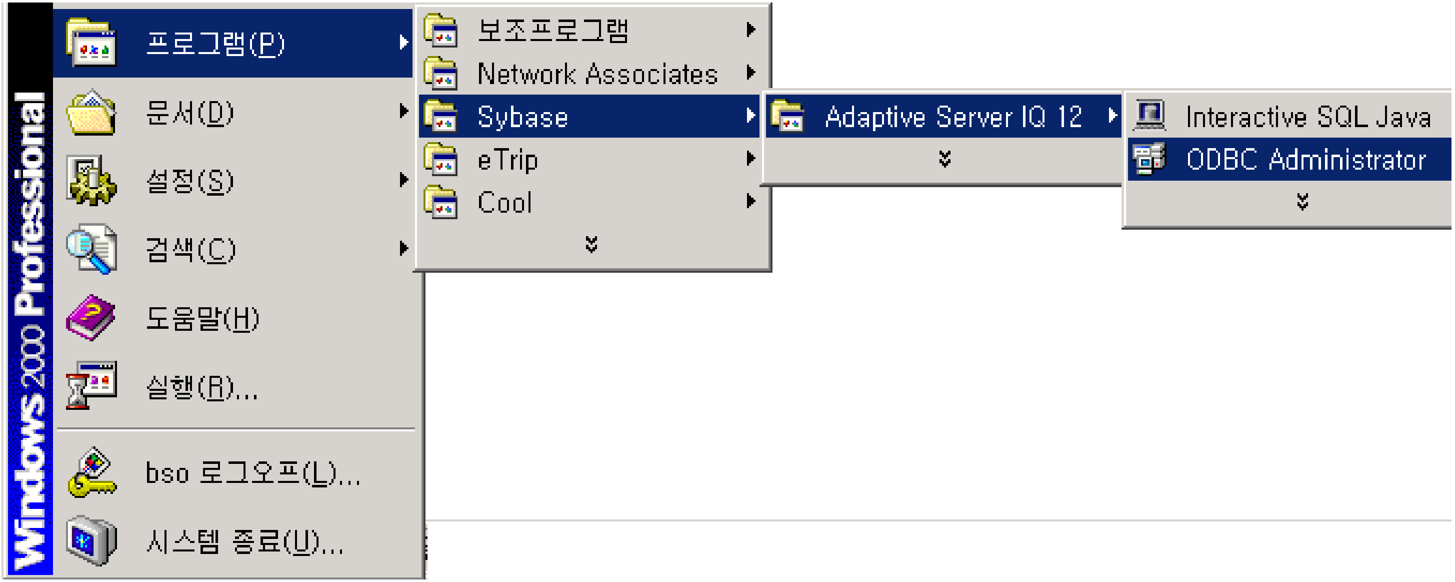
- ODBC를 지원하는 Client : SYBASE IQ에 최적화 된 API 방법이며 각각 UNIX용과 Window용이 있습니다.
  - DBISQLC(Interactive SQL Classic) : SYBASE IQ에 기본적으로 제공되는 SQL Editor
  - 그 외 3rd party 제품

- OCDK를 지원하는 Client : SYBASE IQ는 Native Driver인 OCDK를 지원하지만 ODBC 방법과 여러 가지 기본 설정 값이 달라 원하지 않는 부작용이 존재하기 때문에 충분히 숙지 후에 사용하시기 바랍니다.
  - isql : ASE에 기본적으로 제공되는 명령어 대화방식의 SQL Editor
  - SQL Advantage : ASE에 기본적으로 제공되는 GUI 대화방식의 SQL Editor
  - 그 외 3rd party 제품

- JDBC를 지원하는 Client : Sybase의 JDBC인 Jconnect를 지원하는 API 입니다.
  - DBISQL(Interactive SQL JAVA) : SYBASE IQ에 기본적으로 제공되는 SQL Editor 이며 지원 API에 따라 OCDK (Jconnect)와 ODBC (JDBC-ODBC Bridge)를 선택할 수 있습니다.
  - 3rd party 제품

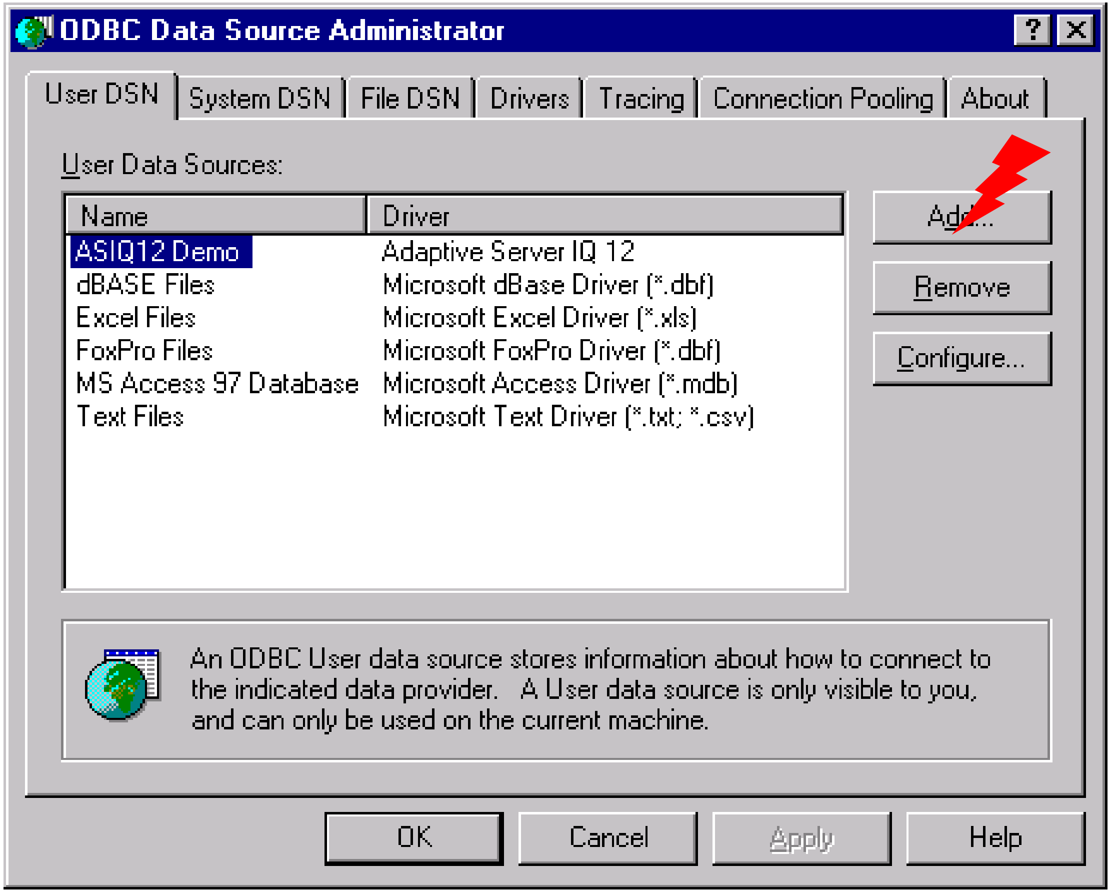
ODBC 설정 – win client

- ODBC Administrator를 실행합니다



ODBC 설정 – win client

- **Add 버튼을 클릭하여 새로운 data source를 생성합니다**



User Data Sources:

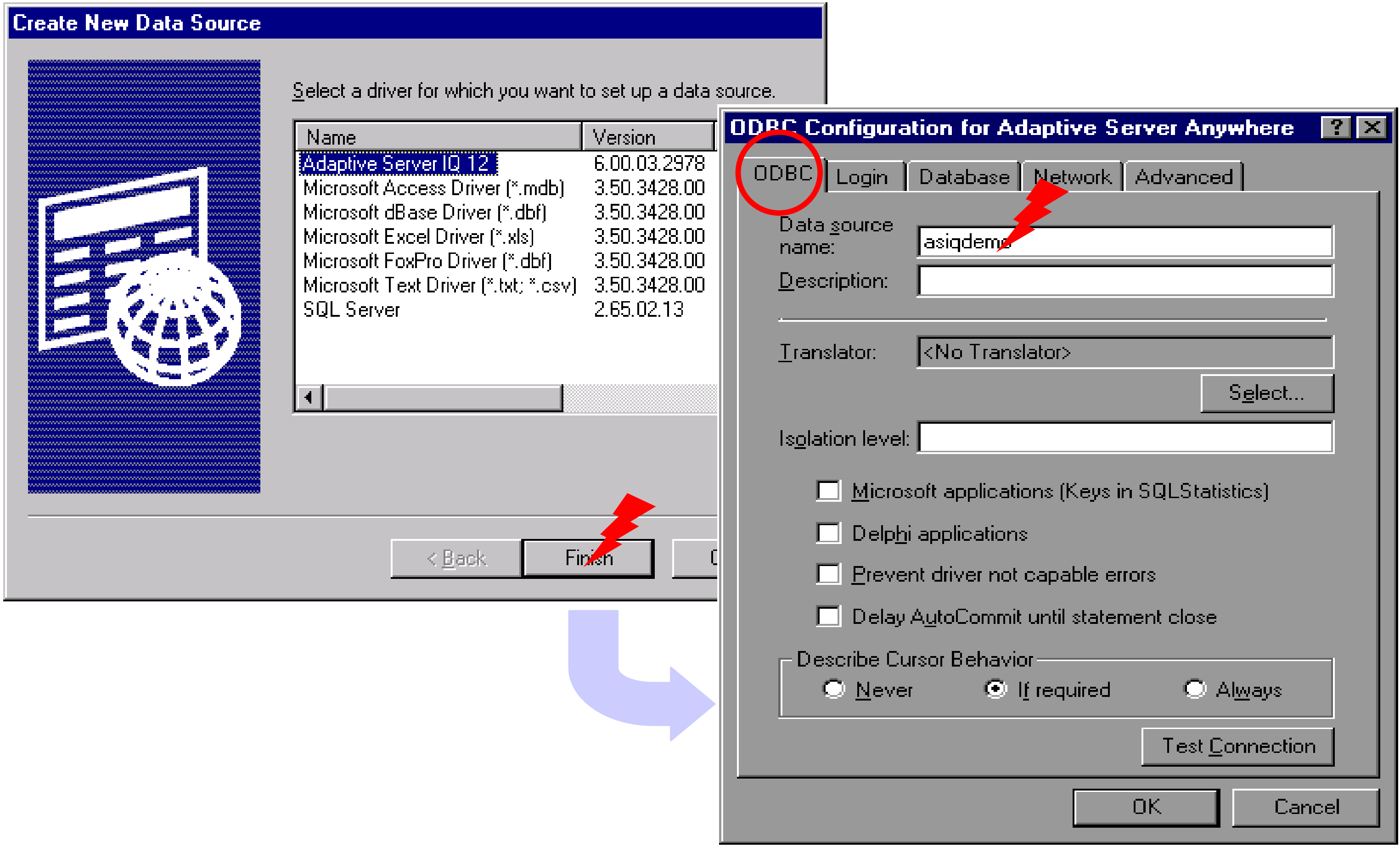
Name	Driver
ASIQ12 Demo	Adaptive Server IQ 12
dBASE Files	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files	Microsoft FoxPro Driver (*.dbf)
MS Access 97 Database	Microsoft Access Driver (*.mdb)
Text Files	Microsoft Text Driver (*.txt; *.csv)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

- **단, SYBASE IQ Server는 Unix Server에 존재하는 것으로 가정합니다.**

ODBC 설정 – win client

- **Adaptive Server IQ 12 드라이버를 선택하고 Finish 버튼을 클릭합니다**
- **ODBC 탭을 선택하고 Data source name 필드에 적절한 이름을 입력합니다**



Create New Data Source

Select a driver for which you want to set up a data source.

Name	Version
Adaptive Server IQ 12	6.00.03.2978
Microsoft Access Driver (*.mdb)	3.50.3428.00
Microsoft dBase Driver (*.dbf)	3.50.3428.00
Microsoft Excel Driver (*.xls)	3.50.3428.00
Microsoft FoxPro Driver (*.dbf)	3.50.3428.00
Microsoft Text Driver (*.txt; *.csv)	3.50.3428.00
SQL Server	2.65.02.13

ODBC Configuration for Adaptive Server Anywhere

Data source name: asiqdemo

Description:

Translator: <No Translator>

Isolation level:

□ Microsoft applications (Keys in SQLStatistics)

□ Delphi applications

□ Prevent driver not capable errors

□ Delay AutoCommit until statement close

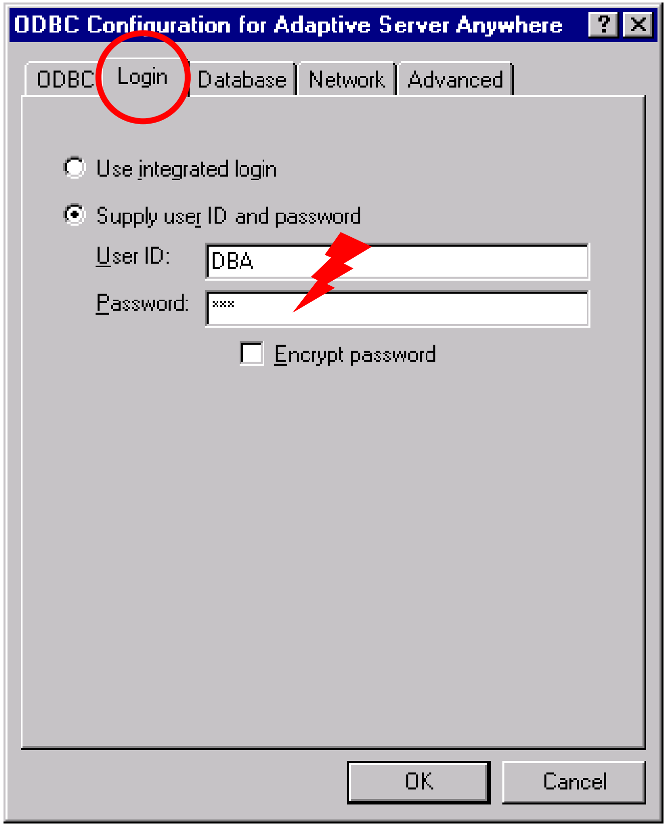
Describe Cursor Behavior

○ Never ● If required ○ Always

ODBC 설정 – win client

- Login 탭을 선택하고 User ID와 Password의 필드를 입력합니다

- Database 탭을 선택하고 Server name과 Database name 필드를 입력합니다



ODBC Configuration for Adaptive Server Anywhere

ODBC | Login | Database | Network | Advanced

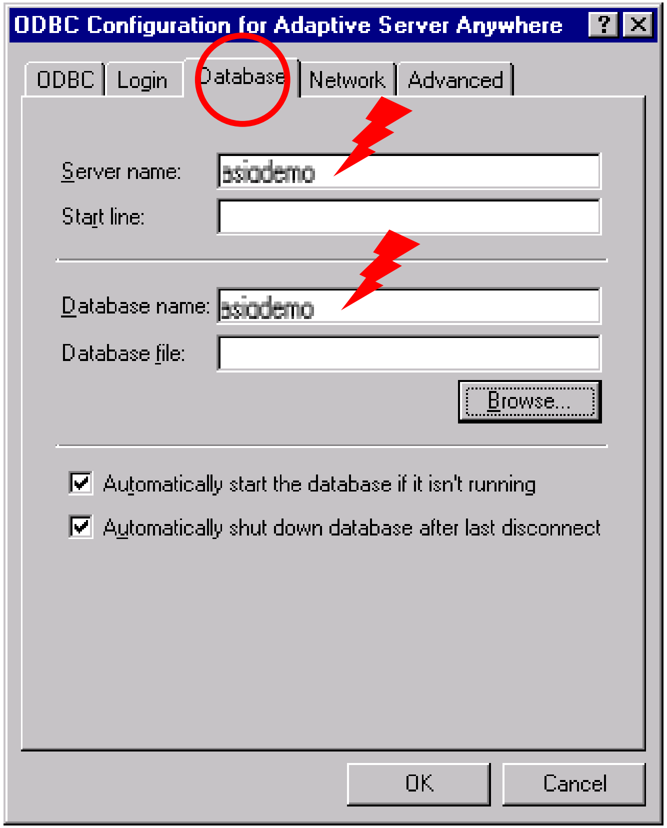
○ Use integrated login

● Supply user ID and password

User ID: DBA

Password: ***

□ Encrypt password



ODBC Configuration for Adaptive Server Anywhere

ODBC | Login | Database | Network | Advanced

Server name: asiodemo

Start line:

Database name: asiodemo

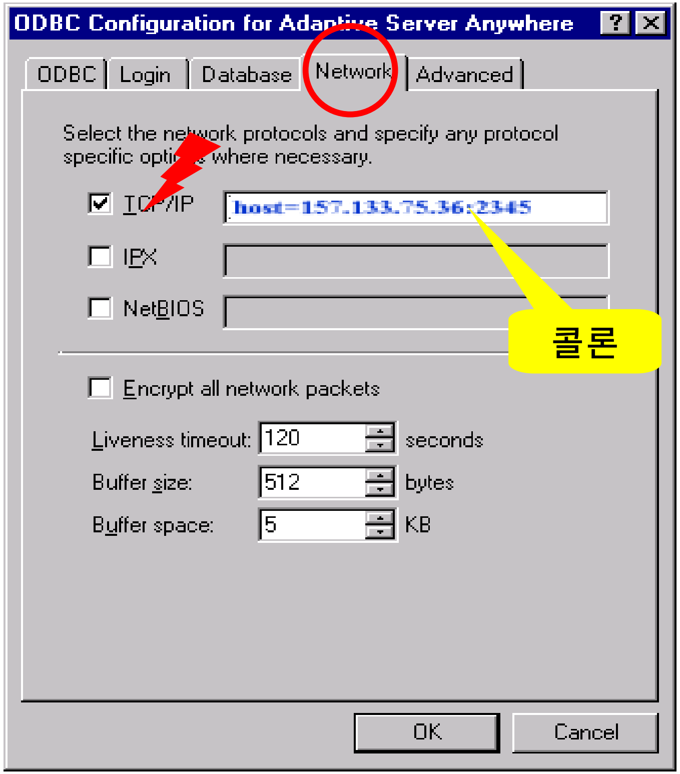
Database file:

☑ Automatically start the database if it isn't running

☑ Automatically shut down database after last disconnect

ODBC 설정 – win client

- **Network 탭을 선택하고 TCP/IP 체크박스를 클릭하고 SYBASE IQ 서버의 IP 주소와 port 번호를 콜론(주의:세미콜론 아님)으로 분리하여 입력합니다.**
- **ODBC 탭을 선택하고 Test connection 버튼을 클릭합니다.(단, 이 기능은 Win98과 같은 클라이언트에서는 되지 않고 WinNT와 같은 서버 계열에서만 실행됩니다.)**



ODBC Configuration for Adaptive Server Anywhere

Select the network protocols and specify any protocol specific options where necessary.

☑ TCP/IP host=157.133.75.36:2345

□ IPX

□ NetBIOS

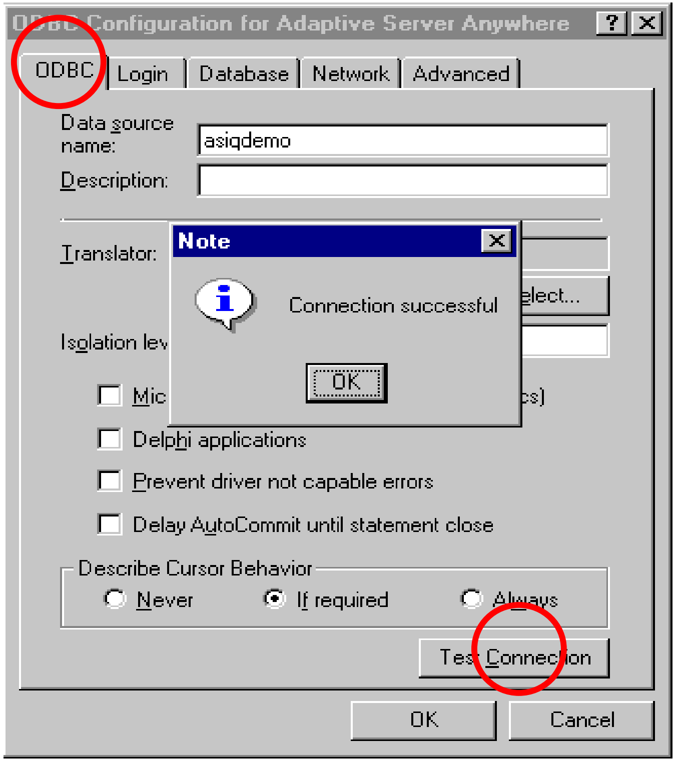
콜론

□ Encrypt all network packets

Liveness timeout: 120 seconds

Buffer size: 512 bytes

Buffer space: 5 KB



ODBC Configuration for Adaptive Server Anywhere

Data source name: asiqdemo

Description:

Translator:

Isolation lev

□ Mic ...cs)

□ Delphi applications

□ Prevent driver not capable errors

□ Delay AutoCommit until statement close

Describe Cursor Behavior ○ Never ● If required ○ Always

Note

Connection successful

ODBC 설정 – unix client

- **dbdsn 유틸리티 이용**

% $SYBASE/ASIQ-12_5/bin/dbdsn –c “ENG=asiqdemo;DBN=asiqdemo;UID=dba; PWD=SQL;CommLinks=tcpip{host=157.133.75.36;port=2345}” –w asiqdemo

위 명령어가 정상적으로 수행되었다면 **$SYBASE/.odbc.ini** 파일에 다음과 같은 내용이 들어 있습니다. 단, **ODBCINI**라는 환경변수가 특별히 따로 설정되었다면 설정된 디렉토리의 **.odbc.ini** 라는 파일에 생성됩니다. 또한 **.odbc.ini** 라는 파일은 반드시 존재해야 합니다.

```
[asiqdemo]                                        // data source name
Userid=dba                                        // user id
Password=SQL                                      // password
DatabaseName=asiqdemo                             // IQ database name
ServerName=asiqdemo                               // IQ engine name
CommLinks=tcpip{host=157.133.75.36;port=2345}     // protocol, IP, port
```

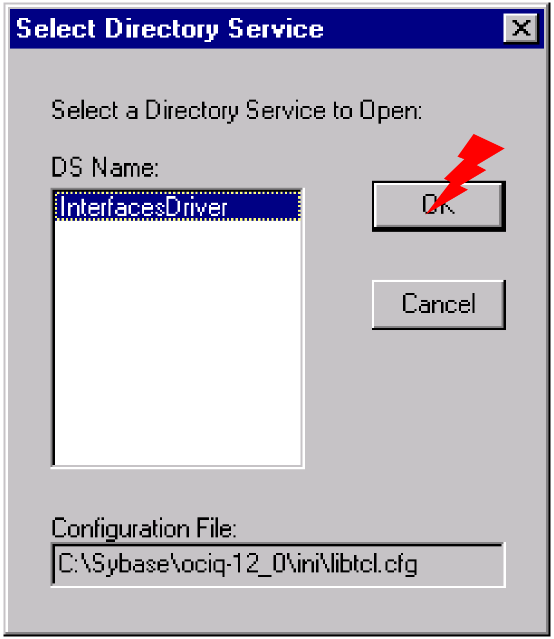
- 수동으로 편집

만약 **ODBCINI** 라는 환경변수가 설정되어 있다면 **$ODBCINI/.odbc.ini** 파일을, 설정되어 있지 않다면 **$SYBASE/.odbc.ini** 파일을 **vi** 와 같은 편집기를 사용하여 편집하면 됩니다.

JDBC/OCDK 설정 – win client

- **DSEDIT라는 유틸리티를 실행하고 OK 버튼을 클릭합니다. 단, SYBASE IQ 12.5 이상에서는 OCDK 모듈이 설치되지 않으므로 별도로 설치해야 합니다.**
  start → programs → sybase → Dsedit Utility

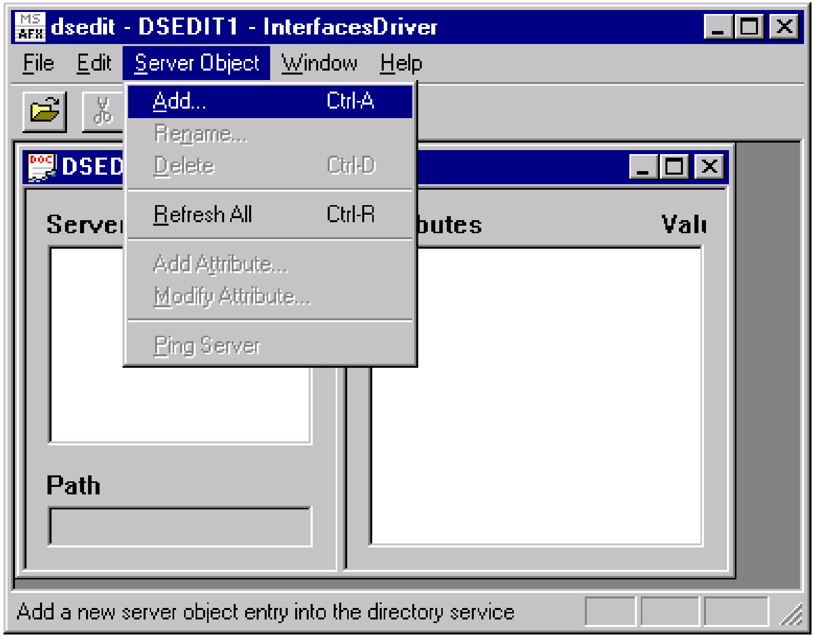
- **Server를 하나 추가시킵니다.**
  Server Object → Add……



Select a Directory Service to Open:

DS Name: InterfacesDriver

Configuration File: C:\Sybase\ocig-12_0\ini\libtcl.cfg

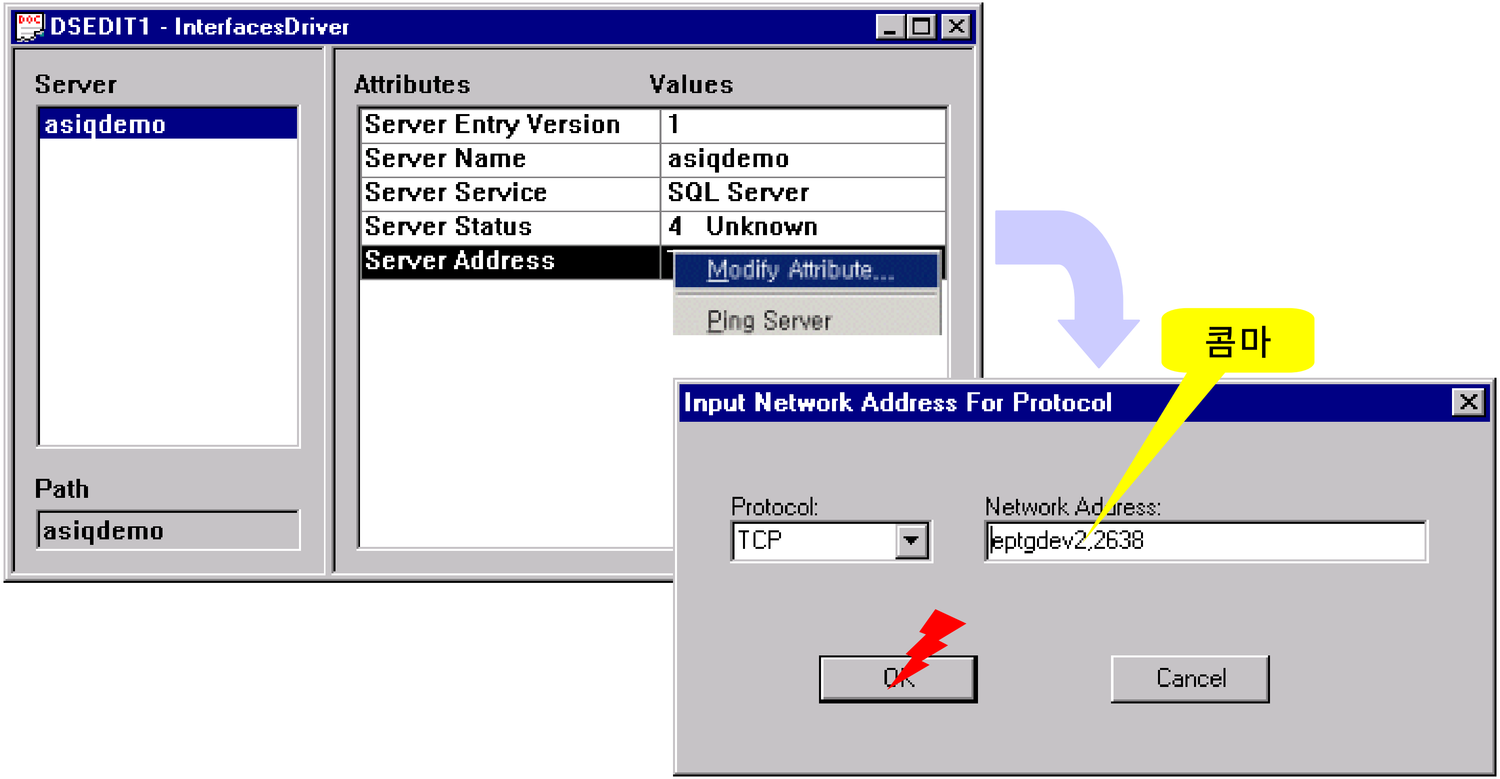


Add a new server object entry into the directory service

JDBC/OCDK 설정 – win client

- 등록된 서버의 IP 주소와 port 번호를 입력합니다
  Server Object → Modify Attribute

- OK 버튼을 클릭합니다



DSEDIT1 - InterfacesDriver

Server: asiqdemo

Path: asiqdemo

Attributes	Values
Server Entry Version	1
Server Name	asiqdemo
Server Service	SQL Server
Server Status	4 Unknown
Server Address	

Input Network Address For Protocol

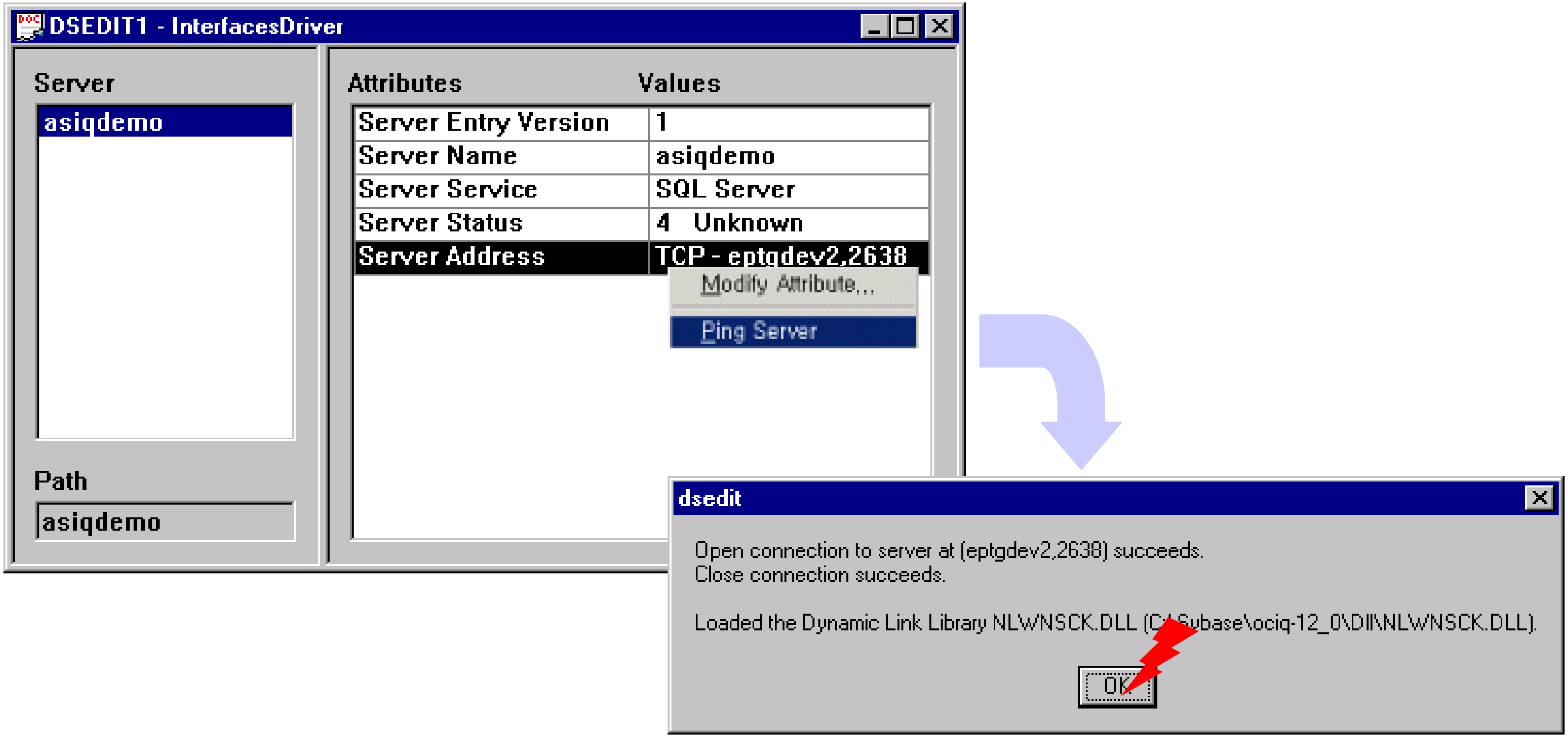
Protocol: TCP

Network Address: eptgdev2,2638

JDBC/OCDK 설정 – win client

- **SYBASE IQ** 서버와 연결 테스트를 실행합니다
  Server Object → Ping Server

- 아래와 같은 연결 성공 메시지가 보이면 종료합니다.



DSEDIT1 - InterfacesDriver

Server

asiqdemo

Path

asiqdemo

Attributes	Values
Server Entry Version	1
Server Name	asiqdemo
Server Service	SQL Server
Server Status	4 Unknown
Server Address	TCP - eptgdev2,2638

dsedit

Open connection to server at (eptgdev2,2638) succeeds.
Close connection succeeds.

Loaded the Dynamic Link Library NLWNSCK.DLL (C:\Sybase\ociq-12_0\Dll\NLWNSCK.DLL).

JDBC/OCDK 설정 – unix client

- **dscp 유틸리티를 이용**
  % $SYBASE/OCS-12_5/bin/dscp

```
>> open
OK
FailOver to Interface Driver
Session 1 InterfacesDriver>> add asiqdemo                  // IQ engine name
Service : [SQL Server]                                     // enter
Transport Type : [tcp] tli tcp                             // sun을 제외하면 tcp
Transport Address : 157.133.75.36 2345                     // IP, port
Transport Type : [tcp]                                     // enter
Transport Address :                                        // enter
Security Mechanism [ ] :                                   // enter
HA Failoverserver : [ HA Failover Server ]                 // enter
Error in adding asiqdemo

Session 1 InterfacesDriver>> exit
```

JDBC/OCDK 설정 – unix client

- **dscp 유틸리티를 이용(계속)**
  위 명령어가 정상적으로 수행되었다면 **$SYBASE/interfaces** 파일에 다음과 같은 내용이 들어 있습니다.

```
asiqdemo                                                        // for Solaris
          master tli tcp /dev/tcp \x000209299e4d32730000000000000000
          query tli tcp /dev/tcp \x000209299e4d32730000000000000000
```

```
asiqdemo                                                        // for HP, IBM….
          master tcp ether 157.133.75.36 2345
          query tcp ether 157.133.75.36 2345
```

- 수동으로 편집
  $SYBASE/interfaces 파일을 **vi** 와 같은 편집기를 사용하여 위에서 언급한 것처럼 편집하십시오. 단, 편집할 때 주의할 점은 **master, query** 앞의 문자는 반드시 탭으로 분리해야 합니다.

DBISQL

- SYBASE IQ의 기본적인 질의 편집 툴

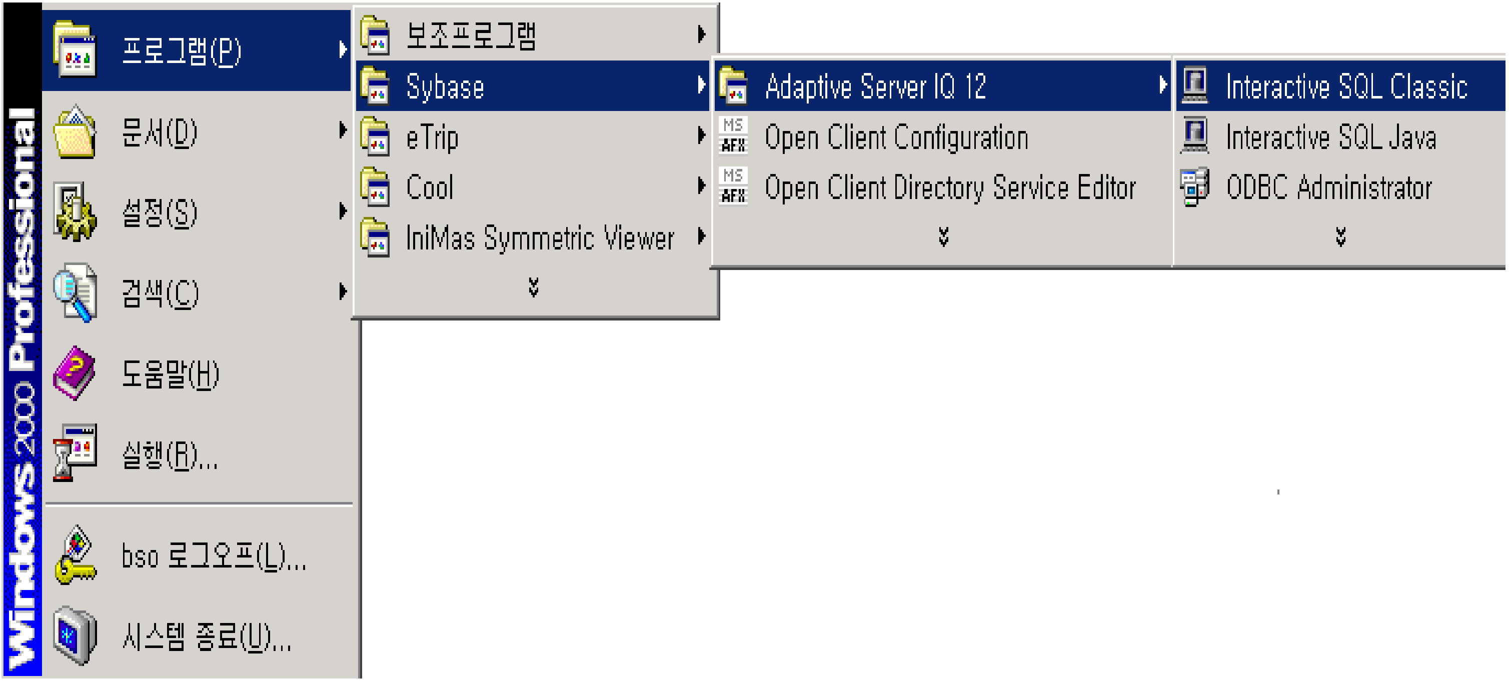
- Embedded SQL/C 계열
  - ● UNIX : dbisqlc
  - ● Window : interactive SQL Classic

- JAVA 계열
  - ● UNIX : dbisql
  - ● Window : interactive SQL JAVA

- 참고 : 원래 dbisql은 ASA(Adaptive Server Anywhere)의 기본 클라이언트 입니다.

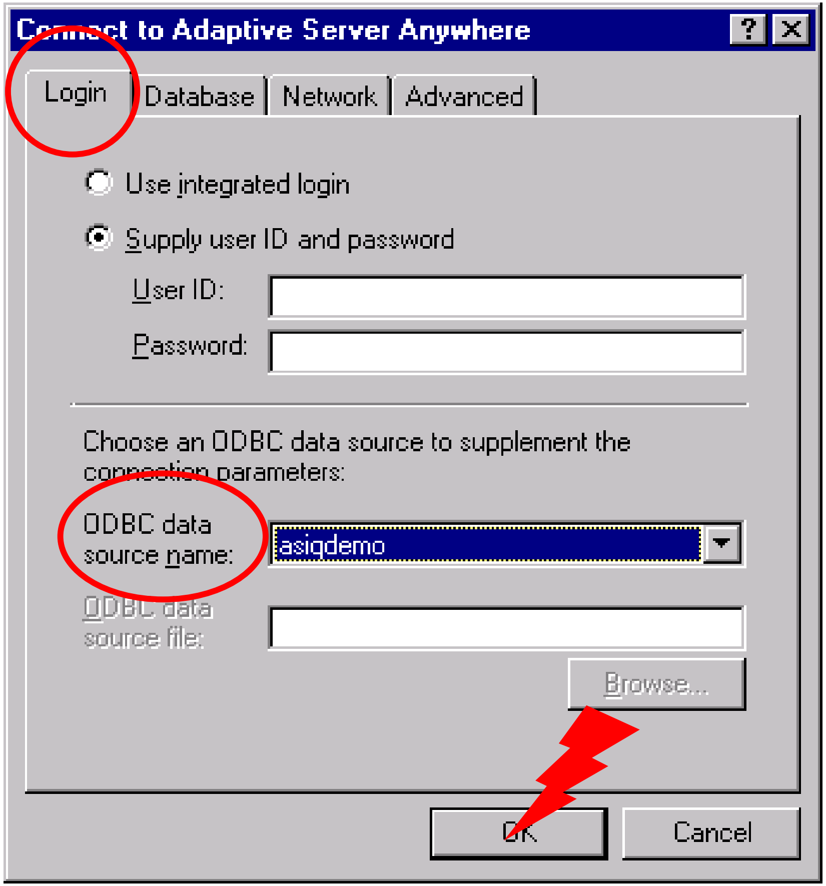
Win : Interactive SQL Classic

- Window 계열에서 사용하며 **ODBC**를 표준 **API**로 사용하고 있습니다.
- 아래 순서대로 **Interactive SQL Classic**을 실행합니다



Win : Interactive SQL Classic

- **Login 탭의 ODBC data source name 콤보 박스에서 ODBC 설정에서 미리 만들어 놓은 접속을 원하는 data source name을 선택한 후 OK 버튼을 클릭합니다**



Connect to Adaptive Server Anywhere

○ Use integrated login
● Supply user ID and password

User ID:
Password:

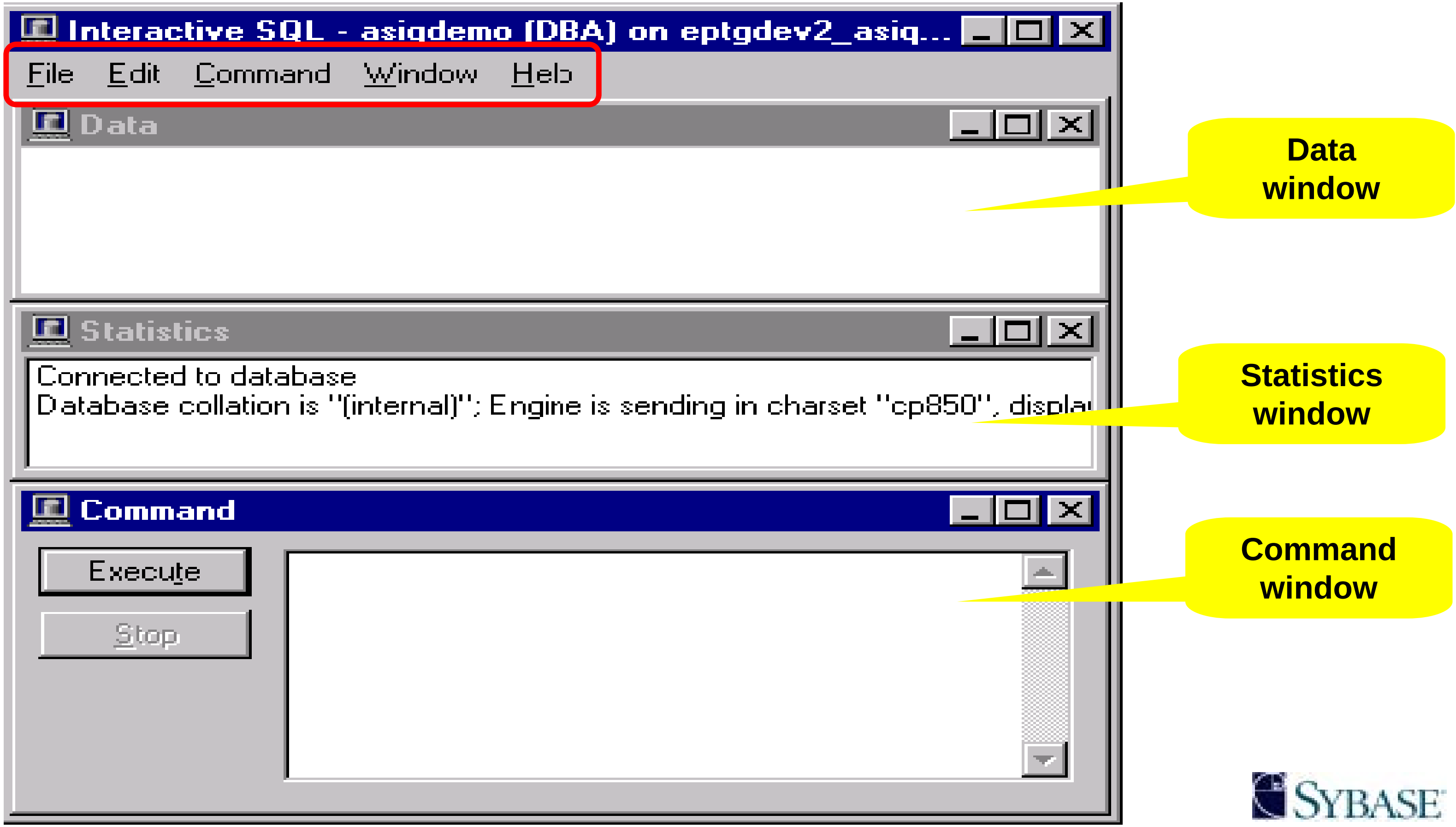
Choose an ODBC data source to supplement the connection parameters:

ODBC data source name: asiqdemo

ODBC data source file:

Win : Interactive SQL Classic

- **Data, Statistics, Command** 세 개의 서브 창을 가지 **Interactive SQL Classic** 메인 화면이 나오고 **Statistics** 창에 **Connected to database** 라는 접속 성공 메시지가 모이면 정상 접속 상태가 됩니다. 이 상태에서 **SQL**문을 편집하여 결과를 조회할 수 있습니다



```
Connected to database
Database collation is "(internal)"; Engine is sending in charset "cp850", displa
```

Data window

Statistics window

Command window

Win : Interactive SQL Classic

- File menu : Command 창에서 편집한 SQL문을 저장할 수 있는 Save as, 저장된 SQL문을 Command 창으로 불러올 수 있는 Open을 비롯하여 New, Exit 등이 있습니다.

- Edit menu : Command 창에서 편집할 수 있는 Cut, Copy, Paste, Delete와 접속한 서버의 특정한 테이블을 선택할 수 있는 Insert Table 등이 있습니다.또한 각각의 메뉴에는 단축 키와 기능 키가 설정되어 있어 사용이 간편합니다

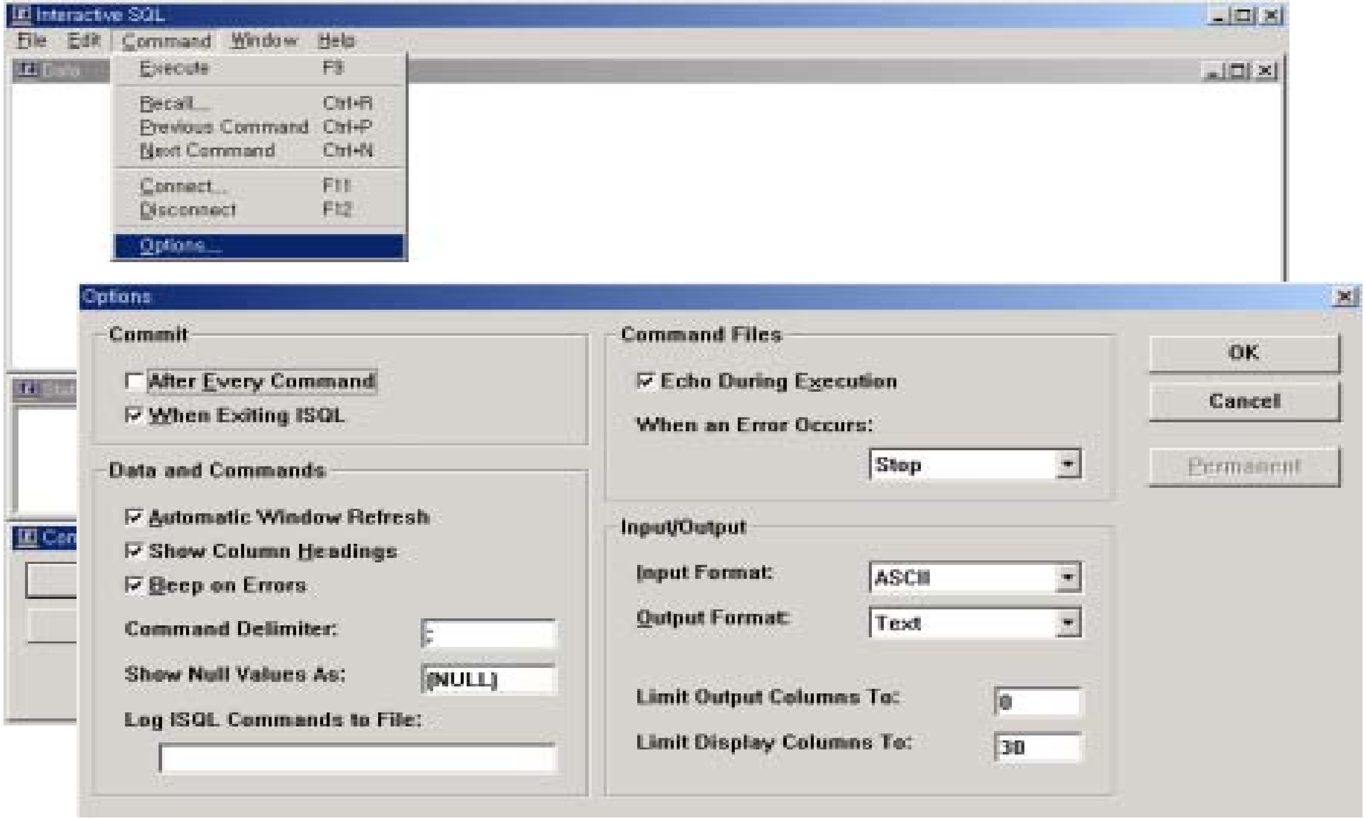
- Command menu : Command 창에서 편집한 SQL문을 실행하기 위한 Execute, Command 창에서 이미 사용한 명령어를 다시 사용할 수 있도록 제어하는 Recall, Previous Command, Next Command, 새로운 접속을 위한 Connect, 접속을 끊기 위한 Disconnect, 여러 가지 옵션을 설정할 수 있는 Options 등이 있습니다.

- Window menu : 서브 창의 배치를 제어하는 Tile, Always Tile, 세 개의 서브 창을 각각 활성화 시키는 Data, Statistics, Command 등이 있습니다.

- 그 외에 Help 메뉴 등이 있습니다

Win : Interactive SQL Classic

- 옵션 설명 : **Command** 메뉴에 **Options** 서브 메뉴를 클릭하면 아래와 같은 창이 새로 만들어 지며 각각에 대한 설명은 다음과 같습니다. 또한 각 옵션은 **SYBASE IQ**에 접속시 서버에 등록되어 있는 해당되는 각각의 옵션 값을 읽어 들여 설정합니다.



Commit

☐ After Every Command
☑ When Exiting ISQL

Data and Commands

☑ Automatic Window Refresh
☑ Show Column Headings
☑ Beep on Errors

Command Delimiter: ;

Show Null Values As: (NULL)

Log ISQL Commands to File:

Command Files

☑ Echo During Execution

When an Error Occurs: Stop

Input/Output

Input Format: ASCII

Output Format: Text

Limit Output Columns To: 0

Limit Display Columns To: 30

Win : Interactive SQL Classic

- **After Every Command : 실행되는 모든 단위 명령어 뒤에 자동으로 commit 명령어 실행 여부를 제어합니다. 서버에서 인식하는 옵션 명은 Auto_Commit 이며 기본 값은 Off 입니다.**

- **When Exiting SQL : dbisqlc의 접속을 끊거나 종료할 때에 commit 명령어 실행 여부를 제어합니다. 서버에서 인식하는 옵션 명은 Commit_on_Exit 이며 기본 값은 On 입니다.**

- **Automatic Window Refresh : Insert, Update, Delete의 결과가 현재 실행중인 질의 결과에 반영 여부를 제어합니다. 서버에서 인식하는 옵션 명은 Auto_Refetch 이며 기본 값은 On 입니다.**

- **Show Column Headings : 각 컬럼의 헤딩을 화면에 표시할 지 여부를 제어합니다. 서버에서 인식하는 옵션 명은 Headings 이며 기본 값은 On 입니다.**

- **Beep on Errors : 오류가 발생했을 때 경고음을 발생할지 여부를 제어합니다. 서버에서 인식하는 옵션 명은 Bell 이며 기본 값은 On 입니다.**

- **Command Delimiter : 한 문장의 종료를 표시하는 문자를 제어합니다. 서버에서 인식하는 옵션 명은 Command_Delimiter 이며 기본 값은 ;(세미콜론) 입니다**

Win : Interactive SQL Classic

- Show Null Value As : 각 테이블에 입력된 값 중에 NULL 값의 표현을 어떻게 할지 제어합니다. 서버에서 인식하는 옵션 명은 Nulls 이며 기본 값은 문자열 NULL 입니다.

- Log ISQL Commands to File : Command 창에서 편집된 SQL이 기록되는 파일의 이름을 제어합니다. 서버에서 인식하는 옵션 명은 Isql_Log 이며 기본 값은 공백 문자 입니다. 즉 별도 파일로 기록하지 않습니다.

- Echo During Execution : SQL 문이 실행되기 전에 Echo 될지 여부를 제어합니다. 서버에서 인식하는 옵션 명은 Echo 이며 기본 값은 On 입니다.

- When an Error Occurs : 여러 개의 SQL문을 하나의 배치로 실행 중에 오류가 발생하면 어떻게 처리할지를 제어합니다. 서버에서 인식하는 옵션 명은 On_Error 이며 기본 값은 prompt 입니다.

- Input Format, Output Format, Limit Output columns to : 현재 지원되지 않는 옵션입니다.

- Limit Display Columns To : 화면에 출력되는 각 컬럼의 최대 길이를 제어합니다. 서버에서 인식하는 옵션 명은 Truncation_Length 이며 기본 값은 30 입니다.

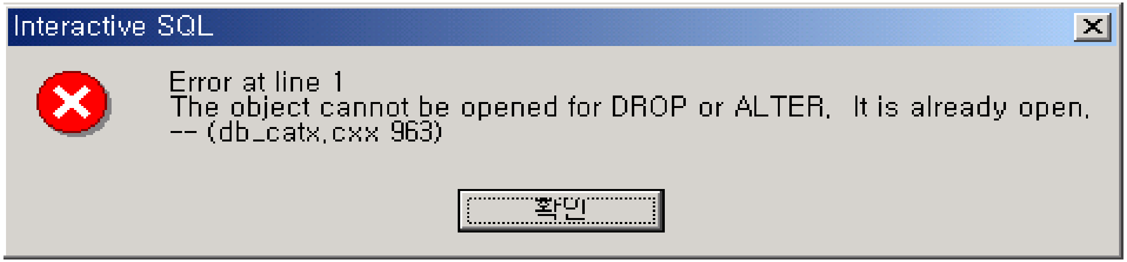
Win : Interactive SQL Classic

- **OK button : 현재 세션에서만 변경된 옵션의 효력이 발생하며 set temporary option 과 같습니다.**

- **Permanent button : 해당되는 사용자의 변경된 옵션의 효력이 지속적으로 발생하며 set option과 같습니다.**

■ 참고

dbisqlc 프로그램은 질의에 대한 결과를 조회할 때 ESQL/C의 커서를 선언하여 사용하고 있습니다. 그래서 커서를 오픈한 후에 원하는 결과가 클라이언트로 모두 전송되지 않고 클라이언트 버퍼 크기 만큼의 양만 전송되고 더 많은 양을 조회할 때 그 때 다시 전송되고 합니다. 여기서 주의할 점은 프로그램이 자동으로 커서를 종료하지 않는다는 것입니다. 이렇게 커서가 열려있는 상태에 있는 테이블이나 Stored Procedure는 다른 사용자의 DDL 명령을 아래 메시지처럼 블록킹하고 있습니다. 이런 상태를 벗어나려면 또 다른 SQL 문을 실행하거나 resume 이라는 명령어를 실행하십시오. 그러면 커서가 종료됩니다.



```
Error at line 1
The object cannot be opened for DROP or ALTER.  It is already open.
-- (db_catx.cxx 963)
```

UNIX : dbisqlc

- **Syntax : dbisqlc [-c "keyword=value……" | -d "delimiter" | -q | -x]**
  여기서 **–q** 는 **quite mode, -x**는 **syntax check only** 옵션이며 **keyword**에 해당되는 자세한 목록은 메뉴얼을 참조하시기 바랍니다

- **GUI Mode :** 그래픽 문자를 사용한 텍스트 형태로 제공되어 마우스를 사용할 수 없다는 것을 제외하면 **Window** 용과 거의 비슷합니다**.** 실행하기 위한 명령어는 **dbisqlc** 이며 뒤에 접속을 위한 여러 가지 옵션을 붙여줘야 하는데 붙이는 방법에 따라 다음과 같이 **3**가지 형태로 구분되며 기호에 따라 선택하여 혹은 조합하여 사용하십시오**. dbisqlc** 실행파일의 위치는 **$ASDIR/bin**에 있습니다**.**

  - **Command Line**에 직접 입력 **: dbisqlc**를 실행하면서 접속에 필요한 옵션을 명령어와 같이 입력합니다**.** 단**, dbisqlc**의 실행을 위한 **path**가 적절히 설정되었다고 가정합니다

    % dbisqlc –c "uid=DBA;pwd=SQL;eng=asiqdemo;dbn=asiqdemo"

  - **SQLCONNECT** 환경변수 이용 **:** 접속에 필요한 옵션을 **SQLCONNECT**라는 환경변수에 설정하고 그냥 단순히 명령어만 실행하면 됩니다**.**

    % dbisqlc

  - **.odbc.ini** 이용 **:** 접속에 필요한 옵션을 **.odbc.ini**라는 파일에 **data source name** 이라는 대표이름을 이용하여 저장하고 접속할 때 이 이름을 옵션으로 입력합니다**.**

    % dbisqlc –c dsn=asiqdemo

UNIX : dbisqlc

- **Quite Mode : UNIX 환경의 GUI 모드에서는 여러 가지 불편한 점이 많은 관계로 잘 사용하지 않지만 배치 파일 형태로 SQL 명령어들을 파일로 저장하여 놓고 이 파일을 입력 파일로 이용하여 실행할 때 많이 사용하는 방법입니다. 실행하기 위한 명령어 및 3 가지의 사용 형태는 모두 GUI 모드와 같고 각각의 뒤에 "–q filename" 옵션을 붙여주면 됩니다.**

- **Command Line에 직접 입력**

```
% dbisqlc –c "uid=DBA;pwd=SQL;eng=asiqdemo;dbn=asiqdemo" –q query.sql
```

- **SQLCONNECT 환경변수 이용**

```
% dbisqlc –q query.sql
```

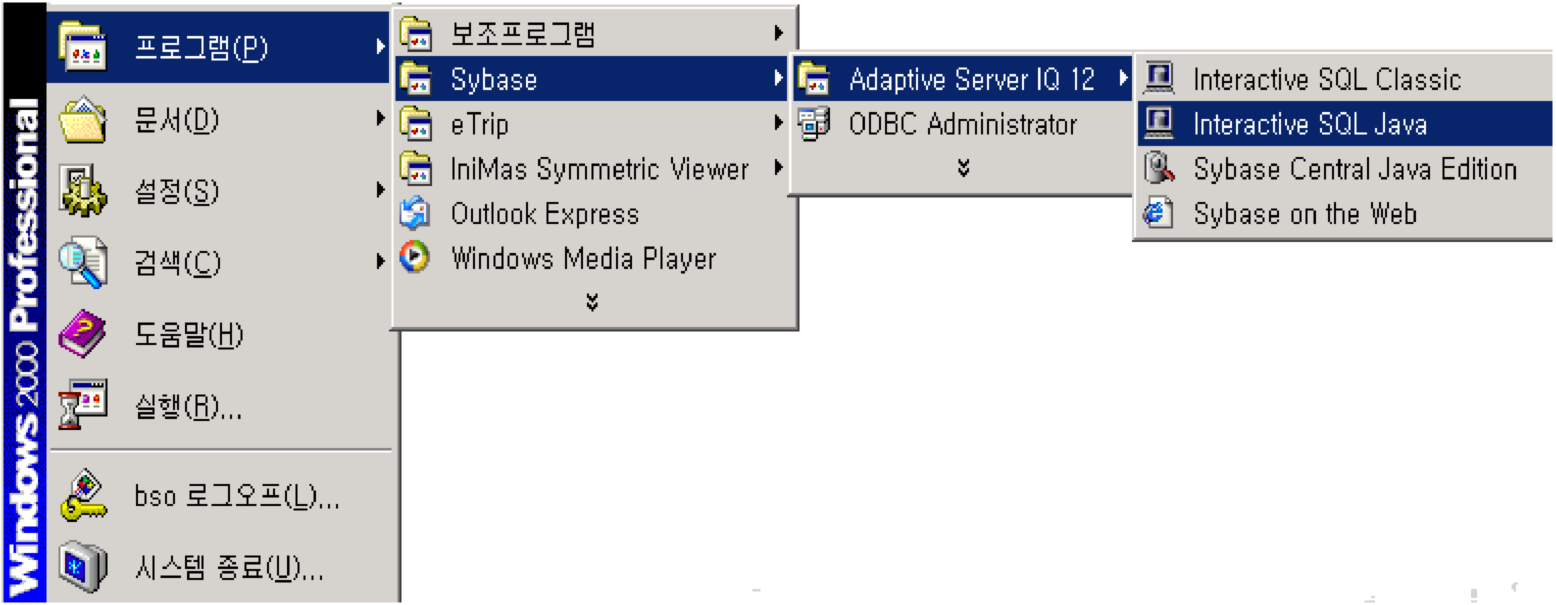
- **.odbc.ini 이용**

```
% dbisqlc –c dsn=asiqdemo –q query.sql
```

Win : Interactive SQL JAVA

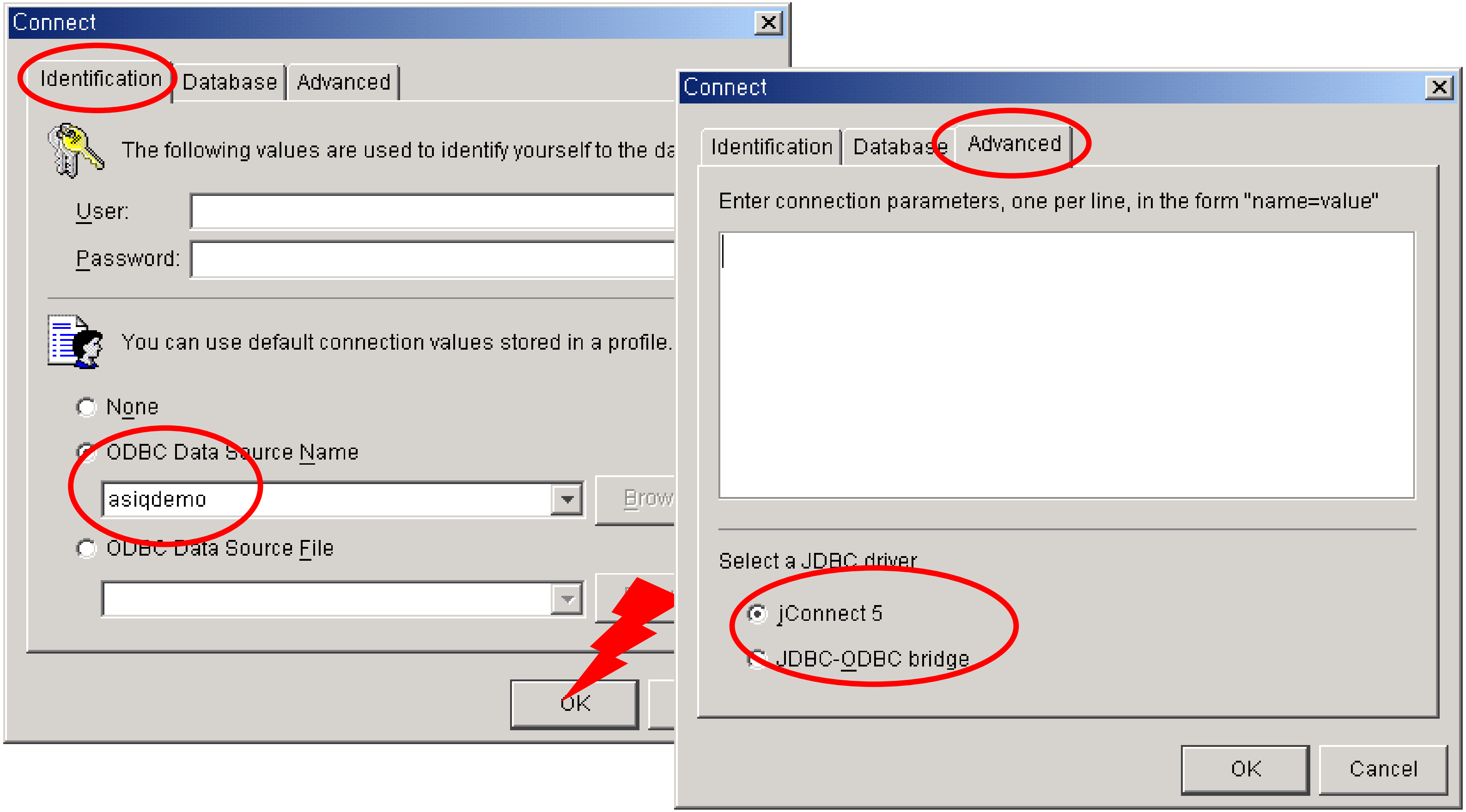
- **Window 계열에서 사용하며 JDBC를 표준 API로 사용하고 있으며 JDBC-ODBC API를 옵션으로 제공하고 있습니다.**

- 아래 순서대로 **Interactive SQL JAVA**를 실행합니다



Win : Interactive SQL JAVA

- **Identification** 탭의 **ODBC data source name** 콤보 박스에서 **ODBC** 설정에서 미리 만들어 놓은 접속을 원하는 **data source name**을 입력한 후 **OK** 버튼을 클릭합니다. 참고로 **Advanced** 탭에서 추가적으로 필요한 옵션과 **API**로 **JDBC**를 사용할 것인지 **JDBC-ODBC bridge**를 사용할 것인지를 선택할 수 있습니다.



The following values are used to identify yourself to the da

User:

Password:

You can use default connection values stored in a profile.

○ None

● ODBC Data Source Name

asiqdemo

○ ODBC Data Source File

Enter connection parameters, one per line, in the form "name=value"

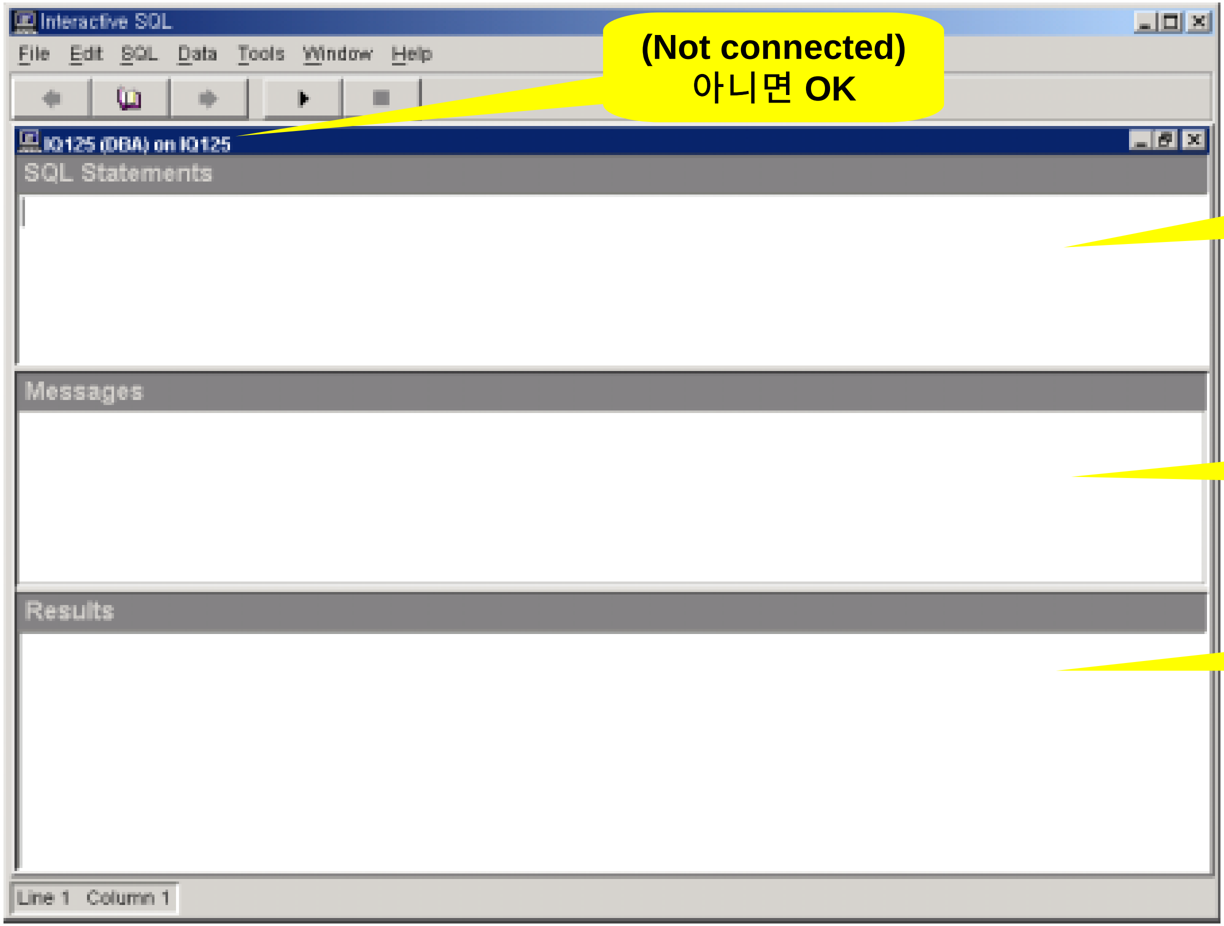
Select a JDBC driver

● jConnect 5

○ JDBC-ODBC bridge

Win : Interactive SQL JAVA

- **SQL Statements, Messages, Results** 세 개의 서브 창을 가지 **Interactive SQL JAVA** 메인 화면이 나오고 이 상태에서 **SQL Statements** 창을 통해 **SQL**문을 편집하여 결과를 조회할 수 있습니다



(Not connected)
아니면 **OK**

SQL Statements window

Messages window

Results window

Win : Interactive SQL JAVA

- File menu : SQL Statements 창에서 편집한 SQL문을 저장할 수 있는 Save as, 저장된 SQL문을 SQL Statements 창으로 불러올 수 있는 Open을 비롯하여 새로운 창을 통해 접속을 가능하게 해주는 New window, 열려있는 특정 창을 닫는 Close, SQL 문을 바로 읽어 들어 실행하는 Run Script, Exit등이 있습니다.

- Edit menu : SQL Statements 창에서 편집할 수 있는 Redo, Undo, Cut, Copy, Paste, Delete와 접속한 서버의 특정한 테이블을 선택할 수 있는 Insert Table 등이 있습니다.또한 각각의 메뉴에는 단축 키와 기능 키가 설정되어 있어 사용이 간편합니다

- SQL menu : SQL Statements 창에서 편집한 SQL문을 실행하기 위한 Execute, Execute Selection, 실행을 취소하는 Stop, SQL Statements 창에서 이미 사용한 명령어를 다시 사용할 수 있도록 제어하는 History, Previous SQL, Next SQL, 새로운 접속을 위한 Connect, 접속을 끊기 위한 Disconnect, SQL문의 기록을 시작, 종료하는 Start Logging, Stop Logging 등이 있습니다.

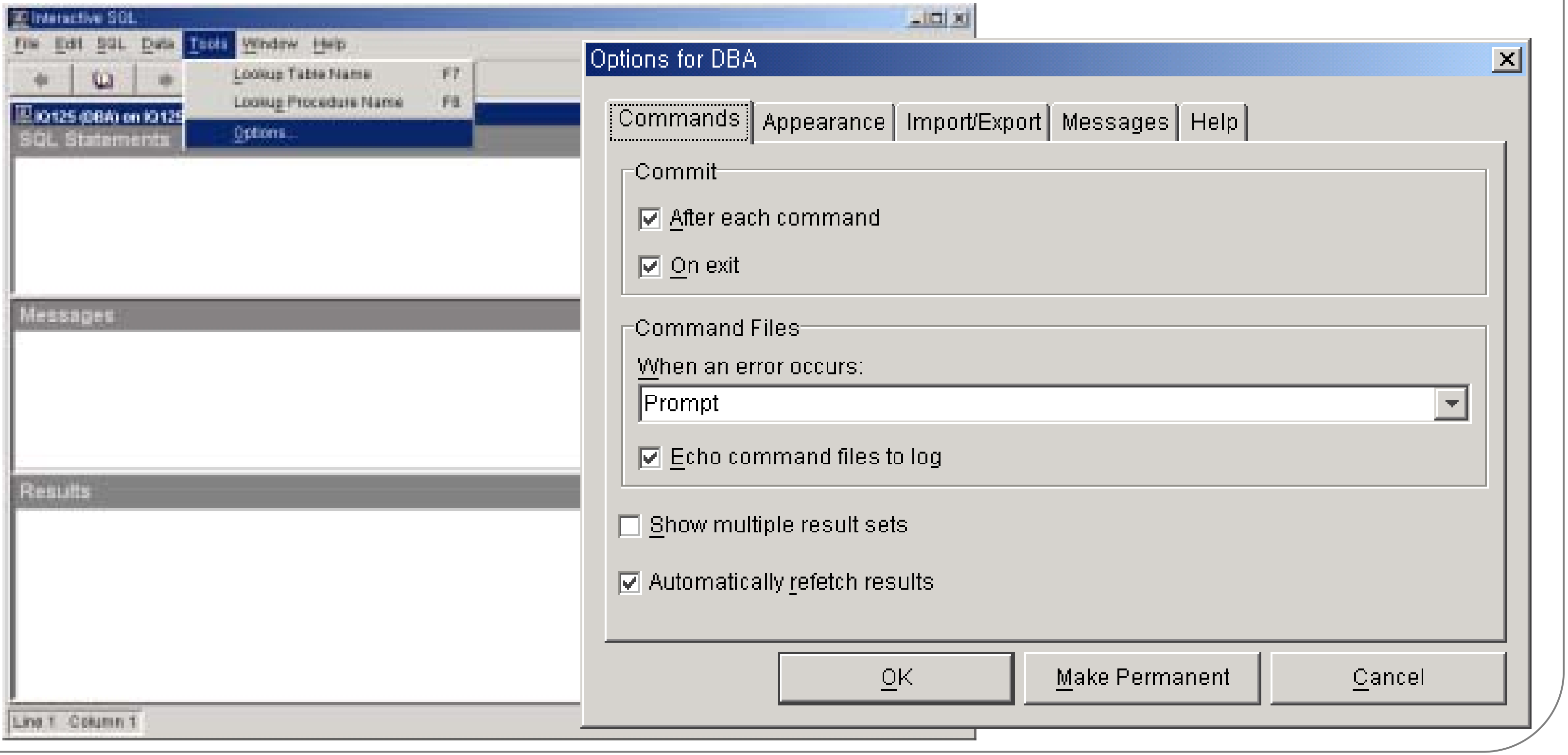
- Data menu : 현재 툴을 사용하고 있는 클라이언트의 위치를 기준으로 DB로 혹은 DB에서 데이터를 Import, Export 해 줍니다. Import 시 주의할 점은 데이터가 insert into values 형태로 한건 한건 들어 가기 때문에 부작용을 충분히 숙지한 후에 사용 바랍니다.

- Window menu : 서브 창 모두를 종료하는 Close all, 서브 창의 배치를 제어하는Cascade, Tile Horizontally, Tile Vertically, 열려 있는 서브 창을 각각 활성화 시키는 메뉴등이 있습니다. 그 외에 Help 메뉴 등이 있습니다

Win : Interactive SQL JAVA

- **Tools menu : DB에 생성되어 있는 테이블을 조회, 선택할 수 있는 Lookup Table Name, Stored Procedure를 조회 선택할 수 있는 Lookup Procedure Name, 설정의 변경을 가능하게 해주는 Options등이 있습니다.**

- **옵션 설명 : 옵션은 SYBASE IQ에 접속시 서버에 등록되어 있는 해당되는 각각의 옵션 값을 읽어 들여 설정합니다. 각각의 옵션은 Interactive SQL Classic와 거의 유사합니다.**



Commit

☑ After each command

☑ On exit

Command Files

When an error occurs: Prompt

☑ Echo command files to log

☐ Show multiple result sets

☑ Automatically refetch results

UNIX : dbisql

- 실행 파일은 **$SYBASE/ASIQ-12_5/bin/dbisql** 이며 실행 방법은 **dbisqlc**와 유사하고 실행 후의 내용은 **Interactive SQL JAVA**와 같습니다.

dbisql command

- clear : data window 혹은 results window의 내용을 모두 지웁니다.

- configure : dbisql 툴의 옵션을 변경할 수 있는 option window를 활성화 합니다.

- connect : 기술된 데이터베이스와의 연결을 수행하며 연결중이었던 채널은 종료되지 않고 비활성화 되며 새로운 채널이 활성화 됩니다. 이렇게 하나의 클라이언트에서 여러 개의 채널을 열어서 사용할 수 있습니다.
  - connect [to *engine-name*] [database *database-name*]
    [as *connection-name*] [user] *userid* [identified by *password*]
  - connect using *connect-string*

- disconnect : 현재 활성중인 데이터베이스와의 연결을 종료합니다. 열려 있는 모든 채널과의 연결을 종료하려면 all 옵션을 주면 됩니다. Commit_On_Exit 옵션의 영향이 없습니다.
  - disconnect *[{ connection-name | current | all }]*

- exit, quit, bye : dbisql을 종료합니다. Commit_On_Exit 옵션의 설정 값에 따라 종료되지 않은 트랜잭션의 COMMIT 혹은 ROLLBACK을 수행합니다.

- help : ASA에서만 온라인 도움말 기능이 가능합니다.

dbisql command

- **parameters : quite**모드에서 사용하는 **command file**에서 파라미터를 정의하는데 사용합니다. 선언된 이후에 참조될 때는 **{ }**로 양쪽이 감싸져야 하며 나중에 **read**명령어를 통해 파라미터 전달이 가능합니다.
  - ● **parameters** *param1 [,param2,……]*

- **read : command file**에 선언된 파라미터의 값을 넘겨줍니다.

- 예**) parameter & read**

  % dbisqlc –c dsn=asiqdemo –q read query.sql 50000

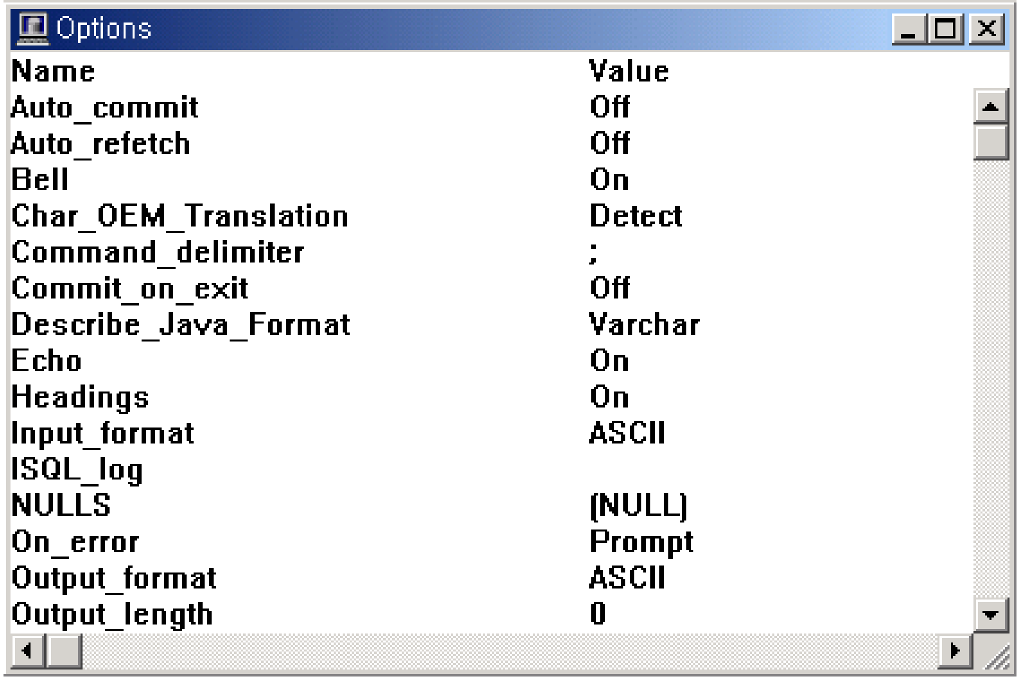
  % vi query.sql

```
parameters param_salary;
SELECT emp_lname
FROM employee
WHERE salary > {param_salary};
```

- **set connection : connect** 명령어에 의해 연결된 여러 개의 채널 중에 활성 채널을 선택합니다.

dbisql command

- **set :** 현재 연결된 사용자에게 설정된 옵션 값들을 별도의 창에 출력해 줍니다.



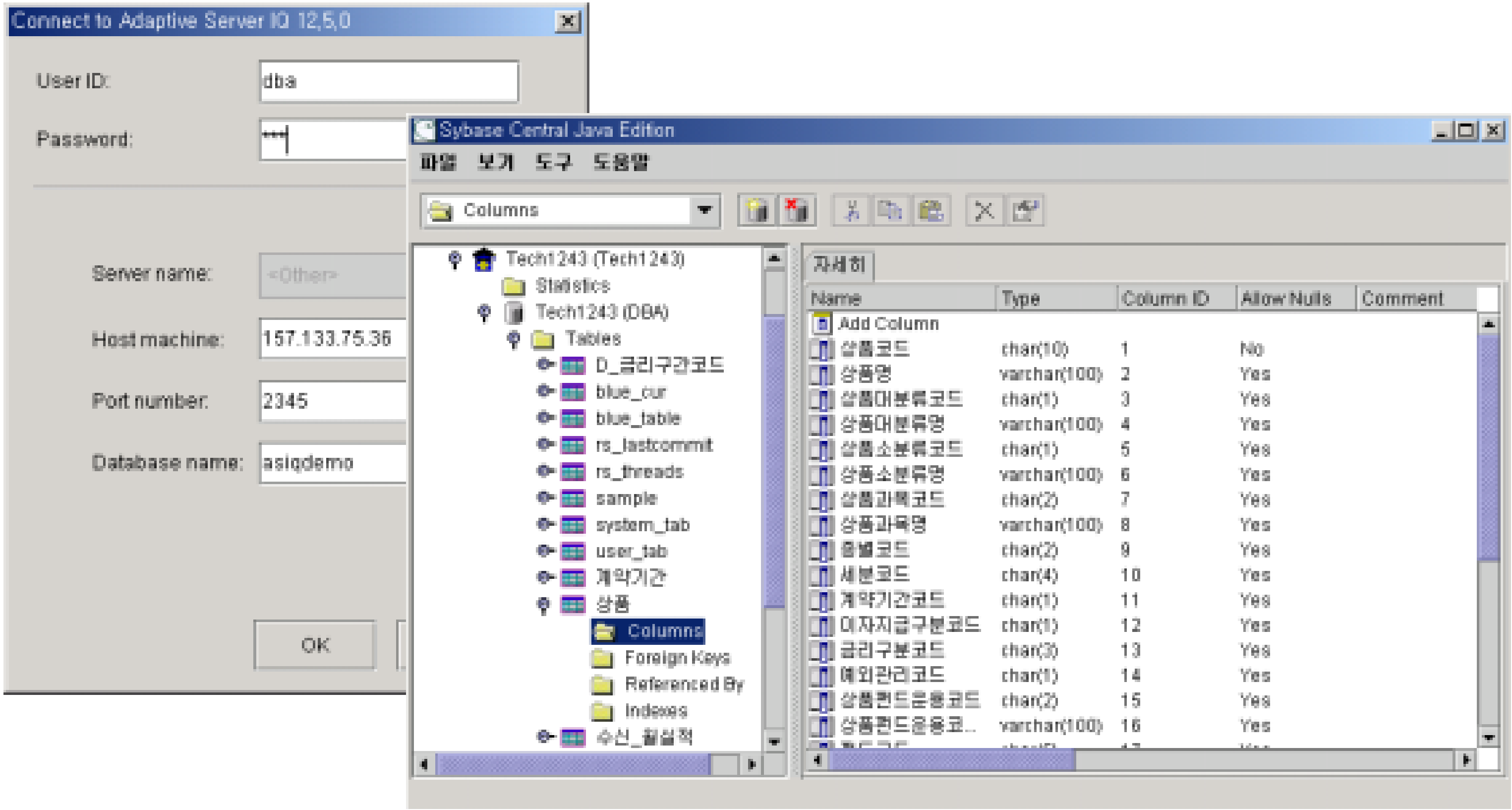
Options

Name	Value
Auto_commit	Off
Auto_refetch	Off
Bell	On
Char_OEM_Translation	Detect
Command_delimiter	;
Commit_on_exit	Off
Describe_Java_Format	Varchar
Echo	On
Headings	On
Input_format	ASCII
ISQL_log	
NULLS	(NULL)
On_error	Prompt
Output_format	ASCII
Output_length	0

- 그 외에 **start engine, stop engine, start database, stop database**등이 있는데 이는 **ASA** 환경에서 사용되는 명령어들 입니다. **SYBASE IQ** 환경에서는 잘못 동작할 수 있으니 주의 바랍니다.

참조 : Sybase Central

- **SYBASE IQ**와 연결하여 **Admin** 작업을 할 수 있는 디폴트 툴이며 다음과 같은 작업을 수행할 수 있습니다.
  - 데이터베이스 생성, 디바이스 생성 및 제거
  - 테이블, 뷰, 인덱스, **Stored Procedure** 생성
  - 사용자 및 그룹의 권한관리


Connect to Adaptive Server IQ 12.5.0
OK
Sybase Central Java Edition
파일
보기
도구
도움말
Columns
자세히

User ID: dba

Password: ***

Server name: <Other>

Host machine: 157.133.75.36

Port number: 2345

Database name: asiqdemo

- Tech1243 (Tech1243)
  - Statistics
  - Tech1243 (DBA)
    - Tables
      - D_금리구간코드
      - blue_cur
      - blue_table
      - rs_lastcommit
      - rs_threads
      - sample
      - system_tab
      - user_tab
      - 계약기간
      - 상품
        - Columns
        - Foreign Keys
        - Referenced By
        - Indexes
      - 수신_월실적

Name	Type	Column ID	Allow Nulls	Comment
Add Column				
상품코드	char(10)	1	No	
상품명	varchar(100)	2	Yes	
상품대분류코드	char(1)	3	Yes	
상품대분류명	varchar(100)	4	Yes	
상품소분류코드	char(1)	5	Yes	
상품소분류명	varchar(100)	6	Yes	
상품과목코드	char(2)	7	Yes	
상품과목명	varchar(100)	8	Yes	
종별코드	char(2)	9	Yes	
세분코드	char(4)	10	Yes	
계약기간코드	char(1)	11	Yes	
이자지급구분코드	char(1)	12	Yes	
금리구분코드	char(3)	13	Yes	
예외관리코드	char(1)	14	Yes	
상품펀드운용코드	char(2)	15	Yes	
상품펀드운용코...	varchar(100)	16	Yes	

isql

- ASE의 기본 SQL 편집툴이며 UNIX와 Win 모두 같은 환경에서 실행 가능합니다.

- 클라이언트를 실행하면서 연결에 필요한 파라미터를 실행 명령어와 같이 입력합니다

```
isql –Udba –Psql –Sasiqdemo
1> sp_iqstatus
2> go
.
.
.
1> exit
```

- 참고
  isql 및 Win 환경에서 사용하는 SQL Advantage와 같은 SYBASE Native Driver인 OCDK를 사용하는 툴들은 SYBASE IQ와 접속할 때 ODBC 연결과는 기본 설정 옵션등 여러 가지면에서 다릅니다. 그래서 가능하면 사용을 자제하시고 꼭 사용하실 때에도 이 점 충분히 숙지하시고 사용하시기 바랍니다.

Using WatcomSQL

WatcomSQL

- WatcomSQL은 ASA에서 사용하는 SQL언어이며 ANSI 표준에 추가적으로 Sybase의 향상되어진 기능을 제공하는 질의를 위한 구조적인 언어입니다. 이 언어는 질의를 위한 SQL 문 뿐만 아니라 제어문과 같은 프로그래밍 구조도 포함하는 절차적 언어이기도 합니다.

- T-SQL은 Transact-SQL의 줄임말로 ASE에서 사용되어지는 SQL 언어입니다.

- SYBASE IQ에서는 WatcomSQL 대부분을 기본적으로 제공하고 있으며 ASE와 호환성을 유지하기 위해 T-SQL도 사용 가능 하지만 성능적인 측면을 고려하여 SYBASE IQ에서는 WatcomSQL 사용을 권장합니다.

- 참고 : SYBASE IQ에서 WatcomSQL 모두를 지원하지는 않으며 이 교재에서 언급하는 WatcomSQL은 ASA가 아닌 SYBASE IQ를 위한 SQL 임을 기본 전재로 합니다. 또한 태생적인 인연으로 인하여 MS SQL Server의 Transact-SQL은 SYBASE가 개발하고 추후 MS가 수정을 한 SQL 언어임을 참고로 알려드립니다.

WatcomSQL 장점

- 논리적으로 관련된 문장들을 하나의 블록형태로 묶어 처리하기 때문에 프로그램을 모듈화해서 개발할 수 있습니다.
- 프로그래밍 언어에서 사용 가능한 변수 등의 선언이 가능합니다.
- IF, LOOP, WHILE 등과 같은 제어문 등을 포함하기 때문에 절차적인 언어 구조로 된 프로그램이 가능합니다.
- 예외처리를 위한 기능을 제공하기 때문에 사용자의 예외에 대한 처리 로직을 줄일 수 있습니다.
- 여러 SQL 문장을 블록으로 묶어 서버와 통신하기 때문에 통신량을 줄일 수 있습니다.
- Stored Procedure와 같이 사용하여 프로그램을 서버쪽에 두고 관리하기 때문에 유지보수에 대한 비용을 줄일 수 있으며 통신량을 추가적으로 줄일 수 있습니다.

compound statement 구조

- **BEGIN, END** 라는 **keyword**로 **SQL**문의 블록을 생성하며 그 안에 지역선언부**,** 실행부**,** 예외처리부와 같은 내용이 들어 갈 수 있으며 지역선언부와 예외처리부는 생략 가능합니다**.** 또한 이렇게 만들어지는 블록은 옵션으로 이름**(LABEL)**을 줄 수도 있고 또한 하나의 블록을 단일 명령어 처리하기 위해서 **BEGIN** 뒤에 **ATOMIC** 옵션을 사용할 수도 있습니다**.** 이러한 **BEGIN~END**로 묶여진 블록구조를 **compound statement** 라고 합니다**.**

- 지역선언부에서는 지역변수**,** 커서**, EXCEPTION,** 임시 테이블을 선언할 수 있으며 이렇게 만들어진 **Object**들은 그 **compound statement** 내에서만 사용 가능하며 경우에 따라 생략 가능합니다**.**

- 예외 처리부에서는 지역선언부에서 선언되어진 예외에 대한 처리루틴이 포함되지만 경우에 따라 생략 가능합니다**.**

- 이런 구조는 중첩될 수 있고 프로시져**,** 함수등의 몸체로 사용되어 지며 블록내의 모든 개별 **SQL**문은 **;**로 구분됩니다**.**

```
[ label;]                                   // 이름
BEGIN [ ATOMIC ]                            // 시작
          [ local-declaration ]             // 지역선언부
          statement-list;                   // 실행부
          [ exception ]                     // 예외처리부
END [ label ]                               // 종료
```

Compound Statement

compound statement 구조

- 참고 : compound statement 내부에서만 사용 가능한 명령어
  - ● declare variable
  - ● declare exception, exception handler
  - ● case statement (not case expression)
  - ● cursor for
  - ● cursor (declare cursor, open, fetch, close)
  - ● execute immediate (Dynamic SQL)
  - ● signal
  - ● ……

배치 구조

- 클라이언트에서 서버로 전송한 **SQL** 문장 단위를 배치라고 말하며 하나의 배치에서는 거의 모든 **WatcomSQL**문을 사용할 수 있습니다. 문법에 따라 어떤 문장요소는 한문장이 바로 배치가 되어야만 하는 경우도 있고 여러 문장이 모여 하나의 배치를 이루는 경우도 있습니다.

- 만약 하나의 배치에서 **syntax** 오류가 발생한다면 그 배치 전체가 실행되지 않습니다. 단, 해당 **object**가 존재하지 않는 경우는 **syntax** 오류로 보지 않습니다.

- **INSERT, UPDATE, DELETE, SELECT, COMMIT**와 같은 질의를 위한 **SQL** 기본 문장들은 여러 개의 **SQL**문을 하나로 배치로 처리하기 위해서 마지막 **SQL** 문장 뒤에 세미콜론으로 문장을 분리하면 되지만 그 외의 프로그래밍을 위한 요소들은 반드시 한 문장이 하나의 배치로 처리되어야만 합니다.

- 예)

```
INSERT INTO department ( dept_id, dept_name ) VALUES ( 220, 'Eastern Sales' )
UPDATE employee SET dept_id = 220 WHERE dept_id = 200 AND state = 'MA'
COMMIT;                                   // 3 statement per 1 batch

INSERT INTO department ( dept_id, dept_name ) VALUES ( 220, 'Eastern Sales' );
UPDATE employee SET dept_id = 220 WHERE dept_id = 200 AND state = 'MA‘;
COMMIT;                                   // 1 statement per 1 batch
```

local variable

- **BEGIN** 바로 다음에 **DECLARE**문에 의해 **compound statement** 내부에서만 선언되며 그 내부에서만 사용할 수 있습니다. 즉 **BEGIN ~ END** 안에서만 사용 가능 합니다.

- 콤마에 의해 여러 개의 변수를 같이 선언하는 것은 불가능하며 만약 필요 하다면 매번 **DECLARE**를 사용해야 하며 선언과 동시에 디폴트로 특정한 값을 할당할 수는 없습니다.

- 변수 이름의 시작은 변수라는 것을 나타내기 위해 **v_**와 같은 특별한 문자 사용하기를 권장하며 이 문장의 끝은 반드시 세미콜론으로 닫아줘야 합니다.

- 사용되기 전에 반드시 선언이 먼저 되어야 하고 선언될 때 **data type**에 관계없이 **NULL** 값을 가지고 있습니다

- **syntax**

```
DECLARE  variable-name  date-type;
```

- 예)

```
BEGIN
   DECLARE v_max_sales INTEGER;
   DECLARE v_half_max    NUMERIC(10,0);
   ………………..
END;
```

connection level variable

- **CREATE VARIABLE**문에 의해 **compound statement** 내**,** 외부에서 선언되며 해당 세션이 종료되거나 **DROP VARIABLE**를 통해 삭제됩니다

- 콤마에 의해 여러 개의 변수를 같이 선언하는 것은 불가능하며 만약 필요 하다면 매번 **CREATE VARIABLE**를 사용해야 하며 선언과 동시에 디폴트로 특정한 값을 할당할 수는 없습니다**.**

- 변수 이름의 시작은 변수라는 것을 나타내기 위해 **v_**와 같은 특별한 문자 사용하기를 권장하며 이 문장의 끝은 반드시 세미콜론으로 닫아줘야 합니다**.**

- 사용되기 전에 반드시 선언이 먼저 되어야 하고 선언될 때 **data type**에 관계없이 **NULL** 값을 가지고 있습니다

- **syntax**

  CREATE VARIABLE *variable-name date-type;*

- 예**)**

  CREATE VARIABLE v_max_sales INTEGER;
  CREATE VARIABLE v_half_max NUMERIC(10,0);

variable-data type

Data Type	Range	Max Prec.	Storage (byte)
CHAR (n) CHARACTER (n)	1 <= n <= 255		n
VARCHAR (n) CHARACTER VARYING (n)	1 <= n <= 255		n
VARCHAR (n) CHARACTER VARYING (n)	256 <= n <= 32K		256 + (n - 255)
INTEGER UNSIGNED INT	-2,147,483,648 ~ 2,147,483,647 0 ~ 42,942,967,294	10 11	4
TINYINT	0 ~ 255	3	1
SMALLINT	-32,768 ~ 32,767	5	2
BIGINT UNSIGNED BIGINT	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 0 ~ 18,446,744,073,709,551,615	19 20	8

variable-data type

Data Type	Range	Max Prec.	Storage (byte)
FLOAT (n)	Flatform-dependent	16	4 or 8
REAL	Flatform-dependent	7	4
DOUBLE	2.22 (^308) ~ 1.79 (^308)	15	8
DECIMAL (p,s) NUMERIC (p,s)	-10^38 ~ 10^38 – 1	126	2 to 69
BINARY (n)	1 <= n <= 255		256
VARBINARY (n)	1 <= n <= (32k – 1)		32K – 1
LONG BINARY			64K –1
BIT	0, 1, NULL		1

variable-data type

Data Type	Range	Max Prec.	Storage (byte)
DATE	0001/01/01 ~ 9999/12/31		4
DATETIME SMALLDATETIME TIMESTAMP	0001/01/01 00:00:00.000000 ~ 9999/12/31 23:59:59.999999		8
TIME	00:00:00.000000 ~ 23:59.59.999999		8

변수에 값 할당하기

- SET문을 통해서

```
CREATE PROCEDURE greater_proc ( IN v_a INT, IN v_b INT, OUT v_c INT )
BEGIN
    IF v_a > v_b THEN
        SET v_c = v_a;
    ELSE
        SET v_c = v_b;
    END IF;
END
```

- single row SELECT문을 통해서

```
BEGIN
    DECLARE v_customer_id INT;
    DECLARE v_orders INT;
    SELECT COUNT(b.id) INTO v_orders
    FROM customer a, sales_order b
    WHERE a.id = b.id
    AND a.id = v_customer_id;
END
// 만약 select 한 결과가 2건 이상이면 오류 발생
```

temporary table

- 프로그램밍 언어에서 임시 결과를 보관하기 위해 변수를 사용해왔던 것 처럼 **SQL** 이란 언어는 취급하는 대상이 한건 한건의 데이터일 수도 있지만 집합단위의 데이터 이기 때문에 이런 집합 단위의 임시 결과를 보관하기 위한 구조가 필요한데 이를 충족할 만한 것이 임시 테이블 **(in-line view** 포함**)** 입니다.

- 데이터베이스 엔진이 질의를 컴파일 해서 결과를 산출할 때 **GROUP BY, ORDER BY, JOIN** 과 같은 작업 역시도 한 번의 작업으로 그 처리를 수행 할 수 없기 때문에 내부적으로 임시 테이블을 사용합니다.

- 임의의 사용자가 엔진이 사용하는 것 처럼 임시 저장용도로 사용하기 위해 임의의 임시 테이블을 만들어 사용할 수 있습니다. 이렇게 임의의 테이블을 만들어 사용해서 얻는 이점은 그 테이블에 인덱스를 생성할 수 있어 임시 저장량이 많을 경우 많은 도움이 됩니다.

- 임시 테이블을 누가 만들어 사용하던지 데이터가 작업되는 영역은 **IQ TEMPORARY STORE** 입니다.

- 참고 **: In-line view vs. Temp table**
  일반적으로 **FROM** 절에 임시 계산 값을 저장하는 **In-line view** 형태를 많이 사용하는데 같은 용도로 인덱스를 만들 정도로 데이터 양이 많이 산출되는 결과라면 임시 테이블을 만들어 사용하는 편이 과정상 불편할 수 있지만 좋은 성능을 발휘합니다.

local temporary table

- **compound statement** 내**/**외부에서 만들어지며 만들어진 영역을 벗어나거나 세션을 마치거나 **DROP TABLE**문을 만나면 삭제되나 **DROP TABLE** 사용을 권장합니다.

- **DECLARE LOCAL TEMPORARY TABLE** 이라는 명령어를 통해 만들 수 있고 다른 세션 다른 사용자는 이 테이블을 사용할 수 없습니다. 또한 임시 테이블에 입력된 데이터는 기본적으로 **COMMIT** 시에 삭제되지만 옵션으로 저장상태를 유지할 수도 있습니다.

- 임시 테이블이라도 필요한 컬럼에 적절한 인덱스는 만들어야 합니다.

- 만약 프로시져 내부에서 만든다면 임시 테이블의 이름은 그 프로시져의 이름과 연관이 있도록 명명하는 편이 디버깅에 편리합니다

- **syntax**

```
DECLARE LOCAL TEMPORARY TABLE  table-name
       ( { column-definition  [ column-constraint….] | table-constraint }, …)
       ON COMMIT  { DELETE | PRESERVE } ROWS ];
```

- 예)

```
DECLARE LOCAL TEMPORARY TABLE customer_temp
( cust_id  INT,  cust_name char(20), cust_address char(255) )
ON COMMIT PRESERVE ROWS;
```

global temporary table

- **compound statement** 내/외부에서 만들어지며 **DROP TABLE**문을 만나면 삭제되며 연결상태가 종료되면 저장된 데이터는 삭제됩니다. 단, 테이블 정의는 보관됩니다.

- **CREATE GLOBAL TEMPORARY TABLE** 이라는 명령어를 통해 만들 수 있고 다른 세션 다른 사용자는 이 테이블을 사용할 수 있지만 데이터는 사용할 수 없습니다. 또한 임시 테이블에 입력된 데이터는 기본적으로 **COMMIT** 시에 삭제되지만 옵션으로 저장상태를 유지할 수도 있습니다.

- 임시 테이블이라도 필요한 컬럼에 적절한 인덱스는 만들어야 하며 만약 프로시져 내부에서 만든다면 임시 테이블의 이름은 그 프로시져의 이름과 연관이 있도록 명명하는 편이 디버깅에 편리합니다

- **syntax**

```
CREATE GLOBAL TEMPORARY TABLE  table-name
        ( { column-definition  [ column-constraint….] | table-constraint }, …)
        ON COMMIT  { DELETE | PRESERVE } ROWS ];
```

- 예)

```
CREATE GLOBAL TEMPORARY TABLE customer_temp
( cust_id  INT,  cust_name char(20), cust_address char(255) )
ON COMMIT PRESERVE ROWS;
```

참고

-
```
DECLARE LOCAL TEMPORARY TABLE employee
(
  emp_id       INT              NOT NULL,
  lname        CHAR(30)         NOT NULL,
  fname        CHAR(30)         NOT NULL,
  salary       UNSIGNED INT     NOT NULL,
  dept_id      INT              NOT NULL
) IN SYSTEM
```

- 위의 테이블은 **SYBASE IQ Temporary Store** 영역에 생성되지 않고 **ASA**의 임시공간인 **/tmp/.SQLAnywhere/**에 휘발성 파일로 생성되며 **2GB**의 파일크기 한계를 가지고 있습니다**.** 이 한계 값에 도달하면 **IQ Server**가**down**되거나 **hang** 상태가 되는등 여러 가지 부작용이 많습니다**.IQ**에서 **ASA**는 단순히 **Catalog**를 위한 공간이므로 사용자들의 사용을 금하여 주십시오**.**

- **local temporary table vs. global temporary table**
  ETL 작업에서 **CDC** 데이터를 본 테이블에 반영하기 전에 로딩하기 위한 임시 테이블 형태와 같이 사용빈도가 많은 경우는 **global temporary table**을 사용하는 편이 서버에게 테이블을 매번 만드는 부담을 줄여줄 수 있습니다**.**

if

- 다른 언어들과 유사하게 일치하는 조건에 따라 선택적으로 작업을 할 수 있게 해줍니다. 만약 조건이 **TRUE** 이면 **THEN ~ ELSE** 사이의 문장을 수행하고 **FALSE, NULL** 이면 **ELSE ~ END IF** 사이의 문장을 수행합니다.

- 논리적으로 이해하기 편하게 하기위해 **IF ~ ELSE**문을 중첩해서 사용하기 보다는 **IF ~ ELSEIF ~ ELSE**나 **CASE** 문의 사용을 권장합니다.

- 문장 구현시 **ELSEIF** 문의 스펠링과 종료시에 **END IF**의 사용에 주의 하시기 바랍니다.

- 조건절에서 **NULL** 값에 대한 비교는 **IS NULL** 연산자를 사용하시면 됩니다.

- **syntax**

```
IF  search-condition  THEN
    statement-list;
[ ELSEIF  search-condition  THEN
    statement-list; ]
[ ELSE
    statement-list; ]
END IF;
```

case(1)

- 중첩된 **IF**문 처럼 여러 조건에 따라 분기해야 한다면 이 때 사용 가능한 구조가 **CASE** 입니다. 이를 **CASE statement**라고 합니다.

- 문장구현 종료시에 **END CASE**의 사용에 주의 하시기 바랍니다.

- **syntax**

```
CASE  value-expression
  WHEN  [ constant | NULL ]  THEN  statement-list;
  [ WHEN  [ constant | NULL ]  THEN  statement-list; ]
  ELSE  statement-list;
END CASE;
```

- 예)

```
BEGIN
   DECLARE prod_name CHAR(20);
   DECLARE type CHAR(10);
   SELECT name INTO prod_name FROM product
   WHERE id = 10;
   CASE prod_name
      WHEN ‘Tee Shirt’ THEN SET type = ‘SHIRT’
      ELSE SET type = ‘Unknown’
   END CASE;
END
```

case(2)

- **SELECT절의 혹은 컬럼을 대신할 수 있는 다른 절에서 사용이 가능하며 컬럼의 분기조건이 나타날 때 사용할 수 있는 구조입니다. 이를 CASE expression 이라고 합니다.**

- **SYBASE IQ의 WatcomSQL에서는 SELECT 절에 IF문을 사용할 수 없고 커서의 사용 빈도를 줄이기 위해 CASE 표현식의 사용을 권장하며 종료시에 END의 사용에 주의 하시기 바랍니다.**

- **syntax**

```
CASE  expression
  WHEN  expression  THEN  expression
  [ ELSE  expression ]
END
```

- **예)**

```
SELECT id,
 ( case name
   when 'Tee Shirt' then 'Shirt'
   when 'Sweatshirt' then 'Shirt'
   when 'Baseball cap' then 'Hat'
   else 'Unknown'
  end ) as Type
FROM product;
```

```
SELECT name,
    ( case
        when id='1' then 'Shirt'
        when id='2' then 'Shirt'
        when id='3' then 'Hat'
        else 'Unknown'
       end ) as Type
FROM product;
```

loop

- 가장 간단한 루프로 **LOOP ~ END LOOP** 사이에 반복되는 문장들로 이루어져 있으며 탈출조건을 제시하지 않으면 무한루프가 되어 시스템에 많은 부하를 주기 때문에 반드시 특정한 조건을 주고 **LEAVE** 문을 통해 루프를 탈출할 수 있도록 해야 합니다.

- **syntax**

```
[ statement-label : ]
LOOP
      statement-list;
END LOOP [ statement-label ]
```

- **예)**

```
………
SET i = 1;
insert_loop:
LOOP
     INSERT INTO counter (number) VALUES (i);
     IF i >= 10 THEN
          LEAVE insert_loop;
     END IF;
     SET i = i + 1;
END LOOP insert_loop
```

// 위 예는 오직 **LOOP**문을 위한 예로 실제 이런 입력문의 사용을 자제바랍니다.

while loop

- 조건이 **TRUE** 인 동안만 일련의 문장을 반복하기 위해 사용하며 조건은 반복이 시작될 때 체크하게 되어 루프내의 문장이 한번도 수행되지 않을 경우도 있습니다.

- 루프를 시작할 때 조건이 **FALSE** 이면 루프를 탈출하게 됩니다.

- **syntax**

```
[ statement-label : ]
WHILE search-condition LOOP
    statement-list;
END LOOP [ statement-label ]
```

- 예)

```
SET i = 1;
insert_loop:
WHILE i <= 10 LOOP
    INSERT INTO counter (number) VALUES (i);
    SET i = i + 1;
END LOOP insert_loop
```

// 위 예는 오직 WHILE LOOP문을 위한 예로 실제 이런 입력문의 사용을 자제바랍니다.

cursor for loop

- 일반적으로 많이 사용하는 **index counter**를 증가시켜 가며 특정 문장을 반복하는 **FOR LOOP**와 다르며 오직 커서와 함께 사용하여 **FETCH** 로직을 커서의 마지막 데이터까지 반복적으로 사용하고 자동으로 커서가 **close** 되는 등 편리한 면이 있습니다.

- **BEGIN ~ END** 사이의 **compound statement** 내 에서만 사용 가능합니다.

- **syntax**

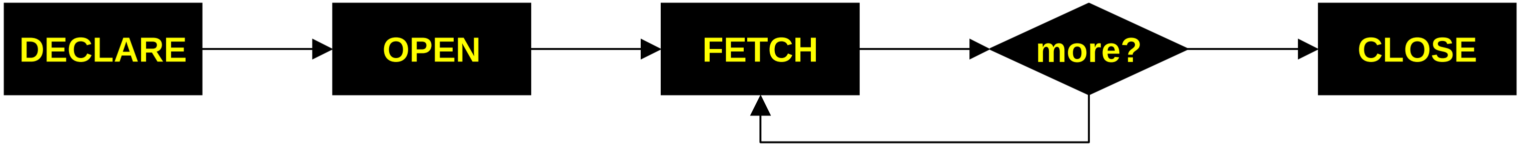
```
FOR for-loop-name AS cursor-name
CURSOR FOR statement
DO
     statement-list;
END FOR
```

- 예)

```
CREATE VARIABLE v_emp_name CHAR(30);
FOR names AS curs
CURSOR FOR SELECT emp_name
                     FROM employee
DO
      SET v_emp_name = emp_name;
      CALL search_for_name_proc ( v_emp_name );
END FOR;
```

cursor

- **SELECT** 문에 의해 산출되는 다중 행의 결과를 개별적으로 처리하기 위해 사용하는 것이 커서입니다
- 질의에 의해 산출된 첫번째 행부터 하나씩 처리할 수 있으며 현재 처리되는 행의 트랙을 유지하고 개발자가 수동으로 제어할 수 있습니다.
- 데이터베이스 특히 **DW**와 같이 대용량일 경우에는 커서의 사용으로 인해 부담해야 하는 비용이 매우 크기 때문에 꼭 필요한 경우를 제외하고 커서 사용을 최소로 줄여주십시오.
- 커서의 흐름



● **DECLARE :** 명시적으로 커서를 선언하기 위해 커서를 위한 **SELECT**와 함께 사용합니다.

DECLARE *cursor-name [no scroll | dynamic scroll | scroll]*
CURSOR FOR *select-statement*;

cursor

- **OPEN :** 질의를 수행하고 검색 조건을 충족하는 모든 행으로 구성된 결과 셋을 생성하기 위해 사용하며 이후 커서는 결과 셋에서 첫번째 행을 가리킵니다.

  OPEN *cursor-name*;

- **FETCH :** 결과 셋에서 하나의 행을 읽고 그 후에 커서는 결과 셋의 다음 행으로 이동 보통 여러 번의 **FETCH**가 이루어지는 관계로 **LOOP**문에서 주로 처리가 됩니다.

  FETCH *cursor-name* **INTO** *variable-list*;

- **CLOSE :** 커서를 더 이상 사용할 수 없게 하고 결과 셋을 해제합니다.

  CLOSE *cursor-name*;

■ 위 처럼 커서를 선언해서 사용시 사용자가 수동으로 모든 것을 처리하는 대신 **for loop**와 같이 사용하는 **cursor for loop** 구조를 사용하면 루프내에서 데이터가 자동으로 **FETCH** 되고 모든 행이 처리되면 자동으로 커서가 종료되므로 사용하기가 편리합니다.

cursor

- 예)

```
CREATE PROCEDURE cur_proc()
BEGIN
    DECLARE err_not_found EXCEPTION FOR SQLSTATE ‘02000’;
    DECLARE reg            CHAR(20);
    DECLARE cyear          CHAR(4);
    DECLARE sqty           DECIMAL(12);
    DECLARE test_cur CURSOR FOR
           SELECT cust_reg, order_year, sum(order_qty)
           FROM customer a, order_detail b
           WHERE a.cust_id = b.cust_id
           GROUP BY cust_reg, order_year
           FOR READ ONLY;
    OPEN test_cur;
    curloop:
    LOOP
           FETCH  test_cur INTO reg, cyear, sqty;
           IF SQLSTATE = err_not_found
                 LEAVE curloop;
           END IF;
    END LOOP curloop;
    CLOSE test_cur;
END
```

resume

- **dbisql**과 같은 툴에서는 **SELECT** 한 결과를 화면상에 출력할 때 조건에 만족하는 모든 데이터를 한번에 출력하지 않고 그 결과의 일부분만을 먼저 출력하는 커서형태의 방식을 적용합니다**.**

- 그래서 커서가 **OPEN**된 상태에서는 해당되는 테이블에 대한 어떠한 종류의 **DDL** 문을 수행할 수 없습니다**.**이럴때 그 커서를 **CLOSE** 해주는 역할을 하는 명령어가 **RESUME** 입니다**.**

- 마찬가지로 **stored procedure**도 똑같이 적용이 되어 실행이 종료된 것처럼 보이는 **stored procedure**도 **RESUME**를 해주기 전까지는 **DROP** 이나 **ALTER** 명령어를 수행할 수 없습니다**.**

- 예**)**
  SELECT emp_name FROM employee
  SELECT name FROM product;
  dbisql과 같은 툴에서 위 처럼 한 배치에서 두 개의 **SELECT**문을 사용했다면 첫번째 결과가 기본적으로 보여지고 **RESUME** 을 해야만 두 번째 결과를 볼 수 있습니다**.**

- 참고
  SELECT를 수행한 후에 **RESUME** 만 실행한 상태는 **CURSOR**가 닫혀있는 상태지만 아직까지도 **SELECT**한 테이블에 대한 **shared lock**을 **release**하지는 않았습니다**.**이 **lock**까지를 **release** 하려면 **RESUME** 후에 **COMMIT**을 실행하면 됩니다**.**

set option

- 데이터베이스내에 디폴트로 선언되어진 여러 가지 옵션을 변경할 수 있습니다. 대상이 되는 옵션에는 **IQ** 영역에서만 적용되는 옵션, **ASA** 영역에서만 적용되는 옵션, **dbisql** 영역에서만 적용되는 옵션, **ESQLC** 영역에서만 적용되는 옵션 등이 있으며 이에 대한 자세한 내용은 메뉴얼을 참고 하시기 바랍니다.

- 연결중인 세션에서만 바꾸려면 **TEMPORARY**라는 **keyword**를 사용해야 하며 생략시는 **PERMANENT** 하게 변경합니다.

- **user-name**을 생략하면 현재 이 명령어를 사용중인 사용자가 디폴트이며 모든 사용자에게 적용시에는 **PUBLIC**이라는 **group-name**을 사용하면 됩니다.

- **syntax**

  SET ***[****temporary****]*** **OPTION** *[user-name|public.]option-name = value*;

  SET OPTION PUBLIC.Load_Memory_MB = 400;
  SET TEMPORARY OPTION Query_Plan = ‘On’;
  SET TEMPORARY OPTION Temp_Extract_Name1 = ‘/asiq/data/test.dat’;

주석

- **Double Hyphen(--)**
  한 줄에서 이 문자 뒤의 문자를 주석 처리합니다

- **Double Slash(//)**
  한 줄에서 이 문자 뒤의 문자를 주석 처리한다**. --** 와 동일합니다

- **Slash-Asterisk(/*   */)**
  /* 와 ***/** 사이의 모든 문자는 줄에 관계없이 주석처리 합니다

- **Percent sign(%)**
  Percent_as_comment 옵션의 값이 **ON** 이면 **–** 와 동일합니다

dynamic SQL

- **SQL keyword나 object의 이름을 변수로 받아서 실행시에 다이나믹하게 새로운 SQL를 만들어 낼 수 있습니다.**

- **EXECUTE IMMEDIATE 문에 사용되어진 변수들은 string 형태여야 하며 2건 이상의 결과 셋을 return하는 명령어는 사용할 수 없습니다.**

- **BEGIN ~ END 사이의 compound statement 내 에서만 사용 가능합니다.**

- **syntax**

  EXECUTE IMMEDIATE *string-expr;*

- **예)**

```
CREATE PROCEDURE CreateTable_proc( in v_tablename char(30) )
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || v_tablename ||
        ' ( column1 int,………..)';
END
```

message

- 메시지를 출력하기 위해 사용되며 메시지가 출력되는 곳을 **CONSOLE**과 **LOG**로 지정하면 **server log** 파일이고 **CLIENT**로 지정하면 **dbisql**과 같은 윈도우를 지원하는 클라이언트입니다. 단, **server log = $SYBASE/ASIQ-12_5/logfiles/*srvname.xxx.srvlog***

- 또한 **MESSAGE TYPE**을 **INFO, STATUS**라 하면 **Message Window**에, **ACTION, WARNING**이라 하면 **Message Box**에 해당 메시지가 출력됩니다.

- **syntax**

```
MESSAGE expression
   [ TYPE { INFO | ACTION | WARNING | STATUS } ]
   [ TO { CONSOLE | CLIENT | LOG } ]
```

- **예)**

```
MESSAGE ‘The current date and time: ’, Now()  TYPE INFO TO CLIENT;
```

- **참조 : SQL사용 중에 UNIX 명령어 사용하여 Message 출력**

```
CREATE VARIABLE MsgText  VARCHAR(255);
CREATE VARIABLE CurrTime VARCHAR(30);
SET CurrTime = CAST(NOW() AS VARCHAR(30));
SET MsgText = ‘echo the message you want to send to the file '
                                        || ' at ' || CurrTime || ‘ >> /tmp/outputfile.txt';
CALL xp_cmdshell(MsgText);                         // OS 명령어를 수행하는 SP
```

exception

- **EXCEPTION** 이란 **compound statement** 실행 중에 발생하는 오류를 의미하며 이런 **EXCEPTION**은 **SYBASE IQ** 엔진에 의해 생성될 수도 있고 사용자가 임의적으로 만들어 발생 시킬 수도 있습니다.

- **compound statement** 수행시에 오류가 발생하면 그 시점에서 실행을 중단하고 관련된 예외 처리기 부분으로 제어를 옮겨 조건에 따라 마지막 조치 작업을 할 수 있도록 도와주는 구조가 **exception handler** 입니다.

- 이렇게 **exception handler**를 사용하려면 미리 자동으로 정의된 예외를 이용하거나 아니면 사용자가 임의적으로 **DECLARE** 문을 이용하여 예외를 선언하여야 하며 또한 선언된 각각에 대한 처리 루틴을 **EXCEPTION**문에 기술해야 합니다.

- 오류가 발생했을 때 **SQLCODE, SQLSTATE** 두 함수를 사용하여 관련된 오류 코드 및 상태를 확인할 수 있습니다.

- **syntax**

DECLARE *exception-name* **EXCEPTION FOR SQLSTATE** *sqlstate-number*;

```
EXCEPTION      [ WHEN exception-name THEN statement-list; ……]
               [ WHEN exception-name THEN statement-list; ……]
               WHEN OTHER THEN statement-list;
```

exception

- 예)

```
BEGIN
    DECLARE column_not_found
                EXCEPTION FOR SQLSTATE ‘52003’;                     // 예외선언
    MESSAGE ‘Hello !!!’ TO CLIENT;
    SIGNAL column_not_found;                                        // 예외발생
    MESSAGE ‘Line following signal !!!’ TO CLIENT;                  // 수행안됨

    EXCEPTION                                               // 예외처리
         WHEN column_not_found THEN                                 // 52003 예외이면
                MESSAGE ‘52003 handling. (SQLSTATE = ’,
                           SQLSTATE, ‘)’  TO CLIENT;
         WHEN others THEN                                           // 그 외 예외이면
                MESSAGE ‘Others handling. (SQLSTATE = ’,
                           SQLSTATE, ‘)’  TO CLIENT;
END
```

exception

- 사용자가 임의의 예외(SQLSTATE) 경우를 만들어 예외 처리를 수행할 수 있으며 이 때 주의 할 점은 선언 가능한 SQLSTATE 번호는 99000 ~ 99999 이라는 사실입니다. 그 외의 번호는 시스템이 예약한 번호들이기 때문입니다.

- 예)

```
BEGIN
    DECLARE my_exception
             EXCEPTION FOR SQLSTATE ‘99000’;        // 사용자 정의 예외선언
    MESSAGE ‘User-Defind Exception test !!!’ TO CLIENT;
    // 여기에 사용자 정의 예외를 만드는 조건절을 입력
    SIGNAL my_exception;                            // 사용자 정의 예외발생
    MESSAGE ‘Line following signal !!!’ TO CLIENT;  // 수행안됨

    EXCEPTION                                 // 예외처리
        WHEN my_exception THEN                      // 사용자 정의 예외이면
             MESSAGE ‘my_exception handling. (SQLSTATE = ’,
                      SQLSTATE, ‘)’  TO CLIENT;
        WHEN others THEN                            // 그 외 예외이면
             MESSAGE ‘Others handling. (SQLSTATE = ’,
                      SQLSTATE, ‘)’  TO CLIENT;
END
```

signal

- 시스템에서 제공된 **SQLSTATE** 혹은 사용자가 만든 **SQLSTATE**를 발생시키는데 사용됩니다.

- 이 명령어는 **compound statement** 내 에서만 사용가능하며 먼저 **DECLARE**문에 의에 **sqlstate-number**에 해당하는 **exception-name**을 선언한 후에 사용해야 합니다.

- 시스템이 제공하지 않는 사용자가 선언할 수 있는 **SQLSTATE** 번호는 **99000 ~ 99999** 입니다.

- **syntax**

```
DECLARE exception-name EXCEPTION FOR SQLSTATE sqlstate-number;

SIGNAL exception-name;
```

- 예)

```
BEGIN
    DECLARE column_not_found
                EXCEPTION FOR SQLSTATE ‘52003’;
    SIGNAL column_not_found;
END
```

traceback

- **compound statement 에서 오류가 발생했을 때 가장 최근에 발생한 오류에 대한 부분을 지적해 줍니다.단, syntax 오류는 제외됩니다.**

- **syntax**

  TRACEBACK(*)

- **예)**

```
BEGIN
     .
     .
     .
     SELECT prod_name FROM employee;
     .
END
SELECT traceback(*);
```

기타

- SELECT statement
- INSERT statement
- UPDATE statement
- DELETE statement

- GRANT statement
- REVOKE statement

- CREATE TABLE | PROCEDURE | FUNCTION | INDEX statement

- BEGIN TRANSACTION statement
- COMMIT, ROLLBACK statement
- SAVEPOINT statement
- RELEASE SAVEPOINT statement

- LOAD TABLE statement
- ……..

- 위 내용들은 계속해서 진행하는 다른 장에서 언급하도록 하겠습니다.

Selecting data from a table

overview

- **syntax**

```
SELECT          select-list                         // select 절
[ FROM          table-expression ]                  // from 절
[ ON            join-condition ]                    // on 절
[ WHERE         search-condition ]                  // where 절
[ GROUP BY      column-list ]                       // group by 절
[ HAVING        search-condition ]                  // having 절
[ ORDER BY      column-list ]                       // order by 절
```

- **SELECT 문에 사용되어지는 SQL Keyword는 대/소문자 구별을 하지 않습니다. 하지만 일반적으로 대문자로 많이 표현하고 있습니다.**

- **SELECT문을 형성하는 SELECT, FROM, WHERE 절과 같은 각 절은 라인을 분리하여 읽기 편하게 사용합니다.**

- **테이블명, 컬럼명과 같은 Identifier는 대소문자 구별을 하지 않지만 테이블의 데이터나 string과 같은 데이터는 데이터베이스 생성시의 옵션에 의해 구별 여부가 결정됩니다.**

- **테이블과 같은 Object 참조시 모호성을 제거하기 위해 owner 명과 같은 qualifier를 사용할 수 있습니다.**

select 절

- **select-list**는 일반적으로 콤마로 분리된 액세스를 원하는 컬럼**,** 산술식**, aggregate** 함수**,** 혹은 모든 컬럼을 지칭하는 *****로 구성되어 집니다**.**

- 질의 결과에 따르는 각 컬럼의 헤딩은 **select-list**에 사용되어지는 **expression**을 디폴트로 사용하는데 이런 헤딩을 변경하기 위해 다음과 같은 **alias**를 사용 하기도 합니다**.**

SELECT *column-name* ***AS alias***
SELECT *column-name* ***alias***
SELECT ***alias* =** *column-name*

- **alias**에 **space** 또는 **keyword**를 사용하고자 한다면 **“**로 양쪽을 감싸주어야 합니다**.**

SELECT ***dept_id* AS “*integer*”**
FROM …

- **select-list**에 데이터가 출력되는 방식을 변경하거나 사칙연산과 같은 계산을 수행할 때 산술 표현식을 사용할 수 있습니다**.**

SELECT name, (quantity * unit_price) – 5
FROM …

- **concatenation** 연산자 **||** 를 사용하여 문자열들을 또 다른 문자열로 만들 수 있습니다**.**

SELECT f_name || l_name “성명**”**
FROM …

select 절

- **select-list**에 사용하는 리터럴은 **alias**나 컬럼명이 아닌 **select** 절에 포함된 문자**,** 표현식**,** 숫자이지만 컬럼과 같이 취급됩니다**.** 날짜와 문자 리터럴은 단일 인용부호**(‘ ‘)**로 에워싸며 숫자 리터럴은 그냥 숫자만 표현하면 됩니다

```
SELECT  l_name || l_name || ‘  ’ || title “직원” // space
FROM …
```

- **NULL**이란 **0**이나 **space**와 달리 알려지지 않은 또는 적용 불가능한 값을 의미하며 이러한 **NULL**값을 표현한 산술식의 결과는 **NULL** 입니다**.** 그래서 일반적으로 **select-list**에 널값 대신에 다른 값으로 대치하기 위해서 **COALESCE**라는 함수를 사용합니다**.**

```
SELECT f_name || l_name, salary * COALESCE(comm,0)/100
FROM …
```

- **SELECT**한 결과로 부터 중복된 행을 제거하기 위해 선택적으로 **DISTINCT keyword** 를 사용할 수 있으며 **NULL**값은 실제 **SELECT**한 결과가 몇건인지에 관계없이 한 건으로 취급 됩니다**.** 또한 **DISTINCT**라는 **keyword**는 항상 **SELECT** 바로 뒤에 기술하며 그 뒤에 여러 개의 컬럼을 기술했다면 나타나는 로우는 컬럼의 조합들이 중복되지 않게 나타납니다**.**

```
SELECT DISTINCT city
FROM …
```

first, top

- 질의의 결과로 부터 첫번째 혹은 기술된 수만큼의 데이터만 반환합니다. **FIRST**은 한 건의 데이터만 반환하며 **TOP**은 **1~32767** 까지의 데이터만 반환합니다.

- 주로 **ORDER BY** 절과 함께 사용되어야 원하는 데이터를 얻을 수 있지 만약에 **ORDER BY**절 없이 사용한다면 매 질의시 다른 결과를 산출할 수 있습니다.

- 이 함수는 질의의 제일 바깥쪽 **SELECT**문에서만 사용 가능 하므로 **derived table**이나 **VIEW**의 생성에는 사용할 수 없습니다.

- **FIRST**는 **ROW_COUNT**라는 옵션의 값을 **1**로 설정하고, **TOP n** 은 **ROW_COUNT** 옵션을 **n**으로 설정하고 질의한 것과 같습니다. 그러나 **TOP n**에서 **n**의 범위는 **1~32767**이지만 **ROW_COUNT**에서는 그런 제약이 없습니다.

- **syntax**
  SELECT [FIRST | TOP *number-of-rows* **]** *select-list*

from 절

- select 절의 select-list에 컬럼의 기술이 전혀 없고 단지 산술 표현식만 사용되었다면 from 절 이하는 생략되거나 sys.dummy 테이블을 사용할 수 있습니다.

```
SELECT  24 * 60 * 60
[FROM sys.dummy];
```

- 테이블이나 뷰를 통해 데이터를 SELECT 할 때 반드시 필요하며 콤마를 이용하여 여러 테이블 혹은 derived table을 나열할 수 있습니다.

```
SELECT …
FROM  employee | ( SELECT … … ) b
```

- 테이블이름이나 뷰 이름은 모호성을 제거하기 위해 owner 명과 같은 qualifier를 사용할 수 있습니다.

```
SELECT …
FROM dba.employee
```

- 타이핑의 수고를 줄이기 위해 테이블 명에 다른 correlation name을 테이블 이름 뒤에 사용할 수 있으며 그렇게 correlation name이 한 번 사용되어진 후에 그 테이블을 참조하는 표현으로는 반드시 correlation name이 사용되어야 합니다.

```
SELECT   d.dept_id, d.dept_name               //d 대신에 department 사용불가
FROM     department d
```

where 절

- **where절에는 SELECT 하는 로우를 제한하기 위한 search condition이 포함될 수 있으며 qualification 혹은 predicate라고도 불리며 다음과 같은 여러 조건절이 사용됩니다**

  - **Comparison operator : =, > , <, <=, >=, !=, !>, !<**

```
SELECT emp_lname
FROM employee
WHERE salary > 50000
```

  - **Range operator : BETWEEN, NOT BETWEEN**

```
SELECT emp_lname
FROM employee
WHERE salary BETWEEN 40000 AND 50000
```

  - **List operator : IN, NOT IN**

```
SELECT company_name, state
FROM customer
WHERE state IN (‘ON’, ‘PQ’, ‘MB’)
```

where 절

- Character match : LIKE, NOT LIKE

```
SELECT company_name, phone
FROM customer
WHERE phone LIKE ‘415%’
```

- Unknown value : IS NULL, IS NOT NULL

```
SELECT dept_name
FROM department
WHERE dept_head_id is NOT NULL
```

- Combination : AND, OR, NOT

```
SELECT emp_fname, emp_lname
FROM employee
WHERE saraly > 50000
AND emp_fname like ‘A%’
```

Summarizing, Grouping, Sorting

group by 절

- 테이블에 있는 데이터를 작은 그룹으로 분류하여 보고자 할 때 사용하며 일반적으로 **aggregate** 함수와 함께 사용됩니다**. select** 절에 나타난 어떠한 컬럼이나 표현식도 **aggregate** 함수가 아닌 경우에는 **group by**절에 나타나야 합니다**.**

- **group by** 절 없이 **aggregate** 함수가 사용된다면 전 테이블을 하나의 그룹으로 취급하고 그에 대한 한건의 결과만을 산출합니다**.** 이 때 **select-list**에는 **aggregation** 함수만 기술될 수 있습니다**.**

- **aggregate** 함수는 **select** 절과 **having** 절에서 사용 가능하고 **NULL** 값은 **COUNT(*)**를 제외한 연산에서 제외되며 다음과 같은 함수들이 있습니다
  - **AVG(***numeric-expr***)**
  - **SUM(***numeric-expr***)**
  - **COUNT(*)**
  - **COUNT(***column-name***)**
  - **MAX(***expr***), MIN(***expr***)**
  - **VARIANCE (***numeric-expr***), STDDEV(***numeric-expr***)**

- 예)

```
SELECT    order_reg, order_prod, SUM(order_qty)
FROM      order_detail
WHERE     order_date >= ‘1999/01/01’
GROUP BY order_reg, order_prod
```

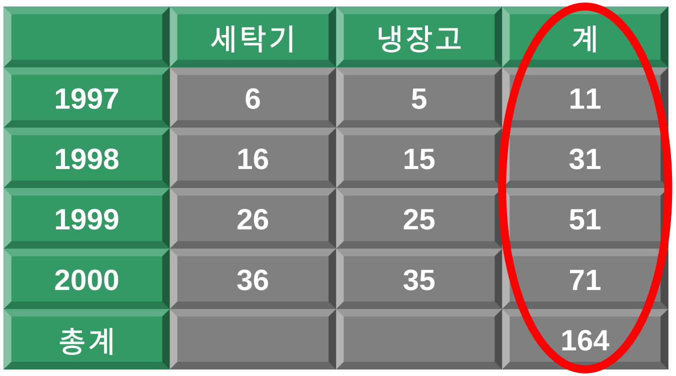
rollup

- **ROLLUP은 GROUP BY절과 함께 사용하여 뒤에 나오는 (컬럼수 – 1) 개 까지의 그룹별 소계와 총계를 산출합니다.**

- **현재까지는 COUNT DISTINCT, SUM DISTINCT는 지원하지 않고 있으며 sub-query에서는 사용 불가능 합니다.**

- **syntax**
  SELECT [GROUPING (*column-name*)…] …
  GROUP BY ROLLUP *expression* [,…..]

- **예)**
  SELECT year, model, sum(qty)
  FROM sales
  GROUP BY ROLLUP year, model



	세탁기	냉장고	계
1997	6	5	11
1998	16	15	31
1999	26	25	51
2000	36	35	71
총계			164

cube

- **ROLLUP은 GROUP BY절과 함께 사용하여 rollup 기능에 큐브 형태의 그룹별 소계와 총계를 산출합니다.**

- **현재까지는 COUNT DISTINCT, SUM DISTINCT는 지원하지 않고 있으며 sub-query에서는 사용 불가능 합니다.**

- **syntax**
  SELECT [GROUPING (*column-name*)…] …
  GROUP BY CUBE *expression* [,…..]

- **예)**
  SELECT year, model, sum(qty)
  FROM sales
  GROUP BY CUBE year, model

	세탁기	냉장고	계
1997	6	5	11
1998	16	15	31
1999	26	25	51
2000	36	35	71
총계	84	80	164

having 절

- **where 절에서 SELECT 하기 위한 로우를 제한 하듯이 GROUP BY 한 결과를 제한하고자 할 때 사용합니다. 즉 aggregation 함수를 사용하는 제한은 문법상 where절에 기술할 수 없습니다. 그러므로 그룹에 대한 제한은 where절이 아닌 having절에 기술합니다.**

- **예)**

```
SELECT     order_reg, order_prod, SUM(order_qty)
FROM       order_detail
WHERE      order_date >= ‘1999/01/01’
GROUP BY order_reg,order_prod
HAVING     SUM(order_qty) > 200
```

order by 절

- **SELECT**한 결과를 어떤 특정 컬럼에 의해 정렬해서 보여주기 위해 사용하며 오름차순과 내림차순으로 선택해서 보여줄 수 있습니다

- **ASC**를 사용해서 오름차순으로**, DESC**를 사용해서 내림차순으로 정렬해서 보여줄 수 있으며 **ASC** 혹은 **DESC**를 생략하면 디폴트로는 **asc** 입니다

- 하나 이상의 컬럼으로 질의 결과를 정렬할 수 있으며 이 때 **order by**절의 컬럼은 콤마로 분리되어야 하며 컬럼 대신 별명이나 **select** 절에서 기술된 컬럼의 위치순서를 기술해도 됩니다**.**

- 일반적으로 다른 **RDBMS**에서는 **GROUP BY**의 결과로 **group by**절에 기술된 컬럼 순으로 자동으로 정렬해 주기 때문에 다른 컬럼 순으로 정렬하고자 했을 때를 제외하고는 **order by**절을 생략하지만 **SYBASE IQ**에서는 **order by** 절을 사용하지 않으면 어떠한 종류의 질의도 결코 정렬해서 볼 수가 없습니다**.**

- 예**)**

```
SELECT    id, name, quantity
FROM      product
WHERE     name like '%shirt%'
ORDER BY name, quantity desc 혹은 ORDER BY 2, 3 desc;
```

분석형 함수

- 전체 그룹에 대한 순위를 내주는 순위형 함수에는 **DENSE_RANK, PERCENT_RANK, RANK**가 있으며 순위에 대한 기준이 되는 **OVER (ORDER BY)** 절을 필요로 합니다.

- 같은 순위를 어떻게 처리하는지 여부에 따라 **DENSE_RANK**와 **RANK**는 다릅니다. 즉 **RANK** 함수는 같은 순위가 존재하면 존재하는 수만큼의 공백 순위가 존재하나 **DENSE_RANK**는 공백순위를 인정하지 않습니다.

- **PERCENT_RANK** 함수는 순위 백분율을 계산해주며 **NTILE** 함수는 분위수를 계산해 줍니다. 또한 비슷한 방법의 **PERCENT_CONT, PERCENT_DISC**가 있습니다.

- 위 함수들은 **sub-query**에서 사용할 수 없으며 그룹별로 별도의 순위를 만들어주는 옵션인 **PARTITION BY** 가 현재 까지는 제공되지 않습니다.

- **syntax**

```
SELECT  DENSE_RANK | PERCENT_RANK | RANK
               OVER ( ORDER BY expr [ ASC | DESC ] )

SELECT  NTILE (expr1) OVER ( ORDER BY expr2 [ ASC | DESC ] )

SELECT  PERCENT_CONT | PERCENT_DISC (expr1)
               WITHIN GROUP ( ORDER BY expr2 [ ASC | DESC ] )
```

분석형 함수

- 참고 : PARTITION BY 기능 구현 하기

```
SELECT district, grade, AVG(math_score),
       rank() over (partition by district order by avg(math_score) desc) as rank_d
FROM math_report
GROUP BY district, grade
ORDER BY district;
```

```
SELECT district, grade, avg(math_score),
       rank() over (order by avg(math_score) desc) as rank_d
 FROM math_report WHERE district = 'essex'
 GROUP BY district, grade
 UNION ALL
SELECT district, grade, avg(math_score),
       rank() over (order by avg(math_score) desc) as rank_d
 FROM math_report WHERE district = 'middlesex'
 GROUP BY district, grade
 UNION ALL
SELECT district, grade, avg(math_score),
       rank() over (order by avg(math_score) desc) as rank_d
 FROM math_report WHERE district = 'suffolk'
 GROUP BY district, grade
 ORDER BY 1,2;
```

Retrieving data from several tables

overview

- syntax

```
SELECT          select-list                          // select 절
[ FROM          table-expression ]                   // from 절
[ ON            join-condition ]                     // on 절
[ WHERE         search-condition ]                   // where 절
[ GROUP BY      column-list ]                        // group by 절
[ HAVING        search-condition ]                   // having 절
[ ORDER BY      column-list ]                        // order by 절
```

```
FROM  table-name [ KEY | NATURAL [ INNER | FULL [ OUTER ] |
          LEFT [ OUTER ] | RIGHT [ OUTER ] JOIN | CROSS JOIN table-name
ON  join-condition
```

join

- 조인은 관계를 맺고 있는 두 개 이상의 테이블을 하나의 **select statement**에 포함하여 원하는 정보를 산출하는 **SQL** 이며 일반적으로 **PK**와 **FK**를 사용하여 조인하는 경우가 대부분이지만 때로는 논리적인 값들의 연관으로 조인하는 경우도 있습니다**.**

- 일반적으로 **where**절에 조인 조건을 기술하여 사용하지만 **WatcomSQL**과 **ANSI**에서는 **from**절에 조인의 방법을 **on**절에 조인의 조건을 기술하여 사용합니다**.**

- 일반적으로 조인할 때 똑같은 컬럼 이름이 존재하거나 문장의 명확성을 위해 컬럼 이름 앞에 테이블 이름이나 혹은 **correlation name**을 붙여서 사용합니다**.**

- 일반적으로 **N**개의 테이블을 조인하려면 **N-1**개의 조인 조건이 필요합니다**.**

- 참고
  조인에 해당되는 컬럼은 데이터 타입 및 길이가 같아야 최대 성능을 발휘 합니다**.** 특히 **SYBASE IQ**에서는 조인에 해당되는 컬럼의 데이터 타입을 **int** 계열 특히 **unsigned int** 계열로 만들어 사용하시면 최대 성능을 발휘합니다**.**

equijoin vs. non-equijoin

- **equijoin**은 조인 조건에서 **=**을 사용하여 값들이 정확하게 일치하는 경우에 사용하는 조인을 말하며 대부분 **PK**와 **FK**의 관계를 이용하여 조인합니다**. equijoin**은 다른 말로 **inner-join** 이라고도 합니다**.**

- 예)

```
SELECT *
FROM sales_order a JOIN customer b ON a.cust_id = b.id
                   JOIN sales_order_items c ON a.id = c.id
```

- 조인 조건에서 연산자로 **=**이 아닌 **BETWEEN ~ AND** 등과 같은 연산자가 사용되어 진다면 이를 **non-equijoin** 이라고 합니다**.**

- 예)

```
SELECT e.emp_id, e.emp_lname, e,salary, s.low_salary, s.high_salary
FROM salary_grade s JOIN employee e
ON e.salary BETWEEN s.low_salary AND s.high_salary;
```

inner-join vs. outer-join

- **equijoin**은 조인 조건에서 **=**을 사용하여 값들이 정확하게 일치하는 경우에 사용하는 조인을 말하며 대부분 **PK**와 **FK**의 관계를 이용하여 조인합니다**. equijoin**은 다른 말로 **inner-join** 이라고도 합니다**.**

- 예**)**

```
SELECT *
FROM sales_order a JOIN customer b ON a.cust_id = b.id
                        JOIN sales_order_items c ON a.id = c.id
```

- 일반적인 **inner join**에서는 테이블들 사이에 조인 조건을 만족하는 로우만을 결과로 산출하는데 반해 조인 조건을 만족하지 않는 로우라도 전부 출력하고자 할 경우에 사용되는 조인을 **outer join**이라고 말하며 전부 출력하고자 하는 테이블을 **outer,** 그 반대쪽을 **inner** 라고 합니다**.** 전부 출력되는 테이블의 위치에 따라 **LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN**이 있습니다**.**

- 예**)**

```
SELECT lname, order_date, city
FROM customer LEFT OUTER JOIN sales_order
ON customer.id = sales_order.cust_id
WHERE customer.state = ‘NY’;
```

key join

- 테이블들간의 조인은 미리 정의되어진 **PK, FK**에 의해 조인 조건이 자동으로 생성되므로 **ON**절이 생략됩니다. 반드시 사전에 조인되어지는 테이블에 **PK, FK**를 선언해야 합니다

- 예)

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FRO M    customer a   KEY JOIN   order_detail b
WHERE  a.cust_reg = ‘Seoul’
AND        b.order_proc = ‘101’
```

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM     customer a, order_detail b
WHERE   a.cust_id = b.cust_id
AND        a.cust_reg = ‘Seoul’
AND        b.order_proc = ‘101’
```

natual join

- 테이블들간의 이름이 같은 컬럼들로 조인 조건이 자동으로 생성되므로 **ON**절이 생략됩니다. 테이블들간의 이름이 같은 컬럼이 최소 하나 이상이 존재해야 하며 만약 일치하는 이름이 없다면 오류가 발생합니다.

- 예)

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM    customer a  NATURAL JOIN  order_detail b
WHERE   a.cust_reg = ‘Seoul’
AND     b.order_proc = ‘101’
```

↓

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM    customer a, order_detail b
WHERE   a.cust_id = b.cust_id
AND     a.cust_reg = ‘Seoul’
AND     b.order_proc = ‘101’
```

general join

- 가장 일반적으로 사용되어지는 형태로 조인 관련된 컬럼 정보를 on절에 직접 기술합니다

- 예)

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM    customer a   JOIN   order_detail b
ON      a.cust_id = b.cust_id
WHERE   a.cust_reg = ‘Seoul’
AND     b.order_proc = ‘101’
```

↓

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM    customer a, order_detail b
WHERE   a.cust_id = b.cust_id
AND     a.cust_reg = ‘Seoul’
AND     b.order_proc = ‘101’
```

cross join(cartesian product)

- 가능한 모든 경우를 조합하여 대단히 많은 결과를 산출하며 테이블들 사이의 조인절이 기술되지 않을 때 발생합니다.

- 조인 조건을 생략한 경우나 조인 조건이 잘못된 경우에 발생되며 첫번째 테이블의 모든 행이 두 번째 테이블의 모든 행과 조인이 됩니다

- 대부분의 경우 그 결과가 거의 유용하지 못하므로 **Max_Cartesian_Result** 라는 옵션을 통해 통제할 수 있습니다.

- 예)

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM     customer a   CROSS JOIN   order_detail b
```

⬇

```
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM    customer a , order_detail b
```

self join

- 같은 테이블을 조인에 이용하는 방법으로 반드시 **correlation name**을 사용해야 합니다.

- 예) 동명이인을 찾는 **self join**

```
SELECT  a.cust_id, a.cust_name, a.cust_addr
FROM    customer a JOIN customer b
ON      a.cust_name = b.cust_name
WHERE a.cust_id != a.cust_id
```

outer join

- 일반적인 **inner join**에서는 테이블들 사이에 조인 조건을 만족하는 로우만을 결과로 산출하는데 반해 조인 조건을 만족하지 않는 로우라도 전부 출력하고자 할 경우에 사용되는 조인을 **outer join**이라고 말하며 전부 출력하고자 하는 테이블을 **outer**, 그 반대쪽을 **inner** 라고 합니다.

- 주의할 점은 **inner** 테이블에 대한 제약 조건이 있다면 이 조건은 반드시 **where**절이 아닌 **on**절에 기술해야 합니다. 만약 **where** 절에 이 제약 조건을 기술하면 **OUTER JOIN**이 아닌 **INNER JOIN**으로 결과를 산출할 것 입니다.이는 **ANSI** 규약이며 **outer** 테이블에 대한 제약 조건은 어디에 기술해도 됩니다.

- 예)

```sql
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM     customer a  left outer join  order_detail b
ON         a.cust_id = b.cust_id
AND        b.order_qty > 10                    // 주의
WHERE   a.cust_reg = ‘seoul’
```

↓

```sql
SELECT  a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM     customer a , order_detail b
WHERE  a.cust_id *= b.cust_id
AND       a.cust_reg = ‘seoul’
AND       b.order_qty > 10
```

derived table join

- from절에서 뷰형태의 중간 테이블을 생성한 후 다른 테이블과 조인을 하는 형태로 편집 성능을 향상 시킬 수 있는 방법을 제공합니다. 단, select절 에서의 사용은 원래 ASA의 WatcomSQL에서는 가능하지만 SYBASE IQ에서는 문법상 불가능합니다.

- Oracle에서는 In-line-view 라고도 하며 대체할 수 있는 다른 방법으로는 임시 테이블을 만들어서 사용할 수 있습니다.

- 예)

```
SELECT   a.cust_fname, a.cust_lname, b.order_unit, b.order_qty
FROM     customer a , (SELECT   cust_id, order_unit, order_qty
                       FROM     order_detail
                       WHERE    order_prod = ‘101’) b
WHERE    a.cust_id = b.cust_id
AND      a.cust_reg = ‘Seoul’
```

- 참고

  derived table의 결과로 많은 수의 데이터가 산출된다면 이를 임시 테이블 형태로 바꾸고 인덱스를 생성하여 사용한다면 조인에 대한 성능이 향상될 것입니다. 이렇게 임시 테이블을 사용하여 편집할 때 불편한 점이 조금은 있을지 모르지만 성능상에는 많은 이점이 있습니다.

sub-query

- **sub-query**는 여러 절에서 사용 가능하며 **SELECT** 문장 안에 기술된 **SELECT** 문장입니다. **sub-query**는 **main query**이전에 한번만 수행되며 **sub-query**의 결과를 **main query**에 의해 조건으로 사용됩니다.

- **sub-query**는 괄호로 양쪽을 묶어야 하고 연산자의 오른쪽에 나타나야 하며 **order by** 절을 포함할 수 없습니다. 또한 **select-list**에 단지 하나의 컬러만을 사용할 수 있습니다.

- **sub-query**의 결과가 한 건의 데이터를 산출한다면 단일 행 연산자를 사용해야 하며 **2**건 이상의 데이터를 산출한다면 다중 행 연산자를 사용해야 합니다.
  - 단일 행 연산자 : **=, >, <, >=, <=, !=**
  - 다중 행 연산자 : **IN, EXISTS (**참고 **: ANY, ALL**은 지원하지 않습니다**)**

- **sub-query**는 **where**절 뿐만 아니라 **having**절**, from**절에서도 가능하지만 특히 **from**절 에서 사용하는 **sub-query**는 **derived-table**이라고 부릅니다.

- **syntax**

```
SELECT  select-list
FROM  table-expression
WHERE   expression  operator  ( SELECT select-list
                                FROM   table-expression
                                WHERE  search-condition )
```

sub-query

- 예)

```
SELECT   cust_fname, cust_lname
FROM     customer
WHERE    cust_id in ( SELECT   cust_id
                      FROM     order_detail
                      WHERE    order_proc = ‘101’)
AND      cust_reg = ‘Seoul’
```

↓

```
SELECT   c.cust_fname, c.cust_lname
FROM     customer c, order_detail o
WHERE    c.cust_id in = o.cust_id
AND      o.order_proc = ‘101’
AND      c.cust_reg = ‘Seoul’
```

union

- 연산자 양쪽의 질의에 대한 결과를 하나의 결과로 합하여 보여주는 연산자로 중복 데이터를 포함하여 보여줄 때는 **ALL** 이라는 **Keyword**가 추가적으로 덧붙여 집니다.

- 연산자 양쪽에서 각각의 결과를 산출하는 컬럼의 수 및 데이터 타입이 기본적으로 같아야 합니다.

- 연산자 양쪽의 질의 결과를 논리적으로 따져볼 때 중복 데이터가 존재하지 않는다면 실제 연산할 때 중복 데이터 제거 과정을 생략하므로 **UNION ALL**이 성능에 아주 좋습니다.

- **UNION** 질의 결과를 정렬해서 보고 싶다면 **UNION** 연산자의 아래 쪽의 질의에 **order by** 절을 추가 기술하면 됩니다. 단, 컬럼 이름이 아니고 컬럼의 위치순서를 기술해야 합니다.

- 예)

```
SELECT cust_name,cust_city,cust_phone
FROM a_customer
UNION [ALL]
SELECT cust_name,cust_city,cust_phone
FROM b_customer
ORDER BY 1, 2;
```

minus, intersect

- **MINUS에 대한 연산자는 직접 제공되지 않기 때문에 사용자가 다음의 보기처럼 질의를 직접 기술해야 합니다.**

- **예)**

```
SELECT product_id
FROM  ( SELECT a.product_id, b.product_id as b_product_id
              FROM inventories a LEFT OUTER JOIN order_item b
              ON a.product_id = b.product_id ) tmp
WHERE b_product_id  IS NULL;
```

- **INTERSECT에 대한 연산자도 역시 직접 제공되지 않기 때문에 사용자가 다음의 보기처럼 질의를 직접 기술해야 합니다.**

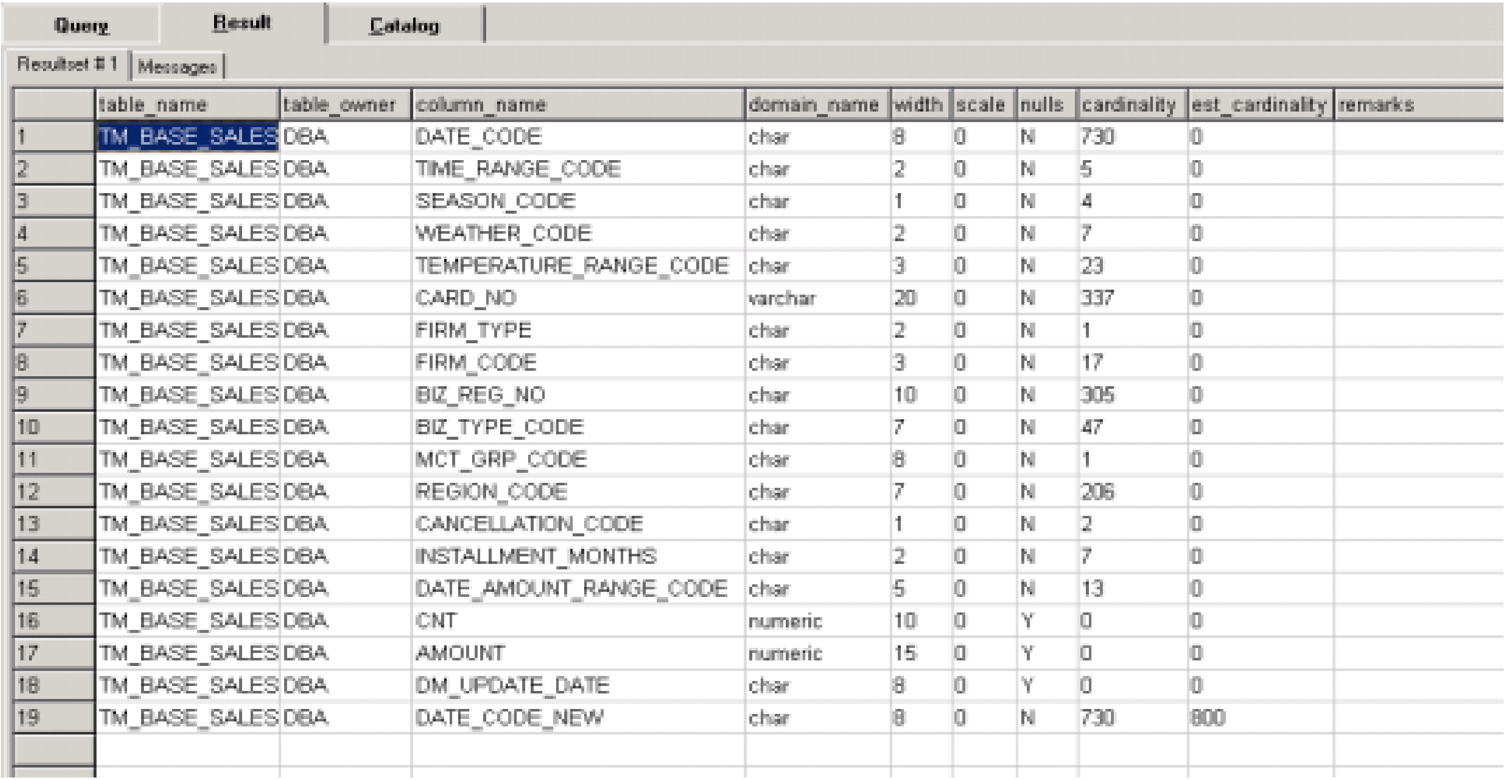
- **예)**

```
SELECT a.product_id
FROM  inventories a  JOIN order_item b
ON a.product_id = b.product_id;
```

sp_iqcolumn

- syntax

  sp_iqcolumn *table-name*;



	table_name	table_owner	column_name	domain_name	width	scale	nulls	cardinality	est_cardinality	remarks
1	TM_BASE_SALES	DBA	DATE_CODE	char	8	0	N	730	0	
2	TM_BASE_SALES	DBA	TIME_RANGE_CODE	char	2	0	N	5	0	
3	TM_BASE_SALES	DBA	SEASON_CODE	char	1	0	N	4	0	
4	TM_BASE_SALES	DBA	WEATHER_CODE	char	2	0	N	7	0	
5	TM_BASE_SALES	DBA	TEMPERATURE_RANGE_CODE	char	3	0	N	23	0	
6	TM_BASE_SALES	DBA	CARD_NO	varchar	20	0	N	337	0	
7	TM_BASE_SALES	DBA	FIRM_TYPE	char	2	0	N	1	0	
8	TM_BASE_SALES	DBA	FIRM_CODE	char	3	0	N	17	0	
9	TM_BASE_SALES	DBA	BIZ_REG_NO	char	10	0	N	305	0	
10	TM_BASE_SALES	DBA	BIZ_TYPE_CODE	char	7	0	N	47	0	
11	TM_BASE_SALES	DBA	MCT_GRP_CODE	char	8	0	N	1	0	
12	TM_BASE_SALES	DBA	REGION_CODE	char	7	0	N	206	0	
13	TM_BASE_SALES	DBA	CANCELLATION_CODE	char	1	0	N	2	0	
14	TM_BASE_SALES	DBA	INSTALLMENT_MONTHS	char	2	0	N	7	0	
15	TM_BASE_SALES	DBA	DATE_AMOUNT_RANGE_CODE	char	5	0	N	13	0	
16	TM_BASE_SALES	DBA	CNT	numeric	10	0	Y	0	0	
17	TM_BASE_SALES	DBA	AMOUNT	numeric	15	0	Y	0	0	
18	TM_BASE_SALES	DBA	DM_UPDATE_DATE	char	8	0	Y	0	0	
19	TM_BASE_SALES	DBA	DATE_CODE_NEW	char	8	0	N	730	800	

Adding, Changing, and Deleting data

insert

- 한 테이블에 **insert**가 진행되고 있을 때 어떤 다른 사용자도 그 테이블에 대해 **write** 할 수 있는 권한이 없습니다. 다만 **write** 이전 버전의 데이터를 **read** 할 수 는 있습니다.

- **syntax1**

  INSERT [INTO] *[owner.]table-name [(column-name[,…])]*
  VALUES (*expression* **…)**

- **syntax2**

  INSERT [INTO] *[owner.]table-name [(column-name[,…])]*
  insert-load-option
  select-statement

- **syntax3**

  INSERT [INTO] *[owner.]table-name [(column-name[,…])]*
  insert-load-option
  [LOCATION ‘*server-name.db-name’***]**
  {*select-statement***}**

insert-manually insert

- 주어진 값을 가지고 특정한 테이블에 한건 한건 데이터를 입력하며 주로 **OLTP**의 운영환경에서 많이 사용하는 방법입니다.

- **SYBASE IQ** 영역에 존재하는 테이블에 이런 식의 **INSERT**는 입력되는 속도도 느릴 뿐만 아니라 서버에 주는 부담도 매우 높습니다. 가능하면 이런 **INSERT**는 자제해 주십시오.

- 만약 사용시에도 컬럼과 값에 대한 매핑을 명시적으로 하십시오

- 예)

```
INSERT INTO department (dept_id, dept_name)
        VALUES (230, 'Eastern Sales');
```

insert-inserting from IQ main store

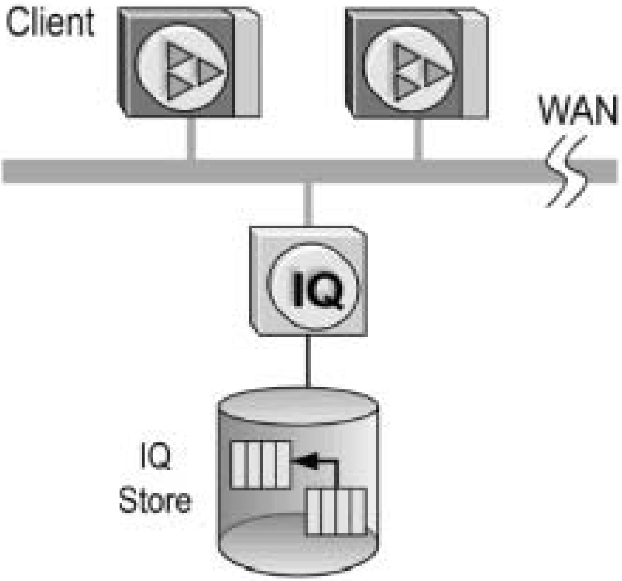
- 아래 그림에서 보듯이 **SELECT**하는 테이블도 **SYBASE IQ**의 **main store**에 존재하고 **INSERT** 하는 테이블도 **SYBASE IQ main store**에 존재합니다.

- **ETL** 과정 중에 많이 발생되는 형태의 입력 방법 이며 다중 쓰레드로 동작되는 **LOAD**와 달리 한 개의 쓰레드가 맡아서 처리합니다.

- 시간이 많이 소요되고 매우 중요한 트랜잭션이라면 경우에 따라 **SAM** 파일로 **UNLOAD** 한 후에 그 파일을 다시 **LOAD** 하는 것도 고려해 보십시오.

- 예)

```
INSERT INTO dept_head ( name, dept )
SELECT emp_lname || ‘ ‘ || emp_fname as name,
           dept_name
FROM   employee   JOIN   department
ON       emp_id = dept_head_id;
```



Insert-inserting directly from a foreign db

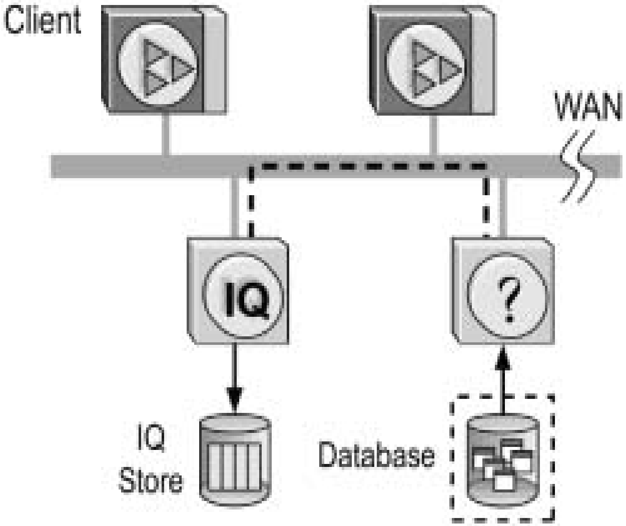
- 아래 그림에서 보듯이 현재 접속하고 있는 **SYBASE IQ**가 아닌 다른 **DB**의 테이블을 읽어서 현재 접속하고 있는 **SYBASE IQ**의 테이블에 입력할 때 사용합니다.

- 이렇게 사용가능한 **DB**로는 또 다른 **SYBASE IQ, ASE, Oracle, Informix** 등이 있습니다. 단 **SYBASE** 제품이 아닌 **Oracle, Informix**등은 중간에 **SYBASE**의 **Gateway** 제품이 있어야 가능합니다.

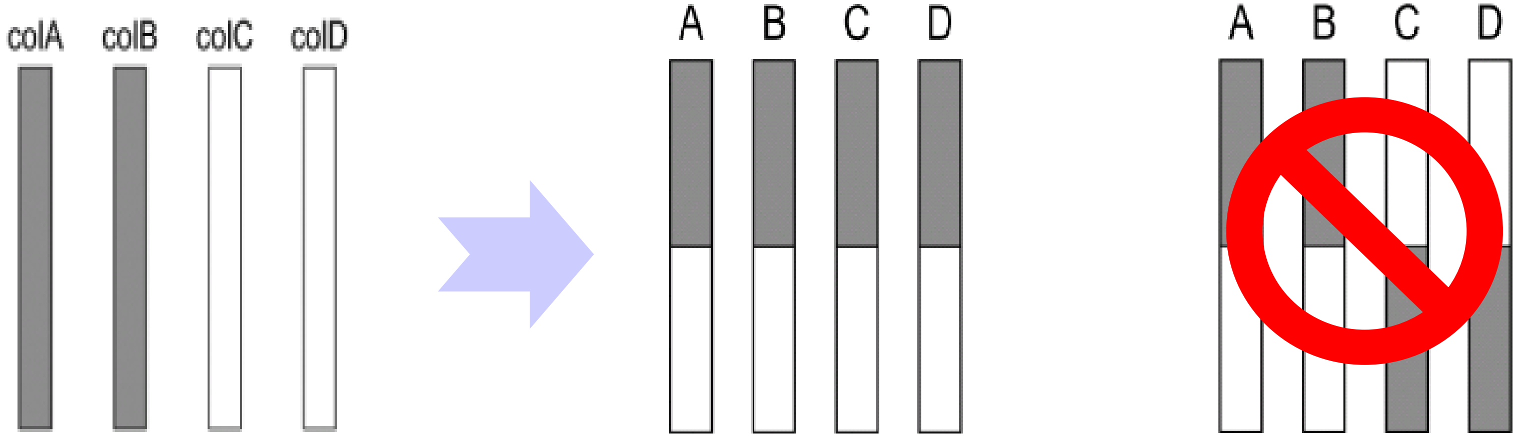
- 입력이 되는 **SYBASE IQ**쪽에 **OCDK**가 설치되어 있어야 하며 그 쪽 **interfaces file**에 **SELECT** 가 수행되는 다른 **DB**에 대한 목록이 등록되어 있어야 합니다.

- 예)

```
INSERT INTO customer
            ( customer_id, cust_type )
LOCATION 'prod.asedb'
{ SELECT customer_id, cust_type
  FROM customer } ;
```



insert-partial width insert



- 한 테이블에 데이터를 입력해서 사용하다가 업무의 요구에 따라 특정한 컬럼을 첨가 했을 때 다른 **RDBMS** 처럼 데이터의 재로드 없이 이 첨가된 컬럼에 데이터를 입력할 수 있는 방법을 **SYBASE IQ**는 제공합니다. **SYBASE IQ**는 컬럼 단위로 데이터를 저장하기 때문에 컬럼 단위의 처리에 매우 강한 특성이 있습니다.

- 예)

```
INSERT INTO lineitem ( colC, colD )
START ROW ID 1
LOCATION 'ase_srv1.part_db'
{ SELECT  colC, colD  FROM lineitem } ;
```

update

- **12.4** 이하의 버전에서는 **SYBASE IQ**에서 **UPDATE**를 사용하는 제약 사항이 조금 있었지만 지금은 모든 제약 사항이 제거 되었을 뿐만 아니라 조인을 이용하여 수정할 수 있는 확장 기능도 제공하고 있습니다.

- **SYBASE IQ**는 컬럼별로 데이터를 저장하기 때문에 특정 몇 개의 컬럼만을 수정하는 것은 다른 **RDBMS**에 비해 월등히 빠른 성능을 보장합니다.

- **syntax**

```
UPDATE [owner.]table-name
     SET  [column-name = expression, …
[ FROM table-expression, ]
[ WHERE search-condition [, join-condition ]] ……
```

- **예)**

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129;

UPDATE employee
SET emp.salary = emp.salary + dept.bonus
FROM employee emp, department dept
WHERE emp.deptnum = dept.deptnum;
```

delete

- 특정 테이블에 조건을 주어 데이터를 삭제할 수 있을 뿐만 아니라 다른 테이블과 조인을 하여 삭제할 수 있는 확장 기능도 제공합니다.

- 조건 없이 전 테이블을 삭제시는 **TRUNCATE TABLE** 명령어를 사용하는 편이 로그를 적게 사용합니다.

- **syntax**

```
DELETE [ FROM ] [owner.]table-name
  [ FROM  table-list ]
  [ WHERE search-condition ]
```

- 예)

```
DELETE employee
WHERE emp_id = 105;

DELETE   contact
FROM     contact, customer
WHERE   contact.last_name = customer.lname
AND       contact.first_name = customer.fname;

TRUNCATE TABLE employee;
```

Using procedure

procedure

- 절차적인 **SQL** 언어에 의해 작성되어 데이터베이스내에 저장되며 모든 응용 프로그램에서 사용 가능합니다.
- 반복문과 조건문등 제어문을 포함할 수 있으며 **CALL**문에 의해 호출되고 파라미터를 주고 받을 수 있습니다
- 프로시져를 호출하는 쪽에 **SELECT**의 결과등 여러 가지 정보를 줄 수 있으며 또 다른 프로시져를 실행할 수도 있습니다.
- 종류
  - ● **User-Defined Stored Procedure**
  - ● **System-Defined Stored Procedure**
- 장점
  - ● 반복적인 일을 단순화/표준화 시킬 수 있습니다
  - ● 네트워크 트래픽을 감소시킬 수 있습니다
  - ● 유지보수가 간편합니다
  - ● ……

creating stored procedure

- **resource** 권한을 가진 사용자가 **CREATE PROCEDURE** 라는 명령어를 사용해서 생성할 수 있습니다

- **ALTER PROCEDURE** 라는 명령어를 사용해서 이미 만들어진 프로시져를 변경할 수 있습니다

- 프로시져의 몸체는 **BEGIN**과 **END**로 이루어진 **compound statement** 이며 각 문장은 세미콜론**(;)**으로 끝내야 합니다**.**

- 생성하는 프로시져의 파라미터가 존재하지 않는다고 하더라고 **( )**는 문법상 반드시 필요합니다**.**

- **syntax**

  CREATE PROCEDURE *procedure_name* **(** *[parameter][,……]* **)**
  [RESULT (*result-column***,…)]**
  BEGIN

  // Business Logic

  END;

Compound statement

declaring parameter

- 호출하는 프로그램과 호출받는 프로시져 사이에 정보를 주고 받는 방법이 정의되어 있으며 호출하는 쪽에서 호출받는 쪽으로 정보를 줄 수 있는 방법은 파라미터를 통해서 입니다.

- 또한 호출하는 쪽에서도 받을 수 있는 방법이 있는데 그 중에 하나가 역시 파라미터를 통해서 이며 이런 파라미터는 **CREATE PROCEDURE**문의 괄호안에 선언하며 콤마로 분리됩니다

- 컬럼처럼 적절한 데이터타입을 가지고 있고 선언과 동시에 **DEFAULT** 선언도 가능하며 이런 파라미터 이름앞에는 다음과 같은 **3**종류의 **Keyword**가 사용됩니다.
  - **IN :** 호출하는 쪽에서 프로시져에게 전달하는 파라미터
  - **OUT :** 프로시져에서 호출하는 쪽으로 전달하는 파라미터
  - **INOUT : IN** 역할과 **OUT** 역할을 동시에 수행하는 파라미터

- **syntax**

  CREATE PROCEDURE procedure-name
  (IN | OUT | INOUT parameter-name data-type [DEFAULT expression],……)

- 예)

  CREATE PROCEDURE ProductType (IN product_id INT, OUT type CHAR(10))
  BEGIN
  …………..

checking parameter

- NULL과 상수값등으로 파라미터의 디폴트 값을 지정하여 프로시저를 호출하는 쪽에 파라미터를 넘기지 않았을 때 처리를 가능하게 할 수 있습니다. 이런 식의 파라미터 점검은 논리 오류를 미연에 방지할 수 있습니다.

- 예)

```
CREATE PROCEDURE CustomerProducts_proc
                                    ( IN v_customer_id INT DEFAULT NULL )
BEGIN
   IF v_customer_id IS NULL THEN          // 만약 파라미터를 받지 못했다면
      RETURN –99000;
   ELSE
      SELECT  a.id, SUM(b.quantity)
      FROM    product a, sales_order_items b, sales_order c
      WHERE   c.cust_id = v_customer_id
      AND     c.order_id = b.id
      AND     b.prod_id = a.id
      GROUP BY a.id;
   END IF;
END;
```

calling stored procedure

- 프로시져는 응용 프로그램에서 혹은 다른 프로시져에서 **CALL** 문에 의해 불려질 수 있습니다.

- 호출하는 사용자는 그 프로시져에 대해 실행 권한을 가져야 합니다

- 프로시져의 실행이 끝났을 때 **OUT, INOUT**으로 선언된 파라미터 값이 호출하는 프로그램이나 프로시져로 되돌려 집니다. 또한 프로시져의 실행 결과 상태 값을 받아 참조할 수 있습니다.

- **Syntax**
  [*variable***=] CALL** *procedure_name* **(** *[parameter][,……]* **);**

- 예1)
  CALL customer_list_proc();

- 예2)
  CREATE VARIABLE v_returnval INT;
  v_returnval = CALL integer_proc (arg1 = val1, ……);

passing parameter

- creator

```
CREATE PROCEDURE Sample_proc ( IN v_var1 INT DEFAULT NULL,
                               IN v_var2 INT DEFAULT NULL,
                               IN v_var3 INT DEFAULT NULL)
BEGIN
    ………………………………
END;
```

- caller : 위치에 의한 방법

```
CREATE VARIABLE v_loc1 INT;
CREATE VARIABLE v_loc2 INT;
CREATE VARIABLE v_loc3 INT;
SET v_loc1 = 100; SET v_loc2 = 200; SET v_loc3 = 300;
CALL Sample_proc ( v_loc1, v_loc2, v_loc3 );
```

- caller : 이름에 의한 방법

```
CREATE VARIABLE v_name1 int;
CREATE VARIABLE v_name2 int;
CREATE VARIABLE v_name3 int;
SET v_name1 = 100; SET v_name2 = 200; SET v_name3 = 300;
CALL Sample_proc ( v_var3=v_name3, v_var2=v_name2, v_var1=v_name1 );
```

returning results : case1

- 호출하는 프로그램에서 프로시저를 호출할 때 **OUT** 혹은 **INOUT**으로 선언되어진 파라미터를 통해 개별적인 값을 받아 볼 수 있습니다.

- 예)

```
 - creator
   CREATE PROCEDURE AverageSalary_proc ( OUT v_avgsal NUMERIC (20,3) )
   BEGIN
        SELECT AVG(salary) INTO v_avgsal
        FROM employee;
   END;

- caller
  CREATE VARIABLE v_average NUMERIC(20,3);
  CALL AverageSalary_proc(v_average);
  SELECT v_average;
```

returning results : case2

- 호출하는 프로그램에서 프로시져에서 **SELECT** 한 결과를 받아 볼 수 있습니다

- 예)

  - creator

```
CREATE PROCEDURE SalaryList_proc ( IN v_dept_id INT )
BEGIN
     SELECT  emp_id, salary
     FROM  employee
     WHERE dept_id = v_dept_id;
END
```

  - caller

```
CALL SalaryList_proc(100);
```

- 참고

  Interactive SQL사용시 해당되는 프로시져의 종료 후에 **RESUME**과 같은 커서를 닫을 수 있는 명령어를 실행해야만 해당되는 프로시져가 **DROP** 되거나 **ALTER** 될 수 있습니다. 그 프로시져에서 **SELECT** 대상이 되는 테이블도 마찬가지 경우입니다

returning results : case3

- 호출하는 프로그램에서 프로시져의 RETURN문과 함께 사용된 숫자 상태 값을 결과로 받아볼 수 있습니다.

- syntax

  ```
  RETURN [(expr)]
  ```

- 예)

  - creator

  ```
  CREATE PROCEDURE SalaryList_proc ( IN v_dept_id INT DEFAULT NULL)
  BEGIN
      IF v_dept_id IS NULL THEN
          RETURN –99000;
      ELSE
          SELECT emp_id, salary
          FROM employee
          WHERE dept_id = v_dept_id;
      END IF;
  END
  ```

  - caller

  ```
  CREATE VARIABLE ret_value INT;
  ret_value = CALL SalaryList_proc(100);
  SELECT ret_value;
  ```

result option

- Embedded SQL나 ODBC와 같은 프로그램에서 RESULT 옵션을 사용하여 SELECT 결과의 컬럼 이름과 데이터 타입을 알 수 있습니다.

- 예)

  - creator

```
CREATE PROCEDURE SalaryList_proc ( IN v_dept_id INT DEFAULT NULL)
RESULT ( “Employee ID” INT, Salary NUMERIC(20,3))
BEGIN
     IF v_dept_id IS NULL THEN
           return –99000;
     ELSE
           SELECT emp_id, salary
           FROM employee
           WHERE dept_id = v_dept_id;
      END IF;
 END
```

dropping stored procedure

- 프로시져의 생성자나 혹은 **dba** 권한을 가진 사용자가 **DROP PROCEDURE** 라는 명령어를 사용해서 제거할 수 있습니다.

- 다른 사용자가 현재 사용중인 프로시져는 삭제될 수 없지만 실행이 종료된 후에 커서가 열려 있는 상태로 된 프로시져도 역시 삭제될 수 없습니다.

- **syntax**

```
DROP PROCEDURE procedure_name;
```

- 예**1)**

```
DROP PROCEDURE customer_list_proc;
```

- 예**2)**

```
IF EXISTS ( SELECT 1 FROM sysprocedure WHERE proc_name = 'proc_name' )
THEN
        DROP PROCEDURE procedure_name;
END IF;

CREATE PROCEDURE ……
```

creating function

- **resource** 권한을 가진 사용자가 **CREATE FUNCTION** 이라는 명령어를 사용해서 생성할 수 있습니다

- **ALTER FUNCTION** 이라는 명령어를 사용해서 이미 만들어진 함수를 변경할 수 있습니다

- 함수의 몸체는 **BEGIN**과 **END**로 이루어진 **compound statement** 이며 모든 파라미터는 **IN** 이기 때문에 **IN, OUT, INOUT** 과 같은 **keyword**가 필요 없습니다

- **return**되는 데이터타입을 표현하기 위한 **RETURNS**절이 꼭 필요하며 하나의 값을 되돌려 주기 위한 **RETURN**문도 반드시 필요합니다

- **Syntax**

```
CREATE FUNCTION function_name ( [parameter][,…….] )
RETURNS data_type
BEGIN
        // Business Logic
        RETURN (return_value);
END;
```

Compound statement

calling function

- 시스템이 제공하는 **non-aggregation** 함수가 사용되는 곳이면 어디나 사용 가능 합니다

- 호출하는 사용자는 그 함수에 대해 실행권한을 가져야 합니다

- **syntax**

```
SELECT function_name ( [parameter][,……] ), ……
FROM table_name;
```

- 예)

```
SELECT fullname ( emp_fname, emp_lname)
FROM employee;

SELECT fullname ( 'Jane', 'Smith' );
```

returning function result

- RETURN문을 사용하여 하나의 값을 되돌릴 수 있습니다

- 예)

  - creator

```
CREATE FUNCTION fullname ( v_fname CHAR(30), v_lname CHAR(30) )
RETURNS CHAR(61)
BEGIN
     DECLARE name CHAR(61);
     SET name = v_fname || ‘ ‘ || v_lname;
     RETURN ( name );
END;
```

  - caller

```
SELECT fullname (emp_fname, emp_lname)
FROM  employee;
```

- 참고

  INSERT … SELECT문에서 사용자 함수를 사용하면 1건씩 입력이 수행되어 성능에 지대한 영향을 미치며 EXTRACT 할 때는 사용이 불가능 합니다.

dropping function

- 함수의 생성자나 혹은 **dba** 권한을 가진 사용자가 **DROP FUNCTION** 이라는 명령어를 사용해서 제거할 수 있습니다

- **syntax**

```
DROP FUNCTION function_name;
```

- **예1)**

```
DROP FUNCTION customer_list_proc;
```

- **예2)**

```
IF EXISTS ( SELECT 1 FROM sysprocedure WHERE proc_name = ‘func_name’ )
THEN
        DROP FUNCTION function_name;
END IF;

CREATE FUNCTION ……
```

permission

- **dba**나 혹은 **resource** 권한을 가진 사용자만이 프로시져나 함수를 생성할 수 있습니다
  - **GRANT** *resource* **TO** *user_name*;

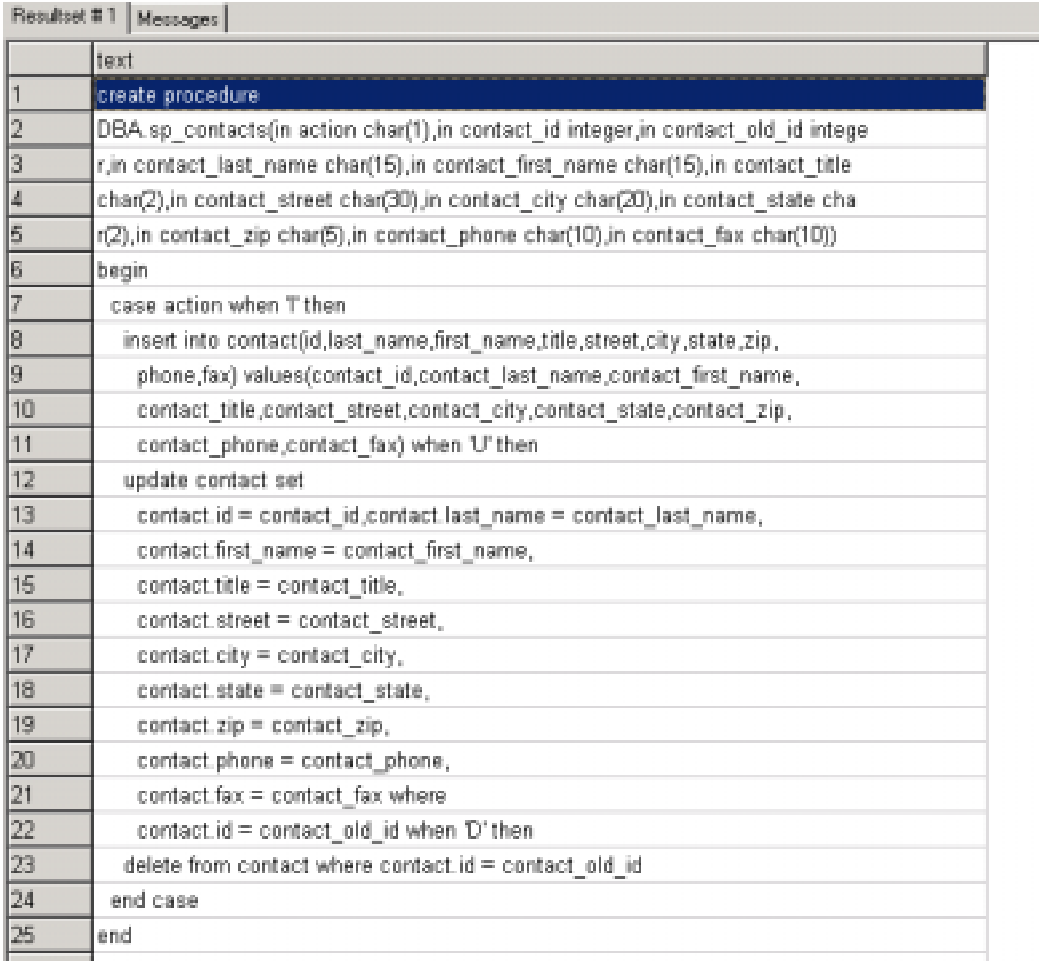
- 호출하는 사용자는 그 프로시져나 함수에 대해 실행권한을 가져야 합니다
  - **GRANT execute ON procedure_name TO user_name;**

- 프로시져나 함수의 생성자 혹은 **dba** 권한을 가진 사용자가 프로시져나 함수를 제거할 수 있습니다

sp_helptext

- syntax

  sp_helptext ***procedure-name***;



```
create procedure
DBA.sp_contacts(in action char(1),in contact_id integer,in contact_old_id intege
r,in contact_last_name char(15),in contact_first_name char(15),in contact_title
char(2),in contact_street char(30),in contact_city char(20),in contact_state cha
r(2),in contact_zip char(5),in contact_phone char(10),in contact_fax char(10))
begin
  case action when 'I' then
    insert into contact(id,last_name,first_name,title,street,city,state,zip,
      phone,fax) values(contact_id,contact_last_name,contact_first_name,
      contact_title,contact_street,contact_city,contact_state,contact_zip,
      contact_phone,contact_fax) when 'U' then
    update contact set
      contact.id = contact_id,contact.last_name = contact_last_name,
      contact.first_name = contact_first_name,
      contact.title = contact_title,
      contact.street = contact_street,
      contact.city = contact_city,
      contact.state = contact_state,
      contact.zip = contact_zip,
      contact.phone = contact_phone,
      contact.fax = contact_fax where
      contact.id = contact_old_id when 'D' then
    delete from contact where contact.id = contact_old_id
  end case
end
```

tip

- **stored procedure** 내에서 사용되어지는 임시 테이블의 이름은 해당 **stored procedure**의 이름과 연관성 있게 명명해야 디버깅이 편리합니다.

- **stored procedure**에서 사용되는 파라미터와 변수는 해당되는 컬럼의 데이터 타입과 동일하게 선언하여 주십시오.

- **stored procedure**에서 사용하는 변수는 **v_**와 같은 접두어를 사용하여 누가 언제 보아도 알 수 있도록 해주어야 읽기에 편리합니다.

- **stored procedure**는 만들어지는 시점에 기본적인 오류 점검만 하고 대부분의 오류는 실행시에 점검되어지기 때문에 반드시 생성 후에 실행해 보십시오.

- **T-SQL**과 혼용하지 말고 **SYBASE IQ**의 기본 **SQL**인 **WatcomSQL**을 사용하여 프로그램하십시오.

- 데이터베이스 복구불능과 같은 만약의 사태에 대비하여 항상 최신 버전의 **script**를 따로 보관하며 때에 따라서는 이런 **script**의 **version** 관리도 직접 하십시오.

System function

문자함수

- **ASCII(***string_expr***) : string_expr 첫 번째문자의 아스키 값을 산출합니다**
  SELECT ASCII(‘A’) -> 65

- **CHAR(***integer_expr***) : integer_expr을 character로 변환합니다**
  SELECT CHAR(65) -> ‘A’

- **INSERTSTR(***integer_expr***,** *string_expr1***,** *string_expr2***) : string2_expr2를string_expr1의 integer_expr번 째에 끼워 넣습니다**
  SELECT INSERTSTR(3,’ABCFG’,’DE’) -> ‘ABCDEFG’

- **LCASE(***string_expr***) : string_expr를 소문자로 변환합니다(lower와 동일)**

- **LEFT(***string_expr, integer_expr***) : string_expr의 왼쪽부터 integer_expr 자리만큼 선택합니다**
  SELECT LEFT(‘ABCDEF’,3) -> ‘ABC’

- **LENGTH(***string_expr***) : string_expr의 character의 길이를 나타냅니다**

- **LOCATE(***string_expr1, string_expr2***) : string_expr2가 string_expr1의 몇 번째 위치하는가를 표시합니다**

문자함수

- **LTRIM(*string_expr*) : string_expr**의 왼쪽 **blank**를 제거합니다
- **RIGHT(*string_expr, integer_expr*) : string_expr**의 오른쪽부터 **integer_expr** 자리 만큼 선택합니다
  SELECT RIGHT('ABCDEF',3) -> 'DEF'
- **RTRIM(*string_expr*) : string_expr**의 오른쪽 **blank**를 제거합니다
- **SIMILAR(*string_expr1, string_expr2*) : string_expr1**과 **string_expr2**의 유사성이 몇 **%** 인가를 표현합니다
- **STRING(*string1, [string2,……..string99]*) :** 나열된 **string**을 연결합니다**. (||** 가능**)**
- **STUFF(*string_expr1, start, length, string_expr2*) : string_expr1**의 **start** 부터 **length** 만큼 버리고 그자리에 **string_expr2**를 채웁니다
- **SUBSTR(*string_expr, start, length*) : string_expr**에서 **start** 번째 부터 **length** 만큼 잘라서 가져옵니다

문자함수

- **TRIM(***string_expr***) : string_expr**의 처음과 끝의 **blank**를 제외시킵니다
- **UCASE(***string_expr***) : string_expr**를 대문자로 변환합니다**(UPPER**와 동일**)**
- 그 외 **……**

숫자함수

- **ABS(***numeric_expr***) : 절대값을 산출합니다**
- **CEILING(***numeric_expr***) : 소수점이 있을 때 가장 큰 integer 값을 산출합니다**
- **FLOOR(***numeric_expr***) : 소수점이 있을 때 가장 작은 integer 값을 산출합니다**
- **MOD(***dividend, divisor***) : dividend를 divisor로 나눈 나머지를 산출합니다**
- **POWER(***numeric_expr1, numeric_expr2***) : numeric_expr1에 numeric_expr2 제곱한 결과를 산출합니다**
- **RAND(***integer_expr***) : 소수점 이하 15자리 난수를 발생시킵니다**
- **ROUND(***numeric_expr, integer_expr***) : numeric_expr을 integer_expr 자리로 반올림 합니다**
- **TRUNCATE(***numeric_expr, integer_expr***) : numeric_expr에서 소수점 integer_expr 자리 미만을 자름**
- **그 외 acos, asin, atan, log……**

날짜함수

- **DATE(*expr*) : expr의 데이터를 년월일만 표시합니다**
- **DATEFORMAT(*date_expr, string_expr*) : date_expr를 string_expr 형으로 변환**
  SELECT DATEFORMAT('1999-01-01','mm dd, yyyy') -> 01 01, 1999
- **DAY(*date_expr*) : date_expr의 날짜를 수로 표현 (1~31) 합니다**
- **DAYNAME(*date_expr*) : date_expr의 요일을 영문으로 나타냅니다**
- **DAYS(*date_expr1, date_expr2*) : date_expr1에서 date_expr2 까지의 날짜 수**
- **DOW(*date_expr*) : date_expr의 요일을 수로 표현합니다. (1=Sunday, 2=Monday, ……7=Saturday)**
- **MONTH(*date_expr*) : date_expr의 월을 수로 표현(1~12)합니다**
- **MONTHNAME(*date_expr*) : date_expr의 월을 영문으로 나타냅니다**
- **MONTHS(*date_expr1, date_expr2*) : date_expr1에서 date_expr2 까지의 총 개월 수를 나타냅니다**

날짜함수

- **QUARTER(***date_expr***) : date_expr**의 분기를 수로 표현**(1~4)**합니다
- **WEEKS(***date_expr1, date_expr2***) : date_expr1**에서 **date_expr2**까지의 총 주
- **YEAR(***date_expr***) : date_expr**의 년도를 수로 표현합니다
- **YEARS(***date_expr1, date_expr2***) : date_expr1**에서 **date_expr2** 까지의 총 년
- **YMD(***year_num, month_num, day_num***) : year_num**의 년도에 **month_num**의 개월을 더하고 **day_num**을 더한 일자를 나타냅니다
- **NOW(*******) :** 현재 일자와 시간**,** 분**,** 초**,Millisecond**를 나타냅니다
- **TODAY(*******) :** 시간**,** 분**,** 초**,Millisecond**를 제외한 현재 일자만 표시합니다

날짜함수

■ 날짜연산

- ● SELECT YEARS(DATE('2001-05-24'), 1) : 2002-05-24 00:00:00.000
- ● SELECT MONTHS(DATE('2001-05-24'), 1) : 2001-06-24 00:00:00.000
- ● SELECT DAYS(DATE('2001-05-24'), 1) : 2001-05-25 00:00:00.000

■ 날짜비교

- ● SELECT YEARS(DATE('2001-05-24'), DATE('2004-05-26')) : 3
- ● SELECT MONTHS(DATE('2001-05-24'), DATE('2001-06-26')) : 1
- ● SELECT DAYS(DATE('2001-05-24'), DATE('2001-05-26')) : 2

■ 날짜추출

- ● SELECT YEAR('2001-05-24') : 2001
- ● SELECT MONTH('2001-05-24') : 05
- ● SELECT MONTHNAME('2001-05-24') : May
- ● SELECT DAY('2001-05-24') : 24
- ● SELECT DAYNAME('2001-05-24') : Thursday

■ 기타

- ● SELECT NOW() 혹은 SELECT TODAY() : 현재일을 구함
- ● SELECT DATEFORMAT(DATE('2001-05-24'), 'yyyy/mm/dd') : 2001/05/24
- ● SELECT DATE('2001-05-24') : string '2001-05-24'를 날짜 2001-05-24로 변경

시간함수

- **HOUR(***datetime_expr***) : datetime_expr**의 시간을 나타냅니다**(0~23)**
- **HOURS(***datetime_expr1, datetime_expr2***) : datetime_expr1**에서 **datetime_expr2**까지의 총 시간을 나타냅니다
- **MINUTE(***datetime_expr***) : datetime_expr**의 분을 나타냅니다**(0~59)**
- **MINUTES(***datetime_expr1, datetime_expr2***) : datetime_expr1**에서 **datetime_expr2**까지의 총 분을 나타냅니다
- **SECOND(***datetime_expr***) : datetime_expr**의 초를 나타냅니다**(0~59)**
- **SECONDS(***datetime_expr1, datetime_expr2***) : datetime_expr1**에서**datetime_expr2**까지의 총 초을 나타냅니다

기타함수

- **NUMBER(*) : result set**의 각 레코드에 대해 일련 번호를 부여하는 함수입니다

- **ROWID(***table_name***) :** 각 테이블에 데이터가 입력될 때 입력되는 순서에 의해 각각의 **ID** 가 일련 번호로 저장됩니다**.** 그 번호를 참조하는 함수입니다**.** 실제로 데이터가 입력된 테이블에서만 사용 가능하여 **derived table(in-line view)** 에서는 사용할 수 없습니다**.**

- **COALESCE(***expr1, expr2***) :** 여기서 **expr1**은 **null** 값을 포함할 수 있는 값 이거나 표현식 입니다**. expr2**는 **null** 값을 전환시 대치할 값입니다**.**

- **IFNULL(***expr1, expr2 [,expr3]***) : expr1**의 값이 **NULL** 이면 **expr2**를**,** 아니면 **expr3**을 반환합니다**.**

- **NULLIF(***expr1, expr2***) : expr1**과 **expr2**가 같으면 **NULL**을**,** 아니면 **expr1**을 반환합니다**.**

시스템함수

- CURRENT DATE : 현재일자를 나타내는 상수함수 입니다.
- CURRENT TIME : 현재시간을 나타내는 상수함수 입니다.
- CURRENT TIMESTAMP: 현재일자와 현재시간을 조합하여 나타내는 상수함수 입니다.
- CURRENT USER : 현재 connection 되어 있는 사용자의 ID, 즉 자기의 ID 입니다.
- SQLCODE : 현재의 SQLCODE값 입니다.
- SQLSTATE : 현재의 SQLSTATE값 입니다. 예를 들어 26501값은 'SQL statement error' 를 표시합니다
- CONNECTION_PROPERTY(*expr*) : expr의 connection property 값을 반환합니다. expr 값은 sa_conn_properties의 결과를 참조하세요.
- DB_PROPERTY(*expr*) : expr의 db property 값을 반환합니다. expr 값은 sa_db_properties의 결과를 참조하세요.
- PROPERTY(*expr*) : expr의 server property 값을 반환합니다. expr 값은 sa_eng_properties의 결과를 참조하세요.

타입변경함수

- **CAST (***expr* **AS** *data_type***) : expr**의 데이터 타입을 **data_type**으로 변환합니다
  SELECT CAST(‘ABCDE’ AS char(2)) : ‘AB’

- **DATE(***expr***) : expr**을 **DATE** 타입으로 변환합니다**.**
  SELECT DATE(20030301) : 2003-03-01

- **DATETIME(***expr***) : expr**을 **DATETIME** 타입으로 변환합니다**.**
  SELECT DATETIME(‘20030301’) : 2003-03-01 00:00:00.000

Creating table and index

create table

- syntax

```
CREATE [ GLOBAL TEMPORARY ] TABLE [owner].table-name
(
     column-definition [column-constraint],
     column-definition [column-constraint],
     . . . . . . . . .
     [table-constraint],
     [table-constraint],
     . . . . . . . . .
)
[IN dbspace-name]                               // system이면 catalog 영역, 사용자제
[ON COMMIT DELETE | PRESERVE ROWS]   // global temporary table일 경우만
```

column-definition : column-name data-type [[not] null]
column-constraint : unique, primary key, references, iq unique
table-constraint : unique, primary key, foreign key

create table-column definition

- column-name, data-type, property로 구성되며 property에는 NULL과 NOT NULL를 선택할 수 있고 생략하면 클라이언트 종류에 따라 NULL과 NOT NULL이 바뀔 수 있으므로 테이블 생성 스크립트에 반드시 표현해 주십시오.

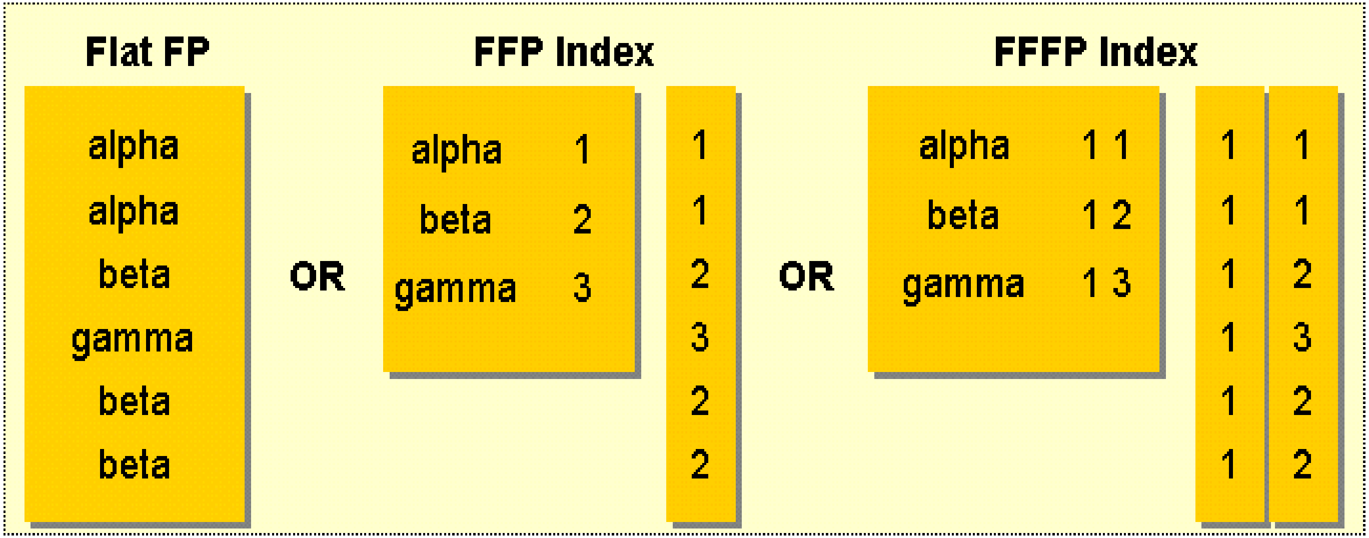
- 예)

```
CREATE TABLE employee
(
  emp_id        INT                  NOT NULL,
  lname         CHAR(30)             NOT NULL,
  fname         CHAR(30)             NOT NULL,
  salary        UNSIGNED INT         NOT NULL,
  dept_id       INT                  NOT NULL
)
```

- 참고
  NULL, NOT NULL 생략시 isql 에서는 NOT NULL, dbisqlc에서는 NULL을 기본값으로 취급합니다.

create table-constraint

- **IQ UNIQUE**
  디폴트 인덱스인 **FP** 인덱스의 타입을 컬럼의 카디날리티에 따라 선택할 수 있도록 합니다. 생략하면 **Flat FP** 인덱스가 만들어지므로 반드시 사용하기를 권장합니다. 카디날리티가 **255** 이하이면 **FFP** 인덱스가, **256~65536** 이면 **FFFP** 인덱스가 자동으로 생성됩니다. 이렇게 **IQ UNIQUE**를 적용하면 더 좋은 압축률과 더 좋은 실행 계획을 산출할 수 있도록 도와줍니다.



Flat FP

alpha
alpha
beta
gamma
beta
beta

OR

FFP Index

alpha 1
beta 2
gamma 3

1
1
2
3
2
2

OR

FFFP Index

alpha 1 1
beta 1 2
gamma 1 3

1 1
1 1
1 2
1 3
1 2
1 2

- 참고
  SYBASE IQ의 **cost-based** 옵티마이저가 실행 계획을 작성할 때 **catalog**로 부터 여러 가지 정보를 취합하게 되는데 여기서 중요한 인자중의 하나가 바로 각 컬럼의 카디날리티입니다. 이런 카디날리티 정보는 **HG, LF** 인덱스와 **IQ UNIQUE**에 의해서 생성됩니다.

create table-constraint

- **UNIQUE**
  unique HG 인덱스가 내부적으로 자동으로 생성되며 옵티마이져에게 보다 많은 정보를 주어 좋은 실행계획을 생성하도록 도와줍니다**.** 그렇지만 관리상 가능하면 **CREATE INDEX**로 사용하시는 편이 좋을 듯 합니다**.**

- **PRIMARY KEY**
  unique HG 인덱스가 내부적으로 만들어지며 **2**개 이상의 컬럼을 조합하여 인덱스를 만들 수 있습니다**.** 데이터의 유일성 체크와 옵티마이져에게 보다 많은 정보를 주어 좋은 실행계획을 생성하도록 도와줍니다. 조합 인덱스의 경우 각 컬럼에 대한 인덱스는 별도로 다시 만들어 줘야 합니다**.**
  디멘션 테이블에서는 반드시 사용을 권장합니다**.** 조합 인덱스를 사용하여 조인 질의 실행시 인덱스로서 기능을 발휘합니다**.** 단**, PK, FK**의 선언이 선행되어야 하며 조합 인덱스로 사용된 모든 컬럼이 **Equal** 질의로 사용되어야만 합니다**.**

- **FOREIGN KEY**
  non-unique HG 인덱스가 내부적으로 만들어지며 **2**개 이상의 컬럼을 조합하여 인덱스를 만들 수 있습니다. 이렇게 **FK**가 생성되어지면 참조 무결성 규칙이 적용됩니다**.** 단**,** **Disable_RI_Check** 옵션이 **Off** 일 경우에만 해당됩니다**.**

- **CHECK, ROLE, DEFAULT**와 같은 **CONSTRAINT**는 **SYBASE IQ**에서는 제공되지 않습니다**.**

referential integrity

- 참조 무결성 규칙은 **12.5** 버전부터 도입되었으며 다음과 같이 동작합니다**. FK**를 포함한 테이블에 **INSERT/UPDATE**가 발생하면 **FK** 데이터를 **PK**와 비교하여 위반되면 **ROLLBACK**하고 아니며 **INSERT/UPDATE**를 실행합니다**.**

- **PK**를 포함한 테이블에 **DELETE**가 발생할 때 이 테이블을 참조하는 **FK** 값이 존재하면 **ROLLBACK**하며 이 테이블에 존재하지 않는 값으로의 **UPDATE**도 역시 **ROLLBACK** 합니다**.** 단**, CASCADE DELETE**와 **FK** 값을 **NULL**로 변경하는 기능은 향후 제공 예정입니다**.**

- 데이터가 존재하는 테이블에 **ALTER TABLE ADD FOREIGN KEY** 명령어를 실행할 때 **RI** 위반이 발생하면 명령어가 실행되지 않을 것 입니다**.**

- **LOAD TABLE** 할 때 **RI** 위반이 발생하면 옵션에 따라 **ROLLBACK** 혹은 재처리 파일로 저장됩니다**.** 재처리 파일로의 데이터 저장 기능은 **12.5** 이상의 기능이며 추후 확인 과정을 거쳐 재입력될 수 있습니다**.**

- **BASE TABLE**를 참조하는 **FK**를 임시 테이블에 생성할 수 없으며 임시 테이블을 참조하는 **FK** 역시 **BASE TABLE**에 생성할 수 없습니다**.**

- 여러 가지 조건에 따라 달라질 수 있으나 **RI** 검사를 위해 일반적으로 **5%** 정도의 오버헤드가 존재하고 **M:M** 조인 성능 향상과 같은 추가적인 성능 향상 이점도 있으므로 상황에 따라 적절히 선택하여 사용하시기 바랍니다**.**

create table-data type

Data Type	Range	Max Prec.	Storage (byte)
CHAR (n) CHARACTER (n)	1 <= n <= 255		n
VARCHAR (n) CHARACTER VARYING (n)	1 <= n <= 255		n
VARCHAR (n) CHARACTER VARYING (n)	256 <= n <= 32K		256 + (n - 255)
INTEGER UNSIGNED INT	-2,147,483,648 ~ 2,147,483,647 0 ~ 42,942,967,294	10 11	4
TINYINT	0 ~ 255	3	1
SMALLINT	-32,768 ~ 32,767	5	2
BIGINT UNSIGNED BIGINT	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 0 ~ 18,446,744,073,709,551,615	19 20	8

create table-data type

Data Type	Range	Max Prec.	Storage (byte)
FLOAT (n)	Platform-dependent	16	4 or 8
REAL	Platform-dependent	7	4
DOUBLE	2.22 (^308) ~ 1.79 (^308)	15	8
DECIMAL (p,s) NUMERIC (p,s)	-10^38 ~ 10^38 – 1	126	2 to 69
BINARY (n)	1 <= n <= 255		256
VARBINARY (n)	1 <= n <= (32k – 1)		32K – 1
LONG BINARY			64K –1
BIT	0, 1, NULL		1

create table-data type

Data Type	Range	Max Prec.	Storage (byte)
DATE	0001/01/01 ~ 9999/12/31		4
DATETIME SMALLDATETIME TIMESTAMP	0001/01/01 00:00:00.000000 ~ 9999/12/31 23:59:59.999999		8
TIME	00:00:00.000000 ~ 23:59.59.999999		8

create table-data type

- 기본적으로 각 컬럼의 데이터 타입 선택할 때 INT 계열을 최우선 순위로 고려할 것을 적극 권장합니다. (INT 계열 : TINYINT, SMALLINT, INT, UNSIGNED INT, BININT, UNSIGNED BIGINT)

- CHAR/VARCHAR : VARCHAR, CHAR에 데이터가 입력되면 그 뒤에 원래 생성시에 기술한 크기 만큼의 공백 데이터가 채워져 기존 시스템의 VARCHAR를 사용할 때의 저장공간 절약이라는 이점은 사라지고 오히려 가변길이중 실 데이터 길이를 저장하는 추가 1Byte라는 오버헤드가 발생합니다. 그래서 255 Byte이하는 CHAR 사용하기를 권장합니다.

- 날짜 : 일반적으로 CHAR(8)로 선언하여 사용하는데 DATE 형태로 선언하여 사용하는 것이 SYBASE IQ의 특성상 인덱스의 검색속도를 향상 시키므로 특별한 이유가 없으면 DATE를 권장합니다. 특히 12.5 이상에서 소개된 DATE INDEX의 성능은 그 대상 데이터 타입이 DATE 일 경우에만 해당됩니다. 또한 데이터로서 시간에 대한 표현이 필요없고 날짜에 대한 표현만 필요하다면 8Byte의 DATETIME 보다는 4Byte의 DATE을 사용하는 편이 공간 절약측면에서 유리합니다.

- 소수점 이하가 없는 숫자 타입은 NUMERIC/DECIMAL로 선언하는 것보다 INT 형태로 선언하는 것이 인덱스 검색 속도 및 조인 속도를 향상 시킵니다. 특히 조인 컬럼은 TINYINT, SMALLINT, UNSIGNED INT, UNSIGNED BIGINT가 가장 좋습니다.

create table-data type

- **NUMERIC(p,s)** 의 **precision :** 아래 표를 참조하여 확장성 고려해 충분한 **precision** 정의하시기 바랍니다**. DECIMAL**은 **NUMERIC**과 같습니다**.**

Precision	Length(byte)
1 – 4	2
5 – 9	4
10 – 18	8
19 이상	4 + 2 * (int(((prec – scale) + 3) / 4) + int((scale + 3) / 4) + 1)

- 참고
  대용량을 취급하는 팩트와 같은 테이블에서는 데이터 타입의 크기 선택이 디스크 절감에 지대한 영향을 미칩니다**.**
  예를 들어 **1,000**만 건의 데이터가 들어갈 팩트 테이블의 특정 컬럼을 **INT**에서 **TINYINT** 나 **SMALLINT**로 바꿀 수 있다면 어떻게 될까요**?**

create table-data type

Column 범주	DataType	Bytes	비고
SystemKey (Dummy Sequence)	Unsigned int	4	0 ~ 42억
Business Key Concatenation	Char	1~30	30 Bytes까지 Concatenate 권장
Amount	[Unsigned]Bigint	8	금액, 이자, 세금 등 소수점 이하 없는 수 Bigint : 999,999,999,999,999,999 지원
Ratio	Numeric(9,6)	4	이율, 0% ~ 999%
Long String	Varchar	256이상	Index사용 안함
코드	Char		단, 숫자형태로 된 코드는 int 형을 사용한다. 가능하면 코드 값은 숫자형으로 변경한다.
일수/건수	Unsigned Int, Smallint,tinyint	4 2,1	Int : 999,999,999 지원
숫자 255이하	Tinyint	1	연령, 가족수 등
날짜	Date	4	IQ UNIQUE 적용 최우선 순위
여부	Tinyint	1	Y=1/N=0
기타 숫자 칼럼	Tinyint, Smallint, Int, unsigned int, Unsigned bigint		소수점 이하 없는 수
소수점 이하 숫자	Numeric(p,s)		Precision은 4,9,18 rule에 적용됨

create domain

- 시스템이 제공하는 **built-in** 데이터 타입을 근간으로 해서 사용자가 임의로 만든 단순한 형태의 사용자 정의 데이터 타입으로 데이터베이스 일치성 향상 및 편이성을 제공합니다.

- **syntax**
  create domain *domain-name data-type [[not] null];*

- 예)
  create domain *street_address char(35)* ;
  create table twocol (id int, street street_address);

- 해당하는 도메인을 사용한 **object** 가 없는 경우에 한해서 삭제가 가능합니다.

- **syntax**
  drop domain *domain-name*;

- 예)
  drop domain *street_address*;

create table in system

- 예)

```
CREATE TABLE employee
(
  emp_id      INT              NOT NULL,
  lname       CHAR(30)         NOT NULL,
  fname       CHAR(30)         NOT NULL,
  salary      UNSIGNED INT     NOT NULL,
  dept_id     INT              NOT NULL
) IN SYSTEM
```

- 참고
  위의 테이블은 **IQ Main Store** 영역에 생성되지 않고 **Catalog Store**에 생성되며 **2GB**의 파일크기 한계를 가지고 있습니다**.** 이 한계 값에 도달하면 **IQ Server**가 **down**되거나 **hang** 상태가 되는등 여러 가지 부작용이 많습니다**.SYBASE IQ**에서 **ASA**는 단순히 **Catalog**를 위한 공간이므로 사용자들의 사용을 금하여 주십시오**.**

create table

- 예)

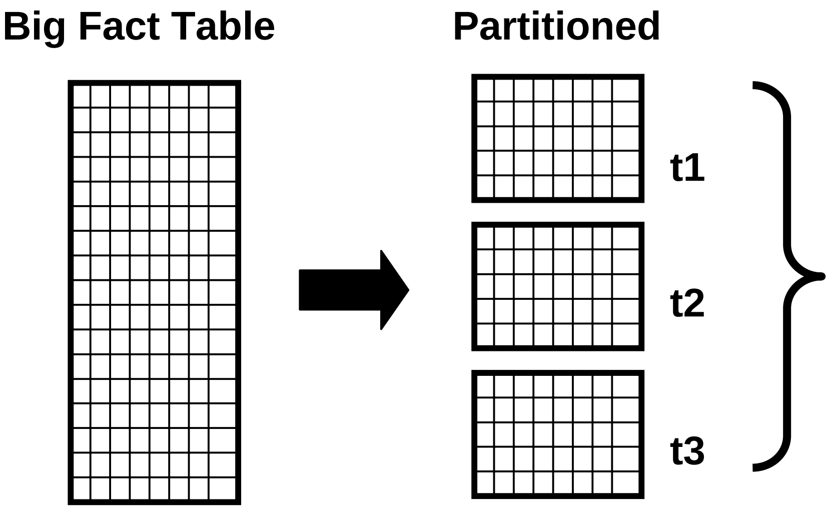
```
CREATE TABLE employee
(
  emp_id        INT               NOT NULL,
  lname         CHAR(30)          NOT NULL IQ UNIQUE(20000),
  fname         CHAR(30)          NOT NULL IQ UNIQUE(20000),
  salary        UNSIGNED INT      NOT NULL IQ UNIQUE(20000),
  dept_id       INT               NOT NULL IQ UNIQUE(50),

  PRIMARY KEY (emp_id),
  FOREIGN KEY (dept_id) REFERENCES dept(dept_id)
)
```

create table-partition

- **12.5** 이상 버젼에서는 매우 큰 팩트 테이블을 **UNION ALL VIEW**를 통해 파티션 처리하여 질의 및 로딩에 대한 성능을 향상 하였습니다**.** 단**,** 이 때 테이블 생성 및 데이터 관리는 **DBA**가 모두 수동으로 작업해야 합니다**.**

- 질의는 각각의 테이블을 하나로 묶어주는 뷰를 통해 가능하고 질의 실행시 각 파티션별로 병렬처리 합니다**.** 각 파티션된 팩트 테이블들은 디멘션 테이블에 대한 **FK** 선언이 선행되어야 합니다**.**

- 매우 커다란 테이블에 대한 로딩 작업시 발생하던 **HG** 인덱스에 대한 오버헤드를 감소시킬 수 있는 방법입니다**.**



UNION ALL VIEW

```
CREATE VIEW bigtable AS

SELECT * FROM t1
UNION ALL
SELECT * FROM t2
UNION ALL
SELECT * FROM t3
```

alter table

- **syntax**

```
ALTER TABLE  [owner.]table-name
{ADD column-definition [column-constraint]…
| ADD table-constraint
| MODIFY column-name [not] null
| DROP column-name
| DROP unique
| DROP primary key
| DROP foreign key
| RENAME new-table-name
| RENAME old-column-name TO new-column-name}
```

- **예)**

```
ALTER TABLE employee
ADD OFFICE CHAR(20) NOT NULL;                 // 데이터가 있을 때 vs. 없을 때

ALTER TABLE employee
MODIFY office NULL;                           // 데이터가 있을 때 vs. 없을 때

ALTER TABLE employee
DROP office;
```

drop table

- **syntax**
  DROP TABLE *[owner.]table-name*

- 이 테이블과 연관되어진 **JOIN INDEX**가 존재한다면 먼저 **DROP** 하고 그 후에 **DROP TABLE**을 해야 합니다**.**

- 다른 사용자가 현재 액세스 하고 있다면 다음과 같은 에러가 발생합니다**.**

  User 'xxx_user' has the row in 'xxx_table' locked

comment

- 테이블이나 뷰와 같은 데이터베이스 **object**에 대해 시스템 카다로그내에 추가 정보를 저장하기 위해 사용 합니다

- **syntax**

```
COMMENT ON
{ COLUMN [owner.]table-name.column-name
  | FOREIGN KEY [owner.]table-name.role-name
  | INDEX [owner.]index-name
  | PROCEDURE [owner.]procedure-name
  | TABLE [owner.]table-name
  ….
  IS comment;
```

- 예)

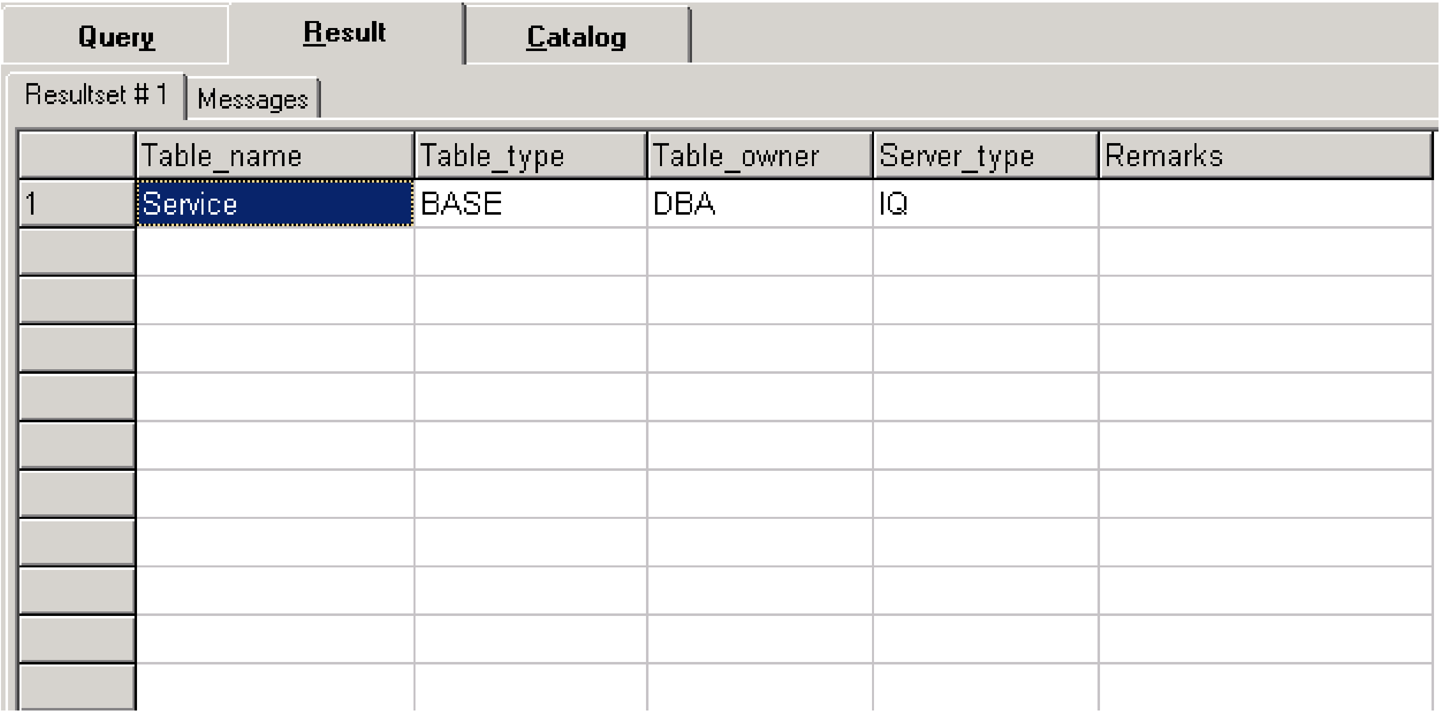
```
COMMENT ON TABLE employee is “Employee Information”;
COMMENT ON TABLE employee is null;
```

- 참고
  이렇게 저장된 정보는 **sp_iqtable**이나 **sp_iqcolumn**등과 같은 **stored procedure** 결과의 **remarks** 컬럼에 출력됩니다.

sp_iqtable

- syntax

  sp_iqtable *table-name*;

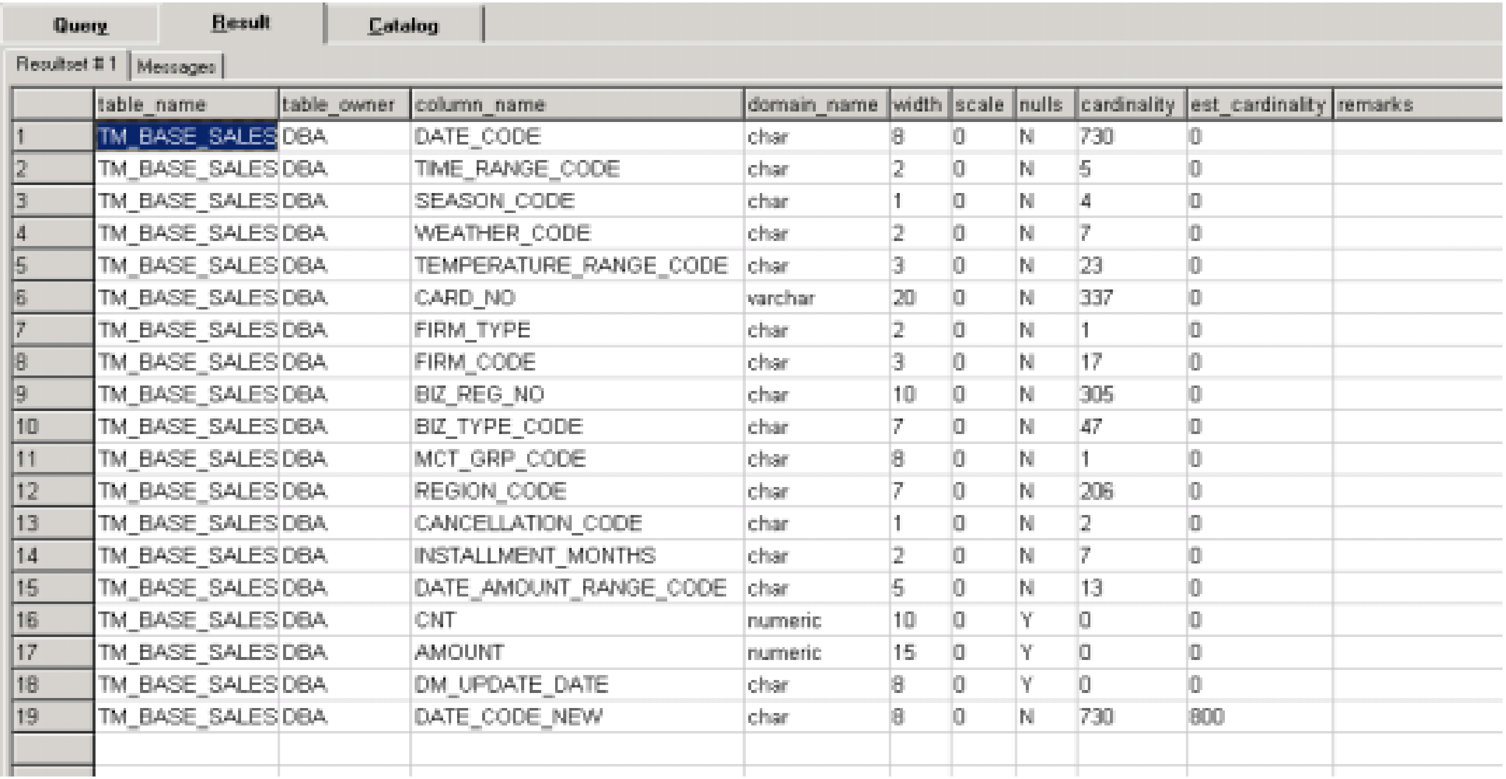


	Table_name	Table_type	Table_owner	Server_type	Remarks
1	Service	BASE	DBA	IQ	

sp_iqcolumn

- syntax

  sp_iqcolumn ***table-name*****;**

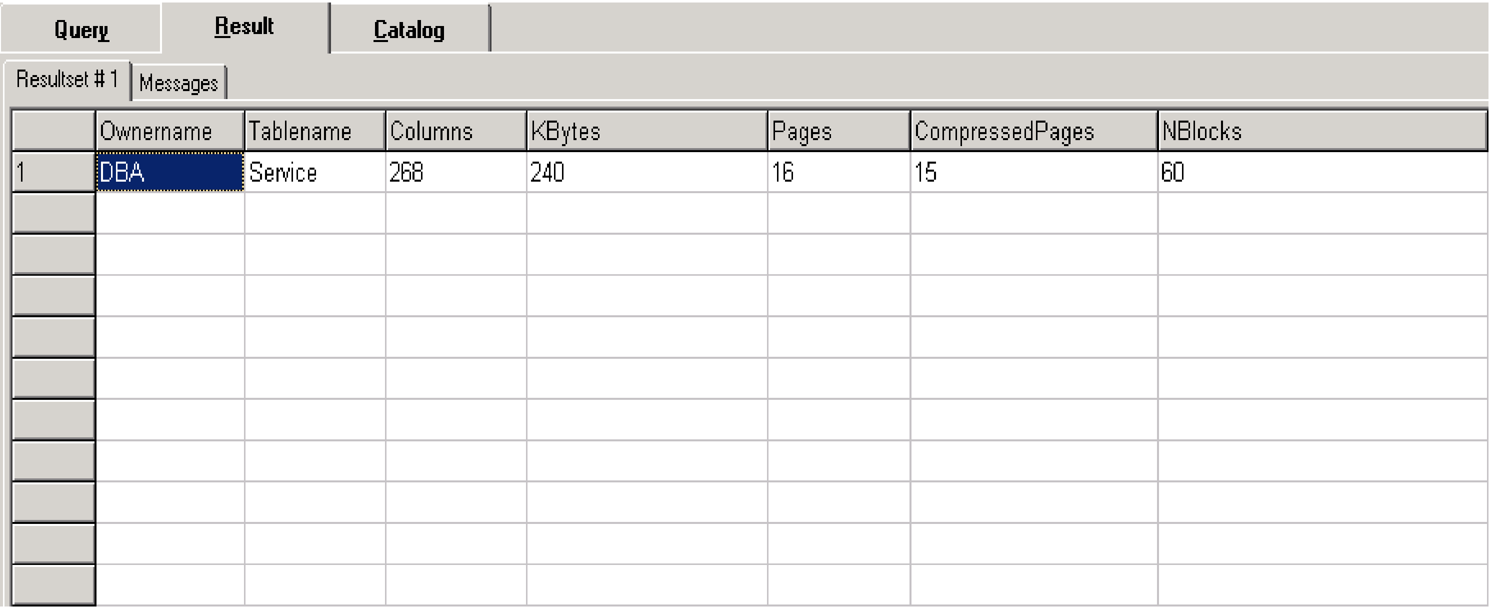


	table_name	table_owner	column_name	domain_name	width	scale	nulls	cardinality	est_cardinality	remarks
1	TM_BASE_SALES	DBA	DATE_CODE	char	8	0	N	730	0	
2	TM_BASE_SALES	DBA	TIME_RANGE_CODE	char	2	0	N	5	0	
3	TM_BASE_SALES	DBA	SEASON_CODE	char	1	0	N	4	0	
4	TM_BASE_SALES	DBA	WEATHER_CODE	char	2	0	N	7	0	
5	TM_BASE_SALES	DBA	TEMPERATURE_RANGE_CODE	char	3	0	N	23	0	
6	TM_BASE_SALES	DBA	CARD_NO	varchar	20	0	N	337	0	
7	TM_BASE_SALES	DBA	FIRM_TYPE	char	2	0	N	1	0	
8	TM_BASE_SALES	DBA	FIRM_CODE	char	3	0	N	17	0	
9	TM_BASE_SALES	DBA	BIZ_REG_NO	char	10	0	N	305	0	
10	TM_BASE_SALES	DBA	BIZ_TYPE_CODE	char	7	0	N	47	0	
11	TM_BASE_SALES	DBA	MCT_GRP_CODE	char	8	0	N	1	0	
12	TM_BASE_SALES	DBA	REGION_CODE	char	7	0	N	206	0	
13	TM_BASE_SALES	DBA	CANCELLATION_CODE	char	1	0	N	2	0	
14	TM_BASE_SALES	DBA	INSTALLMENT_MONTHS	char	2	0	N	7	0	
15	TM_BASE_SALES	DBA	DATE_AMOUNT_RANGE_CODE	char	5	0	N	13	0	
16	TM_BASE_SALES	DBA	CNT	numeric	10	0	Y	0	0	
17	TM_BASE_SALES	DBA	AMOUNT	numeric	15	0	Y	0	0	
18	TM_BASE_SALES	DBA	DM_UPDATE_DATE	char	8	0	Y	0	0	
19	TM_BASE_SALES	DBA	DATE_CODE_NEW	char	8	0	N	730	800	

sp_iqtablesize

- syntax

  sp_iqtablesize ‘[owner.]*table-name*’;



	Ownername	Tablename	Columns	KBytes	Pages	CompressedPages	NBlocks
1	DBA	Service	268	240	16	15	60

create view

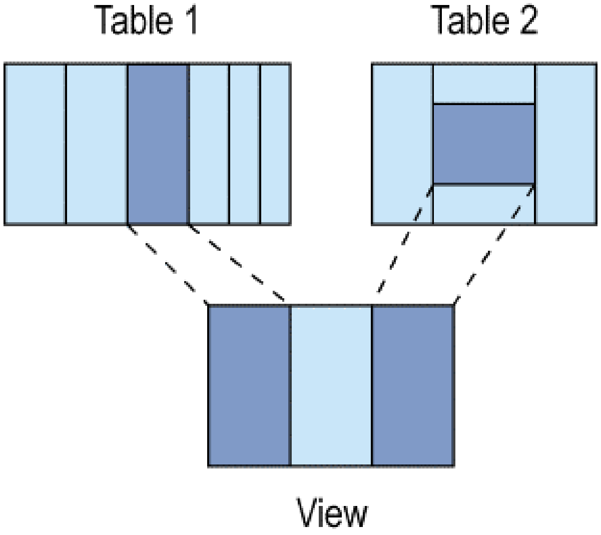
- 실제 데이터를 저장하지는 않으면서 마치 테이블처럼 사용되어지는 가상의 테이블입니다. 뷰가 참조되는 시점에 카다로그로 부터 뷰의 정의를 읽어 뷰를 형성하는 기본 테이블이 참조합니다.

- **syntax**

  CREATE VIEW *view-name [(column-name,…)* **]**
  AS *select-statement-without-order-by*
  [with check option];

- 예)

```
CREATE VIEW emp_dept
AS
SELECT emp_lname,
              emp_fname,
              dept_name
FROM employee e, department d
WHERE e.dept_id = d.dept_id;
```



drop view

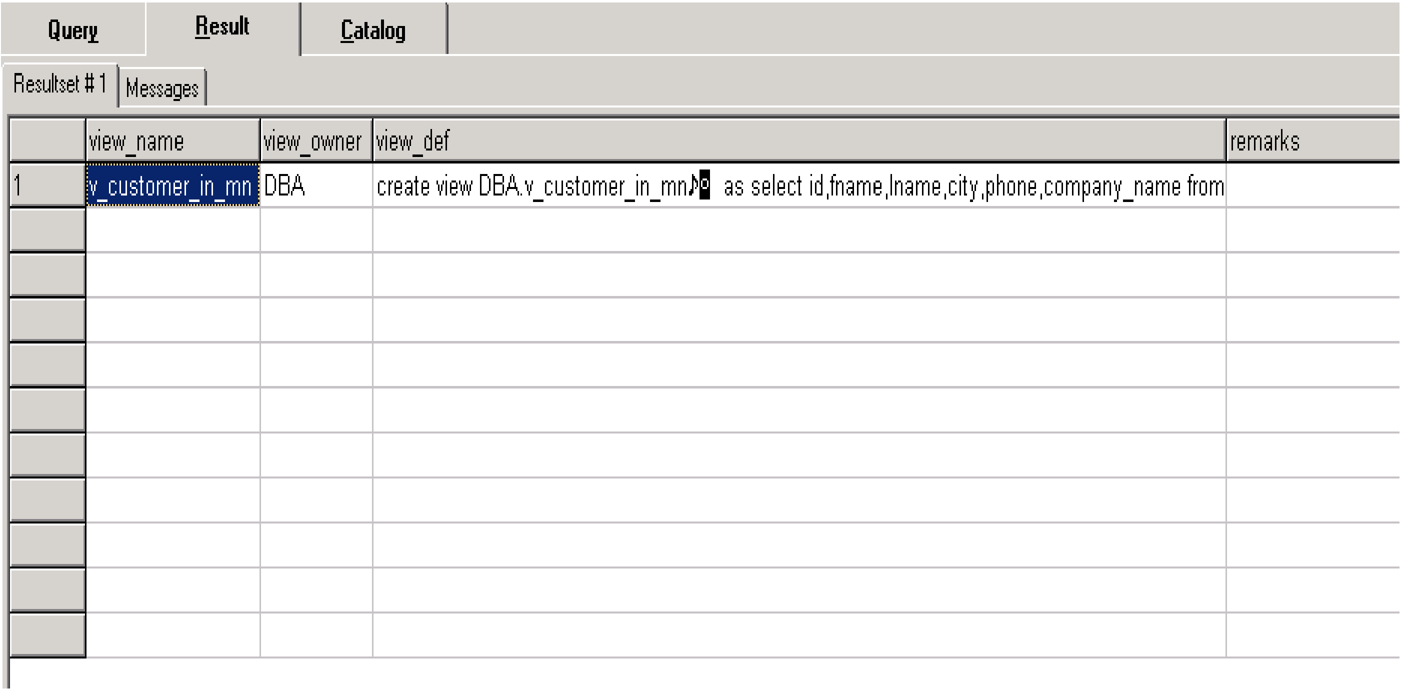
- **syntax**
  DROP VIEW *view-name***;**

- 예)
  DROP VIEW ***emp_dept;***

sp_iqview

- syntax

  sp_iqview ***view-name*****;**



	view_name	view_owner	view_def	remarks
1	v_customer_in_mn	DBA	create view DBA.v_customer_in_mn♪◘ as select id,fname,lname,city,phone,company_name from	

create index

- **syntax**
  CREATE [UNIQUE] [index-type] INDEX *index-name*
  ON *[owner.]table-name(column-name)   [notify interger]*

  index-type : CMP, HG, HNG, LF, WD, DATE, TIME, DTTM

- **UNIQUE를 생략하면 NON_UNIQUE한 인덱스가, index-type을 생략하면 HG 인덱스가 생성됩니다.**

- **FP인덱스를 제외한 각 인덱스는 데이터의 적재 전/후에 만들어 질 수 있으나 데이터 적재 전에 인덱스를 먼저 만들고 그 후에 로딩하는 편이 데이터를 로딩한 후에 인덱스를 만드는 것 보다 빠릅니다.**

- **복합 인덱스는 UNIQUE/NON_UNIQUE HG 인덱스인 경우를 제외하고는 현재 버전까지는 만들 수 없으며 복합 인덱스로 HG 인덱스를 생성하는 경우에도 그 복합 인덱스를 구성하는 개별 컬럼 또한 인덱스 생성 조건에 맞춰 인덱스를 생성해야 합니다.**

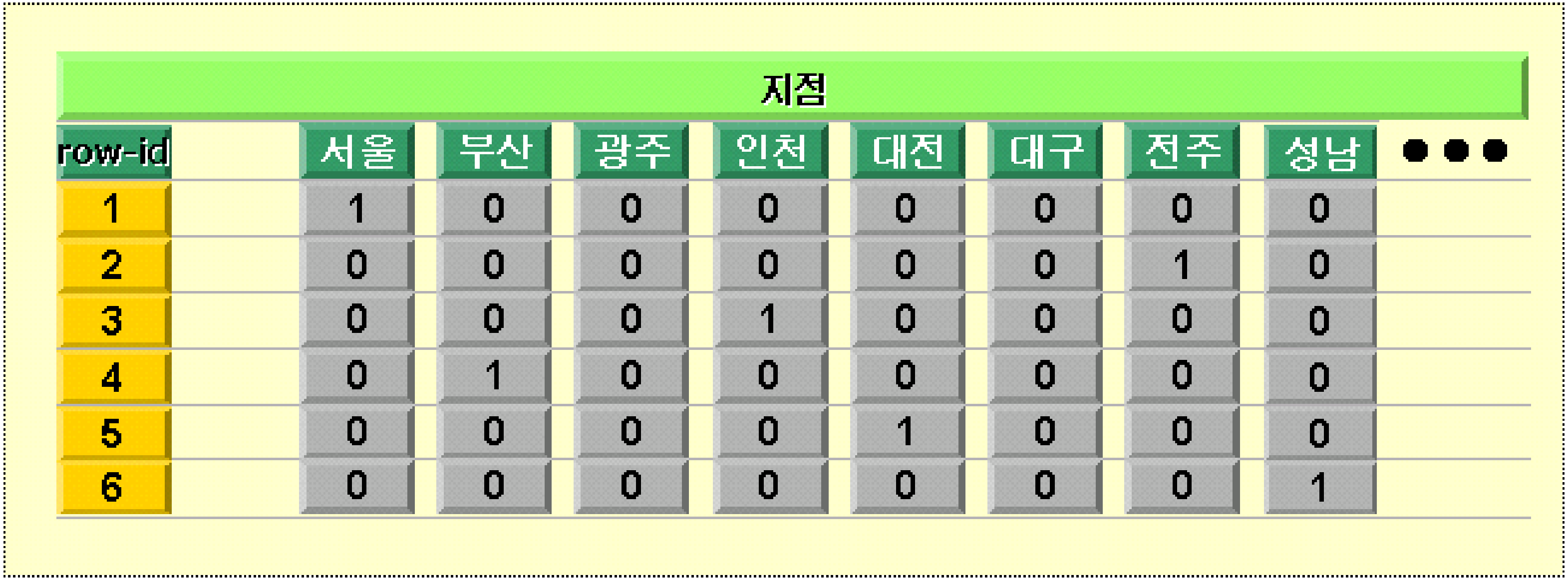
- **FP 인덱스는 CREATE INDEX 를 통해서는 결코 만들 수 없고 CREATE TABLE를 통해서만 생성 가능 합니다**

- **PRIMARY KEY constraint와 UNIQUE constraint에 의해서 UNIQUE HG 인덱스가 자동으로 만들어 지며 이렇게 만들어진 인덱스는 DROP INDEX에 의해 삭제될 수 없고 ALTER TABLE을 통해서만 삭제 가능합니다.**

create index-LF index

- 카디날리티가 비교적 낮은(1500 이하) 컬럼에 가장 적합한 인덱스로 조인, 그룹, 범위와 같은 모든 연산에 가장 빠른 성능을 나타내는 인덱스 입니다. 데이터를 **Bit-Map**으로 저장하며 처리 또한 **Bit Processing**으로 수행하므로 **HG** 인덱스와 함께 가장 빠른 처리 속도를 나타냅니다.

- 기술적으로 카디날리티 **10,000**까지 지원 가능 하나 **1,500** 이하인 경우 사용을 권장하며 일반적으로 원시 데이터의 **70%**로 인덱스를 생성합니다



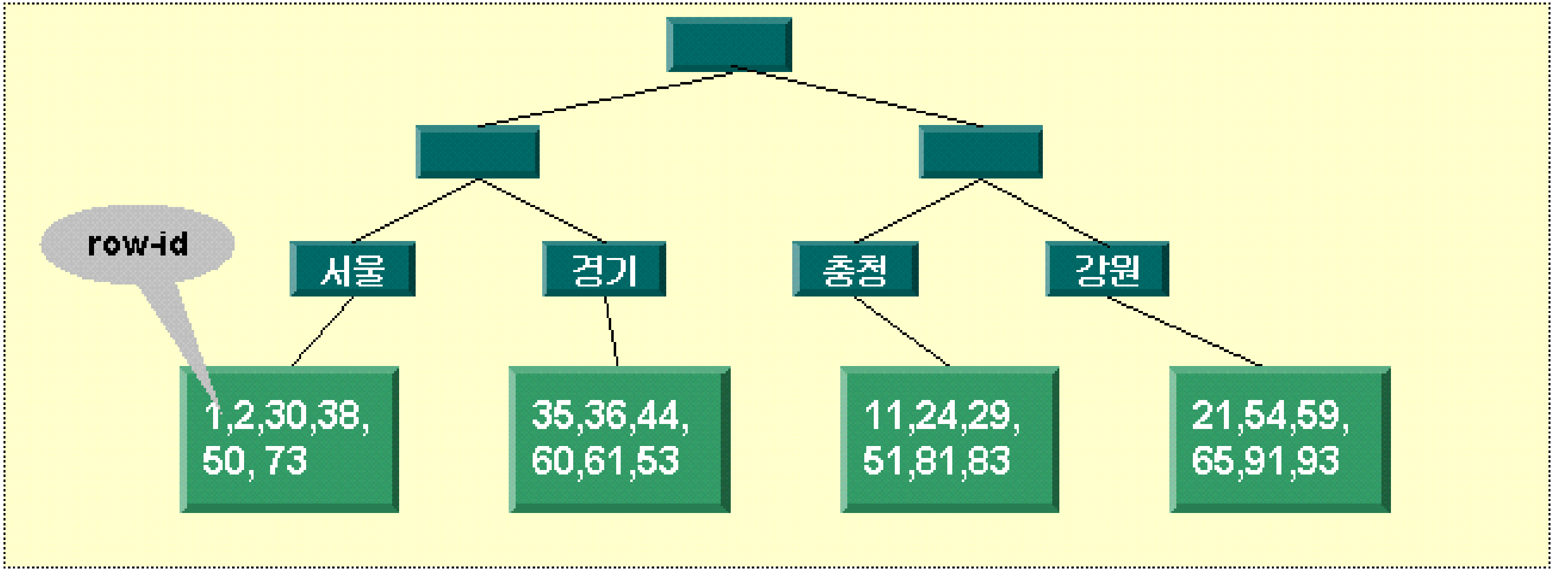
	지점								
row-id	서울	부산	광주	인천	대전	대구	전주	성남	●●●
1	1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	1	0	
3	0	0	0	1	0	0	0	0	
4	0	1	0	0	0	0	0	0	
5	0	0	0	0	1	0	0	0	
6	0	0	0	0	0	0	0	1	

create index-HG index

- 카디날리티가 **1,500** 이상이고 조인**,** 그룹**,** 범위 그리고 특정의 레코드를 찾는 연산을 수행하는 컬럼에 이용하는 인덱스로 **B-Tree**구조를 가지고 있는 인덱스 입니다**.**

- 유일성을 보장하는 경우에는 **UNIQUE HG**를 사용할 것을 권장하고 일반적으로 원시 데이터의 **120%**로 인덱스를 생성하며 데이터 적재 작업시에 많은 부담을 주는 인덱스이므로 한 테이블에 너무 많은 **HG** 인덱스가 만들어 지지 않도록 주의 바랍니다**.** 경험상 테이블당 **4~5**개의 **HG** 인덱스가 적당합니다**.**

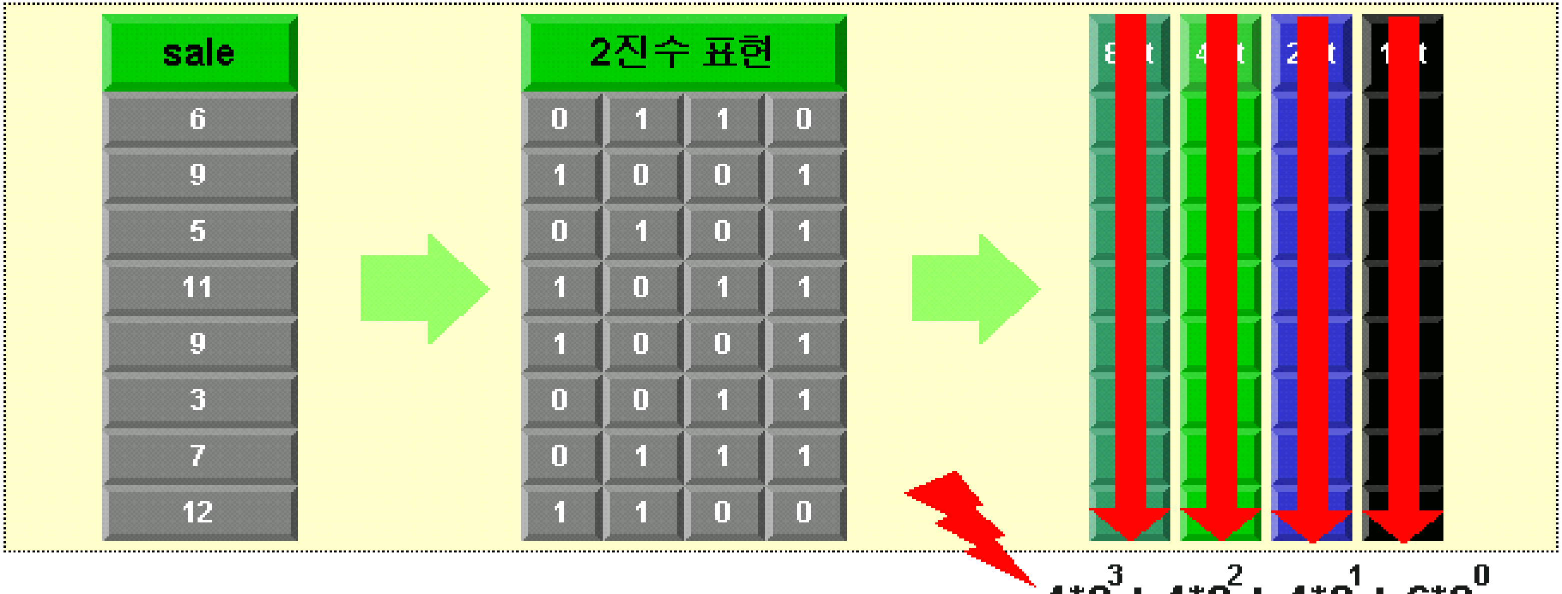
- **CREATE TABLE**의 **UNIQUE**나 **PRIMARY KEY constraint**에 의해 자동 생성됩니다



create index-HNG index

- SUM(), AVG()와 같은 aggregate 함수의 대상이 되는 수치 컬럼이나 범위 질의에 사용되는 컬럼에 사용하는 인덱스로 카디날리티와 무관하며 선호하는 데이터 타입은 INT 계열입니다. 만약 DATE 계열의 컬럼에 대한 범위 질의가 발생한다면 HNG보다는 DATE 인덱스의 사용을 권장합니다.

- 이미 LF 혹은 HG와 같은 인덱스가 생성된 컬럼이라고 해도 그 컬럼이 위와 같은 조건에 만족되면 생성할 것을 권장하며 일반적으로 원시 데이터의 70%로 인덱스를 생성합니다



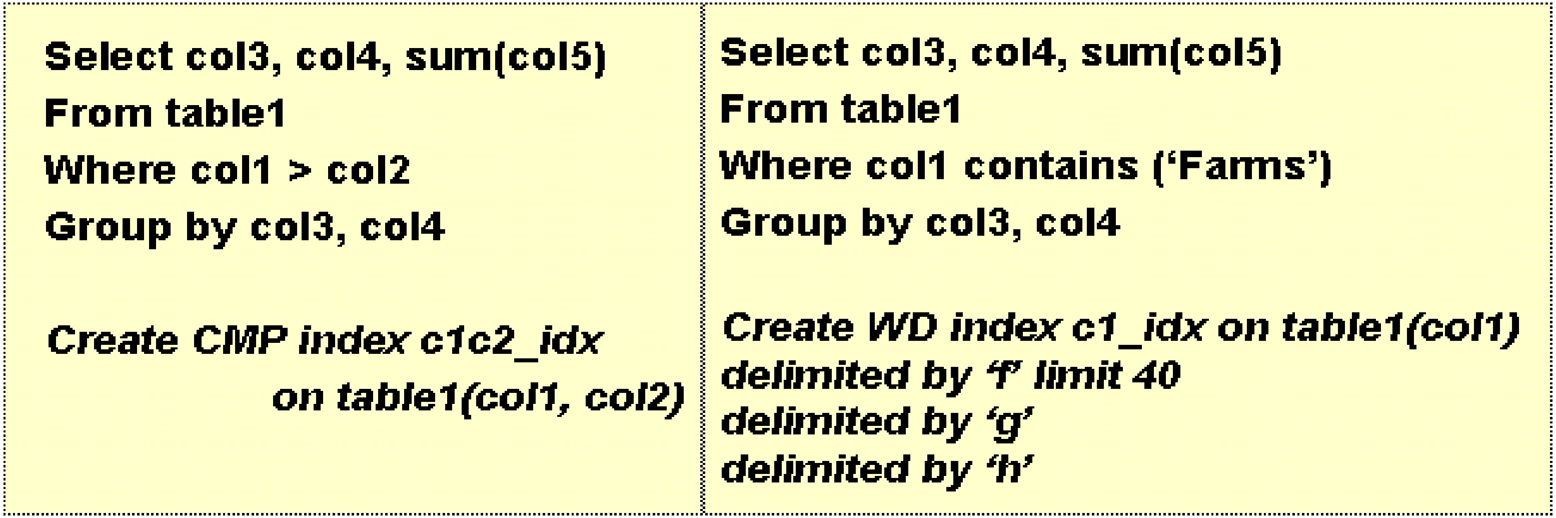
sale	2진수 표현			
6	0	1	1	0
9	1	0	0	1
5	0	1	0	1
11	1	0	1	1
9	1	0	0	1
3	0	0	1	1
7	0	1	1	1
12	1	1	0	0

$$4*2^3 + 4*2^2 + 4*2^1 + 6*2^0$$

create index-CMP, WD index

- **CMP** 인덱스는 카디날리티와 무관하고 컬럼 간의 크기를 비교할 때 사용되는 인덱스로 동일한 데이터 타입간의 비교를 보다 빠르게 하는 인덱싱 기법이며 일반적으로 원시 데이터의 **70%**로 인덱스를 생성합니다.

- **WD** 인덱스는 카디날리티와 무관하며 주소나 설명과 같이 **long varchar** 컬럼에 특정 문자를 빠르게 검색할 수 있는 인덱싱 기법으로 컬럼의 사이즈가 너무 커서 인덱싱하기 어려운 그러나 반드시 필요한 컬럼을 위한 인덱스 입니다. 일반적으로 원시 데이터의 **120%**로 인덱스를 생성합니다.



```
Select col3, col4, sum(col5)
From table1
Where col1 > col2
Group by col3, col4

Create CMP index c1c2_idx
                on table1(col1, col2)
```

```
Select col3, col4, sum(col5)
From table1
Where col1 contains (‘Farms’)
Group by col3, col4

Create WD index c1_idx on table1(col1)
delimited by ‘f’ limit 40
delimited by ‘g’
delimited by ‘h’
```

create index-DATE, TIME, DTTM

- DATE 계열의 데이터 타입을 사용하는 컬럼을 위해 사용 가능한 인덱스로 날짜관련 함수 적용시 기존 인덱스를 사용하면 인덱스를 사용하지 못하는 단점을 보완하기 위해 만들었습니다. (DATE 계열 : DATE, DATETIME, TIME)

- 카디날리티와 무관하며 UNIQUE를 선언할 수 없고 오직 DATE 관련된 컬럼에만 적용 가능합니다.

- 해당 컬럼에 대한 범위 질의, datepart 질의에 사용되며 범위 질의에서는HNG 인덱스 보다 빠른 성능을 보이나 EQUALITY 질의는 LF/HG 보다 느립니다.

- 지원되는 질의는 다음과 같습니다.
  - datepart 함수: Month, Day, Week, Quarter, Year…
  - 범위 : >, >=, <, <=, <>, =, between

- 지원되지 않는 질의는 다음과 같습니다.
  - Aggregate 함수 : sum, avg, min, max

begin parallel IQ

- **syntax**
  BEGIN PARALLEL IQ
  create-index-statement-list;
  END PARALLEL IQ

- **SYBASE IQ**는 인덱스를 생성한 후에 데이터를 로딩하는 속도가 데이터를 로딩한 후 인덱스를 만드는 속도보다 빠릅니다. 그래서 일반적으로 데이터 적재 전에 인덱스를 생성합니다.

- 하지만 이미 데이터가 존재하는 상황에서 인덱스를 만들 필요성이 있을 때 사용할 수 있는 명령어입니다. 이 명령어를 사용하면 **create-index-statement-list**에 기술된 명령어는 기본적으로 병렬로 처리됩니다.

- 이 명령어는 **create-index-statement-list** 전체에 대한 어떤 오류가 발생해도 롤백되는 **ATOMIC** 형태로 동작되며 한 컬럼에 대한 여러 개의 인덱스를 생성하는 것은 직렬로 처리됩니다.

- 참고
  create-index-statement-list의 적정 수치는 **CPU**수 **– 1** 이며 이 명령어가 실행중인 동안에 발생하는 어떤 **DDL**도 실행이 불가능합니다.

drop index

- **syntax**
  DROP INDEX *[[owner.]table-name.] index-name*

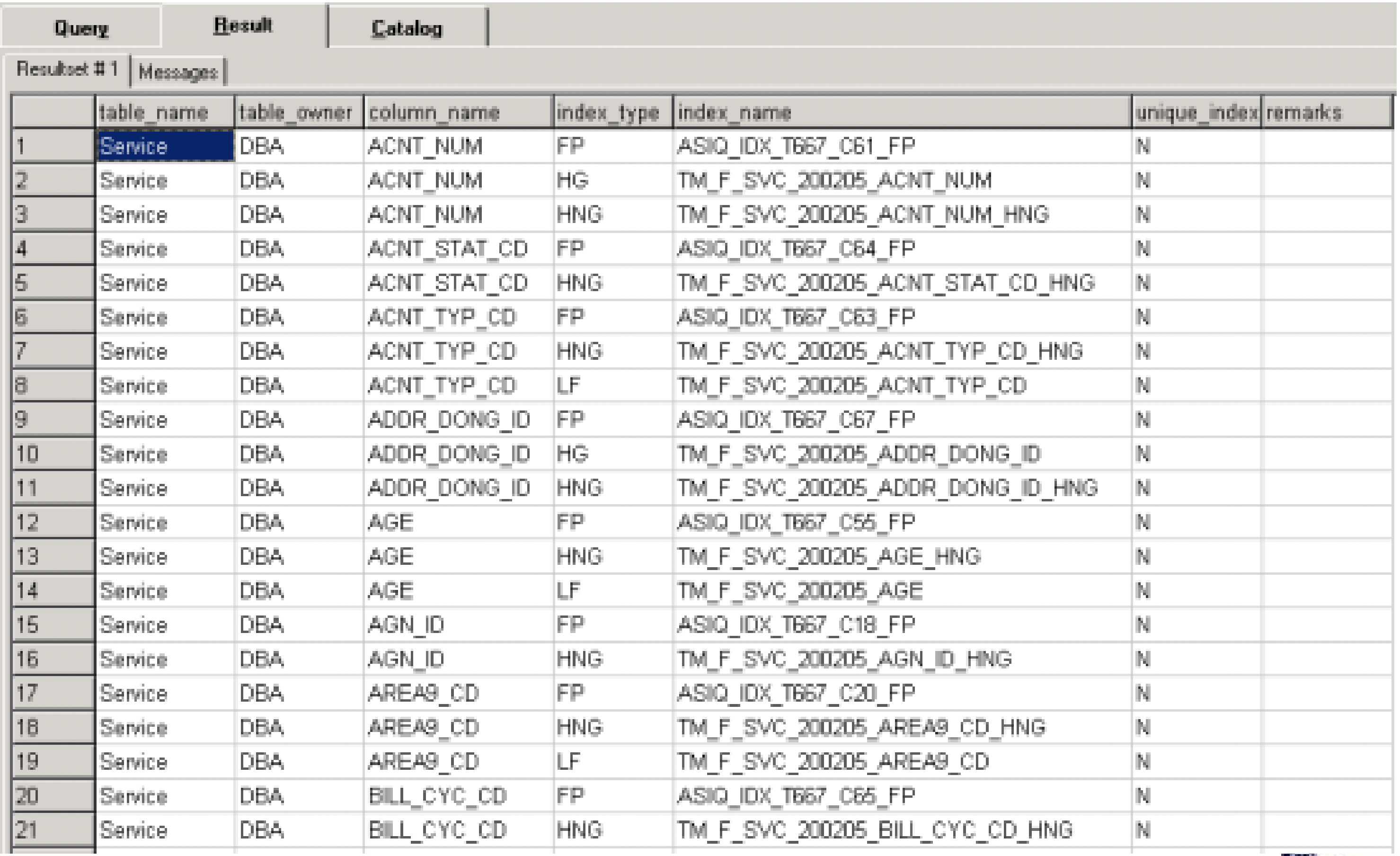
- **FP 인덱스와 PRIMARY KEY, UNIQUE constraint에 의해 자동으로 생성되어진 HG 인덱스는 이 명령어에 의해 삭제가 불가능합니다.**

- **참고**
  CREATE INDEX에 의해 해당 컬럼의 카디날리티 정보가 카다로그에 저장되어 실행계획 작성시 사용되는데 HG와 LF 인덱스만 대상입니다. 이런 카디날리티 정보는 sp_iqcolumn의 cardinality 컬럼에 보여집니다.

sp_iqindex

- syntax

  sp_iqindex ***table-name***;

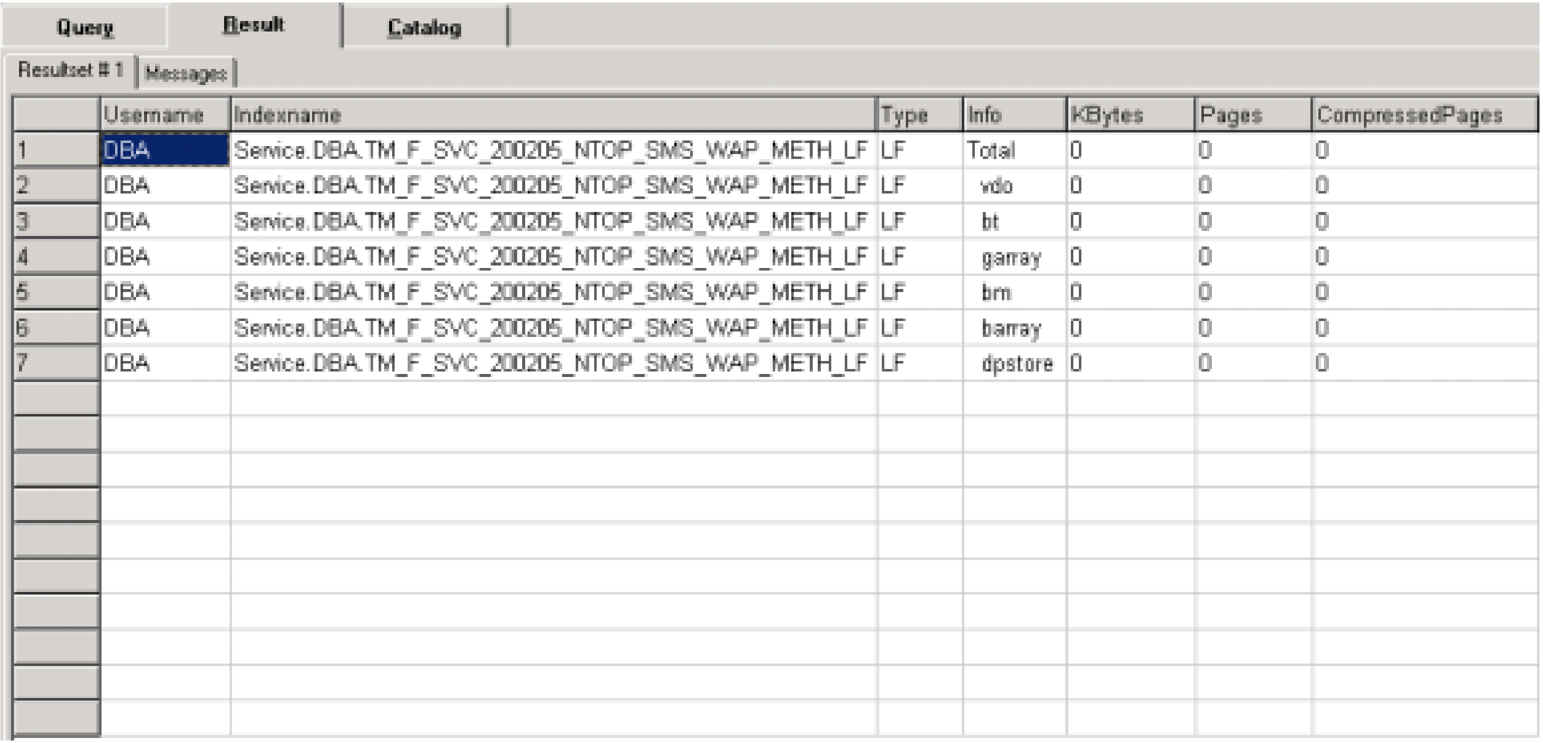


	table_name	table_owner	column_name	index_type	index_name	unique_index	remarks
1	Service	DBA	ACNT_NUM	FP	ASIQ_IDX_T667_C61_FP	N	
2	Service	DBA	ACNT_NUM	HG	TM_F_SVC_200205_ACNT_NUM	N	
3	Service	DBA	ACNT_NUM	HNG	TM_F_SVC_200205_ACNT_NUM_HNG	N	
4	Service	DBA	ACNT_STAT_CD	FP	ASIQ_IDX_T667_C64_FP	N	
5	Service	DBA	ACNT_STAT_CD	HNG	TM_F_SVC_200205_ACNT_STAT_CD_HNG	N	
6	Service	DBA	ACNT_TYP_CD	FP	ASIQ_IDX_T667_C63_FP	N	
7	Service	DBA	ACNT_TYP_CD	HNG	TM_F_SVC_200205_ACNT_TYP_CD_HNG	N	
8	Service	DBA	ACNT_TYP_CD	LF	TM_F_SVC_200205_ACNT_TYP_CD	N	
9	Service	DBA	ADDR_DONG_ID	FP	ASIQ_IDX_T667_C67_FP	N	
10	Service	DBA	ADDR_DONG_ID	HG	TM_F_SVC_200205_ADDR_DONG_ID	N	
11	Service	DBA	ADDR_DONG_ID	HNG	TM_F_SVC_200205_ADDR_DONG_ID_HNG	N	
12	Service	DBA	AGE	FP	ASIQ_IDX_T667_C55_FP	N	
13	Service	DBA	AGE	HNG	TM_F_SVC_200205_AGE_HNG	N	
14	Service	DBA	AGE	LF	TM_F_SVC_200205_AGE	N	
15	Service	DBA	AGN_ID	FP	ASIQ_IDX_T667_C18_FP	N	
16	Service	DBA	AGN_ID	HNG	TM_F_SVC_200205_AGN_ID_HNG	N	
17	Service	DBA	AREA9_CD	FP	ASIQ_IDX_T667_C20_FP	N	
18	Service	DBA	AREA9_CD	HNG	TM_F_SVC_200205_AREA9_CD_HNG	N	
19	Service	DBA	AREA9_CD	LF	TM_F_SVC_200205_AREA9_CD	N	
20	Service	DBA	BILL_CYC_CD	FP	ASIQ_IDX_T667_C65_FP	N	
21	Service	DBA	BILL_CYC_CD	HNG	TM_F_SVC_200205_BILL_CYC_CD_HNG	N	

sp_iqindexsize

- syntax

  sp_iqindexsize *index-name*;



	Username	Indexname	Type	Info	KBytes	Pages	CompressedPages
1	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	Total	0	0	0
2	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	vdo	0	0	0
3	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	bt	0	0	0
4	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	garray	0	0	0
5	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	bm	0	0	0
6	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	barray	0	0	0
7	DBA	Service.DBA.TM_F_SVC_200205_NTOP_SMS_WAP_METH_LF	LF	dpstore	0	0	0

Loading and Extracting data

load

- **DW** 는 **OLTP**시스템과 달리 데이터 입력 방법이 실시간이 아니라 업무의 특성에 따라 일정 주기를 갖고 **bulk**로 적재하는 것이 일반적입니다**. SYBASE IQ**도 데이터를 **bulk**로 적재하는 것을 기본으로 구성하고 있습니다.

- 적재를 위한 원시 데이터는 일정한 포맷을 갖추고 **flat file** 형태로 제공되며 포맷형태에 따라 **binary file**과 **ascii file**로 구분됩니다**.**

- 한 테이블에 적재가 진행되고 있을 때 어떤 다른 사용자도 그 테이블에 대해 쓸 수 있는 권한은 없습니다**.** 물론 읽기는 **snapshot versioning** 이라는 기법을 통해 어떤 사용자도 가능합니다**.**

- 적재 대상이 되는 테이블의 데이터 존재여부에 관계없이 추가됩니다**.** 즉 존재하는 데이터를 **replace** 하는 **UPSERT** 기능은 제공되지 않습니다**.**

- **LOAD** 명령어는 옵션에 따라 적재되는 데이터가 모두다 입력되거나 아니면 전혀 입력되지 않을 수 있습니다**.**

- 또한 **LOAD** 명령어는 옵션에 따라 적재시 오류가 발생한 데이터와 오류 원인에 대해 파일로 저장할 수 있는 기능이 존재합니다**.**

- 임시 테이블도 일반 다른 영구적인 테이블처럼 적재가 가능합니다**.**

load

- **syntax**

```
LOAD TABLE  [owner.] table-name
[ ( column-name [column-spec] | filler (filler-type) ) ]
FROM  'file-name'[, . . . . . . . ]                          // 원시 데이터 파일 이름
 [ FORMAT  'ascii' | 'binary' ]                               // 원시 데이터 파일 포맷
 [ DELIMITED BY 'string' ]                                    // 컬럼 분리 구분자
 [ QUOTES OFF ]                                               // off
 [ ESCAPES OFF ]                                              // off
 [ WITH CHECKPOINT ON | OFF ]                                 // checkpoint 실행여부
 [ ROW DELIMITED BY 'string' ]                                // row 분리 구분자
 [ LIMIT number-of-rows ]                                     // 총 입력 건수 제한
 [ NOTIFY   number-of-rows ]                                  // message display 건수
 [ ON FILE ERROR rollback | finish | continue ]               // multi-file인 경우 사용
 [ SKIP number-of-row ]                                       // 처음부터 몇 건을 생략
 [ START ROW ID number-of-row ]                               // 몇 번째 row 부터 입력
 [ BLOCK FACTOR number | block size number ]                  // block당 row 수
 [ IGNORE CONSTRAINT constraint-type [,…]]                    // 제약사항 위반 횟수 정의
 [ MESSAGE LOG 'file-name'                                    // 오류 기록 파일 정의
   ROW LOG 'file-name'                                        // 오류데이터 기록 파일 정의
   [ONLY LOG log-what, […]]                                   // 오류 정의
 [ LOG DELIMITED BY 'string' ]                                // 오류 데이터 필드 분리자
 . . . . . .
```

load-column spec

- **FROM절에 기술한 소스파일의 필드 포맷과 LOAD TABLE절에서 기술한 테이블 컬럼의 매핑 정보를 기술합니다.**
  - ● **column-name : 적재를 위한 테이블의 컬럼 이름**
  - ● **ascii(numer-of-byte) : number-of-byte 크기의 필드 값을 대상 컬럼과 매핑**
  - ● **‘field delimiter’ : field delimiter 전 까지의 값을 대상 컬럼과 매핑**
  - ● **date(format) : format 형태의 필드 값을 대상 컬럼의 DATE 타입과 매핑**
  - ● **nulls (blanks | zeros | ‘literal’ [,’literal’] . . .) : 괄호안의 내용을 NULL로 변환**
  - ● **filler(number-of-byte) : number-of-byte 만큼의 길이를 건너 뜀**

- **예)**

```
LOAD TABLE sales
(   l_orderkey     ‘|’,
    l_quantity     ascii(4) null(blanks, zeros, ‘9999’),
    filler(1),
    l_shipdate     date(‘YYYY/MM/DD’) null(blanks, zeros, ’0000/00/00’)  )
FROM . . . . . . .
```

```
0001|100002003/10/15  ⎫
0002|200000000/00/00  ⎪
0003|999902003/10/15  ⎬  file format
0004|100002003/10/11  ⎪
................      ⎭
```

load-from clause

- 여러 개의 소스파일을 콤마로 분리하여 하나의 **FROM**절에 사용할 수 있습니다. 즉 여러 개의 소스파일을 하나의 **LOAD** 명령어로 적재할 수 있습니다.

- 이렇게 여러 개의 소스파일을 통해 적재작업 수행할 때 연관있는 옵션으로는 **ON FILE ERROR** 라고 하는 옵션이 있는데 이는 적재 작업을 진행하고 있을 때 오류가 발생하면 어떻게 할 것인지 기술해 주는 옵션 입니다. 단, 여기서의 오류는 해당되는 파일이 없다거나 읽기 권한이 없을 때 발생하는 오류만을 지칭합니다.

  - **ON FILE ERROR ROLLBACK :** 디폴트 옵션으로서 적재 전체를 취소합니다
  - **ON FILE ERROR FINISH :** 현재까지 수행되어진 파일까지만 적재작업을 인정합니다
  - **ON FILE ERROR CONTINUE :** 오류가 발생된 파일을 건너뛰어 작업을 계속합니다

- 예)

```
LOAD TABLE sales
(
    l_orderkey    ‘|’,
   l_quantity     ascii(4) null(blanks, zeros, ‘9999’),
   filler(1),
   l_shipdate     date(‘YYYY/MM/DD’) null(blanks, zeros, ’0000/00/00’)
)
FROM ‘/data/sales1.dat’,’/data/sales2.dat’,’/data/sales3.dat’
ON FILE ERROR FINISH
. . . . . . .
```

load-delimited by option

- **column-spec**에 컬럼당 그 컬럼에 해당하는 분리자를 기술하는 방법이 아니고 적재하는 테이블의 모든 컬럼을 하나의 컬럼 분리자로 표현하는 방법입니다

- 컬럼 분리자의 디폴트는 콤마이며 **row delimited by** 옵션과 병행하여 사용해야 합니다

- 주의해야 할 점은 마지막 컬럼 뒤에도 컬럼 분리자가 있어야 하며 그 뒤에 반드시 행 분리자가 표현되어야 합니다.

- 특수문자는 다음과 같이 **16**진수로 표현가능 합니다
  - **tab : \x09, newline : \x0a, carriage return : \x0d, null : \x00**

- 예)

```
LOAD TABLE sales
(
    l_orderkey,
    l_quantity,
    l_shipdate
)
FROM '/data/sales.dat'
DELIMITED BY '|'
ROW DELIMITED BY '\n'
. . . . . . .
```

sales.dat

```
0001|1000|20031015|
0002|2000|20031012|
0003|9999|20031015|
0004|1000|20031011|
......................
```

load-row delimited by option

- **column-spec**에 마지막 컬럼에 해당하는 분리자를 기술하는 방법이 아니고 적재하는 테이블의 행 분리자를 옵션으로 표현하는 방법입니다

- **12.4.2** 부터 이 옵션을 사용하고 **newline** 으로 행 분리자를 선언해야만 병렬적재가 가능합니다.

- 주의해야 할 점은 마지막 컬럼 뒤에도 컬럼 분리자가 있어야 하며 그 뒤에 반드시 행 분리자가 표현되어야 합니다.

- 특수문자는 다음과 같이 **16**진수로 표현가능 합니다
  - **tab : \x09, newline : \x0a, carriage return : \x0d, null : \x00**

- 예)

```
LOAD TABLE sales
(
    l_orderkey,
    l_quantity,
    l_shipdate
)
FROM ‘/data/sales.dat’
DELIMITED BY ‘|’
ROW DELIMITED BY ‘\n’
. . . . . .
```

sales.dat

```
0001|1000|20031015|
0002|2000|20031012|
0003|9999|20031015|
0004|1000|20031011|
.....................
```

load-ignore constraint option

- 지정된 **constraint**에 대한 위반이 지정된 횟수보다 많이 발생하면 롤백되며 위반된 데이터는 적재되지 않습니다. 만약 각 **constraint**에 대한 위반 횟수를 **0**으로 한다면 위반된 보든 데이터는 적재되지 않고 롤백도 되지 않습니다.

- 일반적으로 **MESSAGE LOG** 절과 함께 사용되어 제약사항이 위반된 데이터를 특정한 로그 파일에 기록하는 용도로 사용합니다.

- 대상이 되는 **constraint** 종류는 다음과 같습니다.
  - ● **UNIQUE :** 유일성이 위반되는지 여부를 점검
  - ● **NULL : NOT NULL** 컬럼에 **NULL** 데이터가 입력되는지 여부를 점검
  - ● **FOREIGN KEY :** 참조 무결성을 위반하는 데이터가 입력되는지 여부를 점검
  - ● **DATA VALUE :** 선언된 데이터 타입에 위반되는 데이터가 입력되는지 여부를 점검
  - ● **ALL : UNIQUE, NULL, FOREIGN KEY, DATA VALUE**의 조합

- 예)

```
LOAD TABLE sales
…………..
…………..
IGNORE CONSTRAINT NULL 50, UNIQUE 100, ALL 125
// 51번 이상의 NULL 값 위반이 발생하거나 101번 이상의 UNIQUE 값 위반이 발생
// 하거나 1번 이상의 FOREIGN KEY 위반이 발생하거나 조합해서 총 126번 이상의
// 위반이 발생하면 롤백됩니다.
```

load-message log option

- 지정된 위반 사항에 대한 메시지를 기록하는 파일과 실제 데이터를 저장하는 파일을 기술할 수 있으며 추후에 이 두개의 파일을 이용하여 디버깅이 가능합니다.

- **MESSAGE LOG**에 기술하는 파일에는 위반 사항을 기록하며 **ROW LOG**에 기술하는 파일에는 필드 분리자로 구분된 오류 데이터를 기록합니다. 필드 분리자는 기본 값으로 콤마를 사용하고 **LOG DELIMITED BY** 옵션에 의해 다른 값으로 변경할 수 있습니다.

- 기록할 수 있는 위반 사항은 **NULL, UNIQUE, FOREIGN KEY, DATA VALUE, ALL**이 있으며 그 내용은 **IGNORE CONSTRAINT**와 같습니다.

- 여기서 말하는 **MESSAGE LOG**는 **SYBASE IQ**의 기본적인 **IQ message log** 와는 별도의 파일입니다.

- **MESSAGE LOG**나 **ROW LOG**가 이미 존재한다면 새로 기록되는 정보는 현재 기록된 정보에 추가 기록됩니다.

- **ONLY LOG**가 생략된다면 적재 시작시간과 종료시간만이 **MESSAGE LOG** 파일에 기록됩니다.

- 적재 작업이 끝나면 **IQ message log** 파일에 **constraint** 위반 횟수와 **skip**된 데이터 건수와 같은 메시지가 기록됩니다.

load-message log option

- 예)

```
LOAD TABLE sales
…………
IGNORE CONSTRAINT UNIQUE 200, NULL 50
MESSAGE LOG ‘msg.log’
ROW LOG  ‘row.log’
ONlY LOG UNIQUE, NULL, DATA VALUE
LOG DELIMITED BY ‘|’
```

- msg.log

```
2002-07-15 15:00:23 Load Table sales: Integrity Constraint Violations
1267   DATA VALUE  4
3126   UNIQUE  1
3216   NULL  3
2002-07-15 16:00:10  LOAD TABLE sales Completed
```

rowid, *type*, *column number*

- row.log

```
2002-07-15 15:00:23 Load Table sales: Integrity Constraint Violations
1267 |Mary Smith|56|M|ABCDEFG|1943/03/31|MC|
3216 |John Jones|NULL|NULL|S|1945/02/28|NULL
 …
2002-07-15 16:00:10  LOAD TABLE sales Completed
```

rowid, *delimiter*, *data*

load-etc

- **WITH CHECKPOINT** : 이 옵션은 적재가 끝난 후에 CHECKPOINT의 실행 여부를 결정합니다. 이 옵션을 on을 바꿀 것을 권장합니다. 즉 적재 후에 메모리상에 존재하던 데이터들이 모두 디스크에 반영됩니다.

- **ESCAPSES** : 적재 대상 파일의 데이터중에 역 슬래시 뒤에 나오는 문자를 특수 문자로 인정할지 여부를 결정하는 옵션으로 ASA 에서는 on/off 설정이 가능했지만 SYBASE IQ 에서는 문법상 off로만 설정해야 합니다. 이 부분을 생략하면 역 슬래시 뒤에 나오는 문자를 특수문자로 인식하는 on으로 설정되며 SYBASE IQ에서는 문법상 오류가 발생합니다

- **QUOTES** : 적재 대상 파일의 데이터중에 quote(‘) 문자를 데이터로 인정할지 여부를 결정하는 옵션으로 ASA 에서는 on/off 설정이 가능했지만 SYBASE IQ에서는 문법상 off로만 설정해야 합니다. 이 부분을 생략하면 quote 문자를 데이터 자체로 인정하지 않는 on으로 설정되며 SYBASE IQ에서는 문법상 오류가 발생합니다.

- **LIMIT** : 입력하기를 원하는 최대 건수를 기술할 수 있으며 기본 값은 0으로 입력건수에 대한 제약이 없습니다. 주로 테스트할 때 많이 사용합니다.

- **NOTIFY** : 몇 건의 데이터가 적재될 때 한번씩 메시지를 로그 파일에 기록할 것인지를 기술합니다. 기본 값은 100,000 건 이며 NOTIFY_MODULUS 라는 옵션 값이 그렇게 설정되었기 때문 입니다.

load-etc

- **SKIP** : 몇 건의 데이터를 제외하고 적재작업을 할 것인지를 기술하며 기본 값은 0으로 모든 데이터를 적재합니다.

- **START ROW ID** : partial with insert 처럼 특정 몇 개 컬럼의 적재 작업을 첫 번째 로우부터 다시 시작합니다. 명령이 LOAD 일 뿐 기본 원리는 INSERT와 같습니다.

- **FILE FORMAT** : BINARY와 ASCII를 선택할 수 있으며 수치 데이터 값이 많으면 많을 수록 BINARY 형태의 파일이 빠른 적재 성능을 나타냅니다. 또한 BINARY 파일 형태의 데이터를 적재시 각 column-spec 뒤에 WITH NULL BYTE 라는 옵션을 주어 NULL 값에 대한 처리를 기술합니다.

- 그 외 : 메뉴얼을 참조하시기 바랍니다.

load-mode

- **SYBASE IQ**의 적재 작업은 두 가지 모드로 진행됩니다**.** 하나는 **single thread** 모드이고 또 다른 하나는 **parallel thread** 모드입니다**.** 일반적으로 **SYBASE IQ**는 각 컬럼당 **1**개의 쓰레드와 각 인덱스당 **1**개의 쓰레드가 동시에 작업을 진행하는 **parallel thread** 모드에서 적재 작업을 진행합니다**.**

- 하지만 여러 가지 이유로 **single thread** 모드로 동작할 수 있으며 이렇게 동작시에는 적재 작업의 속도가 매우 떨어지게 됩니다**.** 이럴 때는 **single thread** 모드로 동작하는 원인을 파악하여 그 원인을 제거하고 정상적인 작업이 될 수 있도록 해야 합니다**.**

- **SYBASE IQ**가 적재 작업을 **single thread** 모드로 진행하는 대표적인 경우는 쓰레드의 부족**, partial with load** 작업**, ROW DELIMITED BY** 옵션의 생략입니다**.**

load-tip

- 데이터 적재 작업 전에 테이블에 필요한 인덱스를 미리 생성하시고 한 테이블에 **HG** 인덱스가 너무 많다면 파티션을 고려해 보세요.

- 테이블의 컬럼 수와 적재 대상 파일의 필드 수를 일치하십시오.

- 가능하다면 대상 파일의 포맷은 수치 데이터가 많으면 많을 수록 **BINARY** 로 하시고 **ASCII**로 하실 경우에도 고정길이 형태로 사용하십시오.

- 만약 고정길이 형태만을 사용할 수 없다면 고정 길이와 변동 길이등을 혼합하여 사용하지 말고 차라리 필드의 분리자를 사용하는 변동길이 형태를 사용하십시오. 이 때에 마지막 필드에도 컬럼 분리자를 사용하여 주십시오.

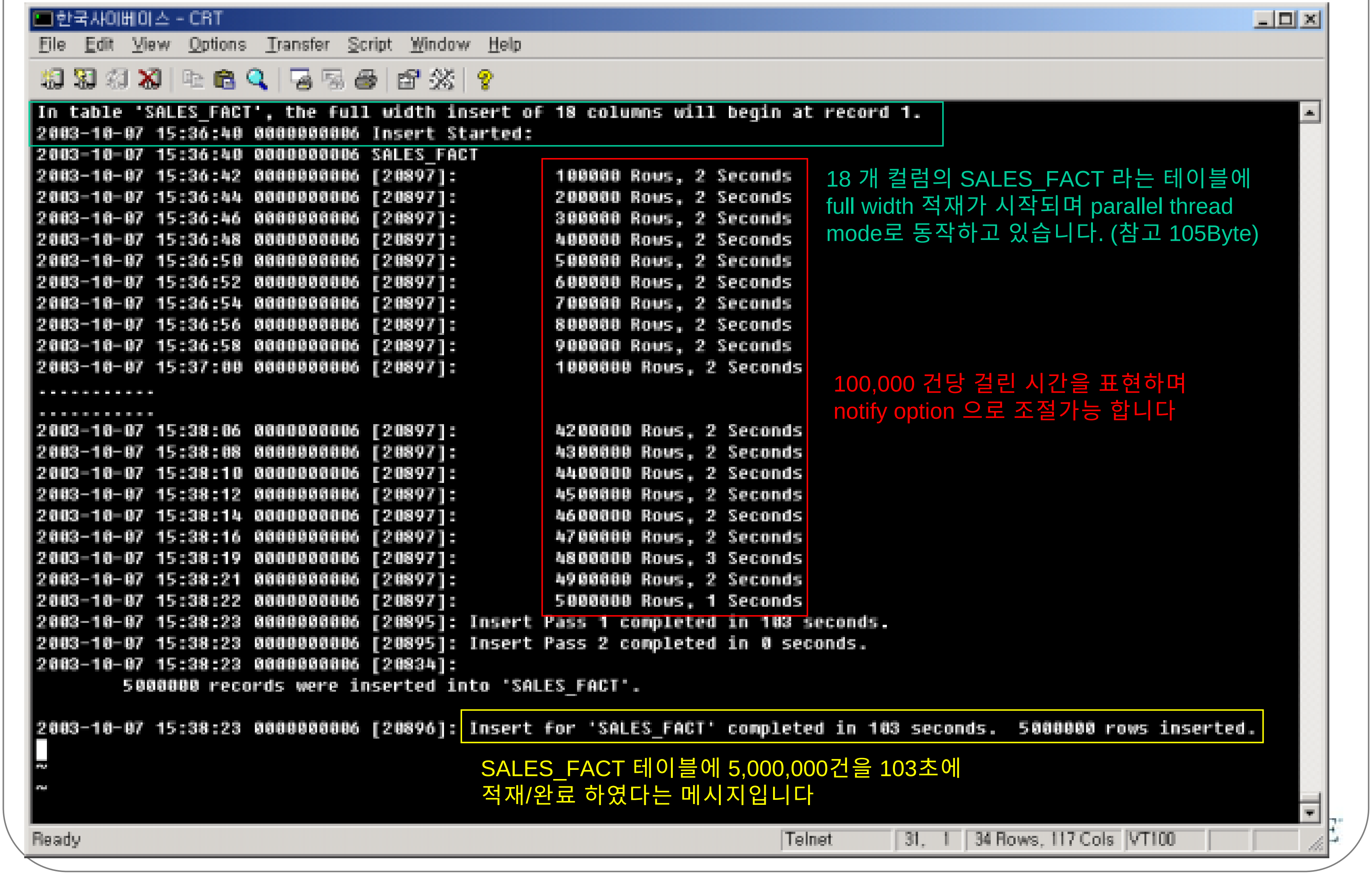
- 각 컬럼의 **column-spec**에 **NULLS( )** 옵션을 사용하여 **NULL** 데이터에 대한 정의를 기술하십시오. 일종의 데이터 클린징 로직이 될 수 있습니다.

- 적재중 잘못 기술된 데이터에 대한 추후 점검을 위해 **IGNORE CONSTRAINT** 옵션과 **MESSAGE LOG** 옵션을 사용하십시오.

- 적재 작업을 진행하면 **message log** 파일을 참조하여 정상적으로 데이터를 들어가고 있는지 확인하여 주십시오.

- 적재 작업은 가능하면 사용자 수가 적은 시간을 고려하여 실행하고 사용자가 많은 경우에도 가능하다면 적재 후에 각 사용자의 재접속을 지시하십시오.

load-message file



```
In table 'SALES_FACT', the full width insert of 18 columns will begin at record 1.
2003-10-07 15:36:40 0000000006 Insert Started:
2003-10-07 15:36:40 0000000006 SALES_FACT
2003-10-07 15:36:42 0000000006 [20897]:             100000 Rows, 2 Seconds
2003-10-07 15:36:44 0000000006 [20897]:             200000 Rows, 2 Seconds
2003-10-07 15:36:46 0000000006 [20897]:             300000 Rows, 2 Seconds
2003-10-07 15:36:48 0000000006 [20897]:             400000 Rows, 2 Seconds
2003-10-07 15:36:50 0000000006 [20897]:             500000 Rows, 2 Seconds
2003-10-07 15:36:52 0000000006 [20897]:             600000 Rows, 2 Seconds
2003-10-07 15:36:54 0000000006 [20897]:             700000 Rows, 2 Seconds
2003-10-07 15:36:56 0000000006 [20897]:             800000 Rows, 2 Seconds
2003-10-07 15:36:58 0000000006 [20897]:             900000 Rows, 2 Seconds
2003-10-07 15:37:00 0000000006 [20897]:             1000000 Rows, 2 Seconds
...........
...........
2003-10-07 15:38:06 0000000006 [20897]:             4200000 Rows, 2 Seconds
2003-10-07 15:38:08 0000000006 [20897]:             4300000 Rows, 2 Seconds
2003-10-07 15:38:10 0000000006 [20897]:             4400000 Rows, 2 Seconds
2003-10-07 15:38:12 0000000006 [20897]:             4500000 Rows, 2 Seconds
2003-10-07 15:38:14 0000000006 [20897]:             4600000 Rows, 2 Seconds
2003-10-07 15:38:16 0000000006 [20897]:             4700000 Rows, 2 Seconds
2003-10-07 15:38:19 0000000006 [20897]:             4800000 Rows, 3 Seconds
2003-10-07 15:38:21 0000000006 [20897]:             4900000 Rows, 2 Seconds
2003-10-07 15:38:22 0000000006 [20897]:             5000000 Rows, 1 Seconds
2003-10-07 15:38:23 0000000006 [20895]: Insert Pass 1 completed in 103 seconds.
2003-10-07 15:38:23 0000000006 [20895]: Insert Pass 2 completed in 0 seconds.
2003-10-07 15:38:23 0000000006 [20834]:
        5000000 records were inserted into 'SALES_FACT'.

2003-10-07 15:38:23 0000000006 [20896]: Insert for 'SALES_FACT' completed in 103 seconds.  5000000 rows inserted.
~
~
```

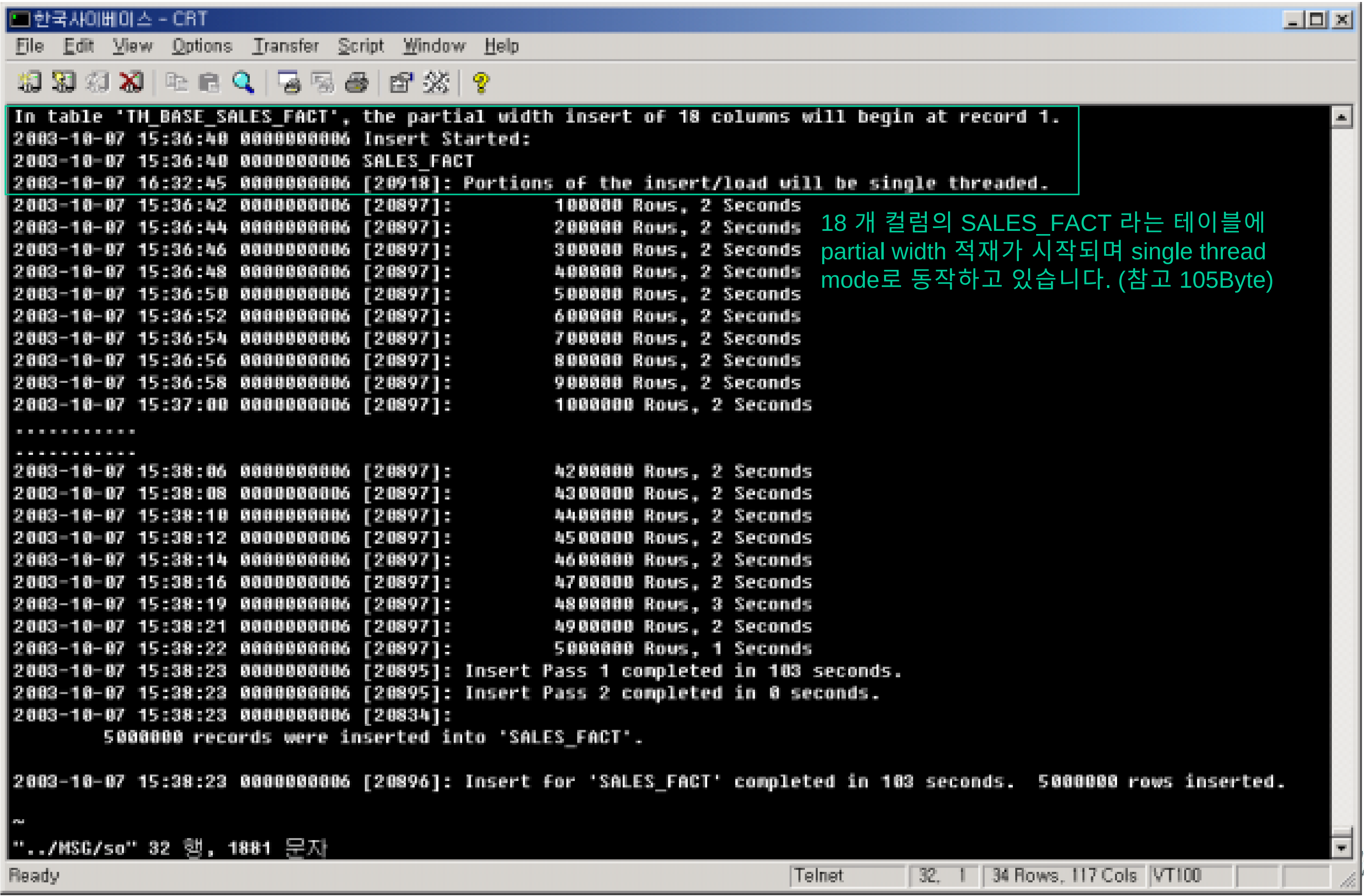
18 개 컬럼의 SALES_FACT 라는 테이블에 full width 적재가 시작되며 parallel thread mode로 동작하고 있습니다. (참고 105Byte)

100,000 건당 걸린 시간을 표현하며 notify option 으로 조절가능 합니다

SALES_FACT 테이블에 5,000,000건을 103초에 적재/완료 하였다는 메시지입니다

이 문서를 상업적인 목적으로 사용하거나 배포하는 것을 금지합니다.

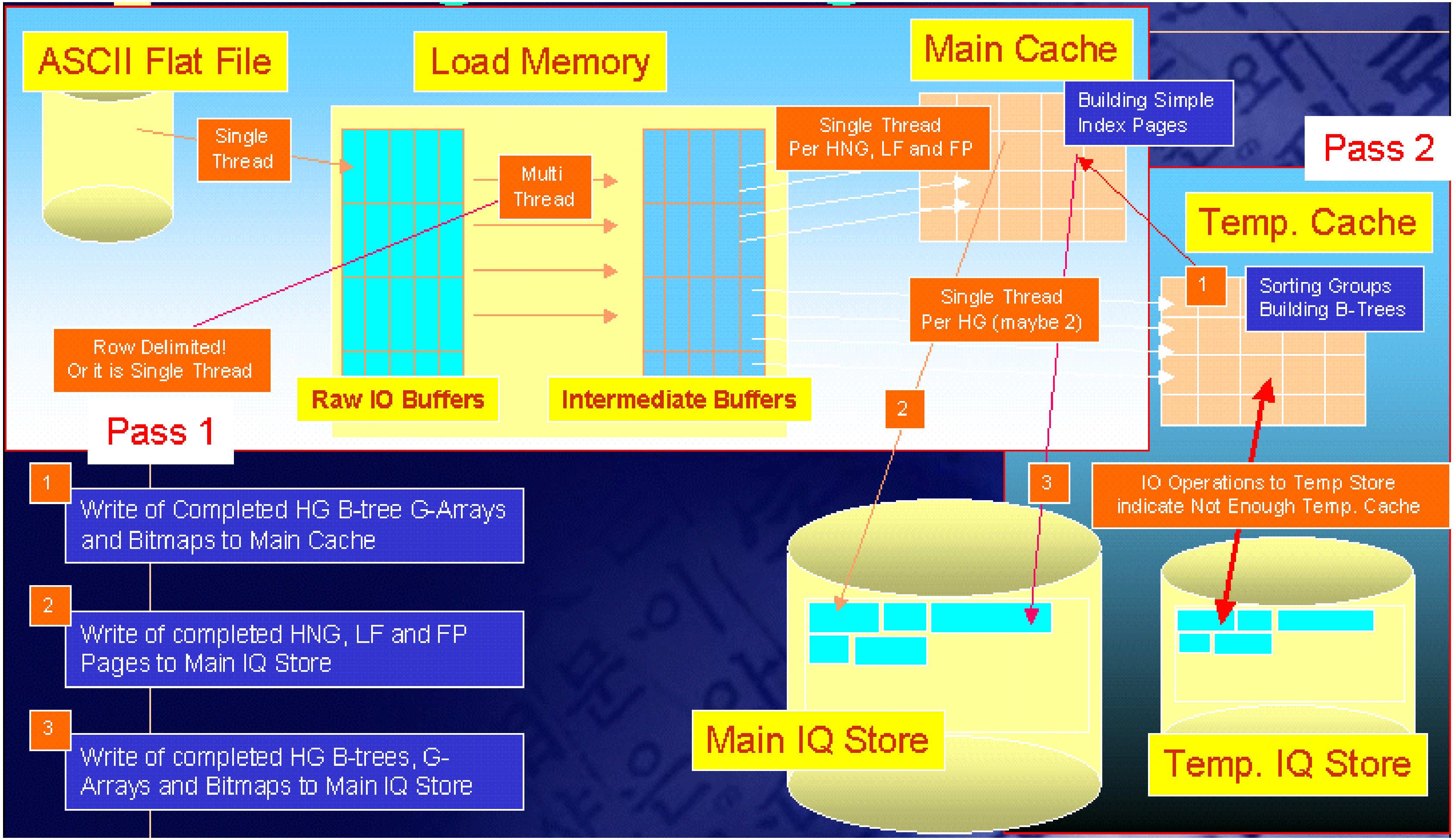
load-message file



```
In table 'TM_BASE_SALES_FACT', the partial width insert of 18 columns will begin at record 1.
2003-10-07 15:36:40 0000000006 Insert Started:
2003-10-07 15:36:40 0000000006 SALES_FACT
2003-10-07 16:32:45 0000000006 [20918]: Portions of the insert/load will be single threaded.
2003-10-07 15:36:42 0000000006 [20897]:         100000 Rows, 2 Seconds
2003-10-07 15:36:44 0000000006 [20897]:         200000 Rows, 2 Seconds
2003-10-07 15:36:46 0000000006 [20897]:         300000 Rows, 2 Seconds
2003-10-07 15:36:48 0000000006 [20897]:         400000 Rows, 2 Seconds
2003-10-07 15:36:50 0000000006 [20897]:         500000 Rows, 2 Seconds
2003-10-07 15:36:52 0000000006 [20897]:         600000 Rows, 2 Seconds
2003-10-07 15:36:54 0000000006 [20897]:         700000 Rows, 2 Seconds
2003-10-07 15:36:56 0000000006 [20897]:         800000 Rows, 2 Seconds
2003-10-07 15:36:58 0000000006 [20897]:         900000 Rows, 2 Seconds
2003-10-07 15:37:00 0000000006 [20897]:         1000000 Rows, 2 Seconds
...........
...........
2003-10-07 15:38:06 0000000006 [20897]:         4200000 Rows, 2 Seconds
2003-10-07 15:38:08 0000000006 [20897]:         4300000 Rows, 2 Seconds
2003-10-07 15:38:10 0000000006 [20897]:         4400000 Rows, 2 Seconds
2003-10-07 15:38:12 0000000006 [20897]:         4500000 Rows, 2 Seconds
2003-10-07 15:38:14 0000000006 [20897]:         4600000 Rows, 2 Seconds
2003-10-07 15:38:16 0000000006 [20897]:         4700000 Rows, 2 Seconds
2003-10-07 15:38:19 0000000006 [20897]:         4800000 Rows, 3 Seconds
2003-10-07 15:38:21 0000000006 [20897]:         4900000 Rows, 2 Seconds
2003-10-07 15:38:22 0000000006 [20897]:         5000000 Rows, 1 Seconds
2003-10-07 15:38:23 0000000006 [20895]: Insert Pass 1 completed in 103 seconds.
2003-10-07 15:38:23 0000000006 [20895]: Insert Pass 2 completed in 0 seconds.
2003-10-07 15:38:23 0000000006 [20834]:
        5000000 records were inserted into 'SALES_FACT'.

2003-10-07 15:38:23 0000000006 [20896]: Insert for 'SALES_FACT' completed in 103 seconds.  5000000 rows inserted.

~
"../MSG/so" 32 행, 1881 문자
```

18 개 컬럼의 SALES_FACT 라는 테이블에 partial width 적재가 시작되며 single thread mode로 동작하고 있습니다. (참고 105Byte)

load-big picture


ASCII Flat File
Load Memory
Main Cache
Building Simple Index Pages
Pass 2
Single Thread
Single Thread Per HNG, LF and FP
Multi Thread
Temp. Cache
1
Sorting Groups Building B-Trees
Single Thread Per HG (maybe 2)
Row Delimited! Or it is Single Thread
Raw IO Buffers
Intermediate Buffers
2
Pass 1
1
Write of Completed HG B-tree G-Arrays and Bitmaps to Main Cache
3
IO Operations to Temp Store indicate Not Enough Temp. Cache
2
Write of completed HNG, LF and FP Pages to Main IQ Store
3
Write of completed HG B-trees, G-Arrays and Bitmaps to Main IQ Store
Main IQ Store
Temp. IQ Store

SYBASE

extract

- **sam** 파일 데이터를 테이블로 적재하는 방법의 반대로 테이블의 데이터를 **sam** 파일 형태로 저장하는 방식을 말하며 **UNLOAD, EXPORT**라고도 합니다**.** 이렇게 저장된 데이터는 또 다른 시스템으로의 이동 및 백업 용도로 사용 가능 합니다**.**

- **SYBASE IQ**의 **EXTRACT** 방법은 무척 간단하며 많은 확장성을 가지고 있습니다**.** 먼저 여러 가지 옵션을 선정한 후에 **SELECT**문을 실행합니다**.** 그러면 화면에 출력되던 혹은 클라이언트에게 보내지던 **SELECT**문의 결과가 옵션에서 정한 형태와 이름을 가지고 **UNLOAD**됩니다**.** 단**,** 출력은 **SYBASE IQ**의 서버 모듈이 존재하는 곳에서만 가능합니다**.**

- **SELECT**문 뒤에 옵션을 취소 시키거나 임시로 옵션을 설정하지 않았다면 해당 사용자의 모든 질의는 선언된 파일로 출력되므로 특별한 목적이 없다면 **temporary option** 사용하기를 권장합니다**.**

- 출력되는 파일의 형태는 **ASCII, BINARY, BINARY/SWAP**로 옵션 설정에 따라 달라질 수 있습니다**.** 단**,** 사용자 정의 함수가 들어 있는 **SELECT**문은 사용할 수 없습니다**.**

- **syntax**

```
SET TEMPORARY OPTION Temp_Extract_Name1 = '…./…./….';     // EXTRACT 설정
SET TEMPORARY OPTION . . . . . . . ;
SET TEMPORARY OPTION . . . . . . . ;
select-statement;
SET TEMPORARY OPTION Temp_Extract_Name1 = '';              // EXTRACT 해제
```

extract-option

- **Temp_Extract_NameN**
  UNLOAD 하는 파일에 대한 이름과 경로를 지정합니다. 여기서 N은 1,2,3,4,5,6,7,8이며 Temp_Extract_SizeN 옵션과 같이 사용하여 한 파일시스템에 수용하기 어려운 결과도 여러 개의 파일로 나눠서 저장할 수 있습니다.이렇게 여러 개의 파일로 나눠서 저장할 때에도 파일별로 병렬로 처리되는 것이 아니라 직렬로 진행됨을 상기하며 만약 병렬로 데이터를 UNLOAD 하고 싶다면 WHERE 절의 조건을 바꾼 질의를 여러 개 실행해주면 됩니다. 또한 Temp_Extract_Name1이 설정되면 대상 테이블의 LOAD, DELETE, INSERT 명령어를 실행할 수 없습니다. 이럴 때 설정된 파일이름을 해제 시킬 때는 이 옵션의 값을 ‘’(empty string) 로 바꾸면 됩니다. 여기서 파일은 일반 유닉스 일반 파일뿐만 아니라 FIFO 파일도 가능합니다.

- **Temp_Extract_SizeN**
  UNLOAD 하는 대상 파일의 최대 크기를 KB 단위로 지정하며 그 크기는 AIX & HP-UX인 경우 64GB, Sun Solaris인 경우 512GB, Window인 경우 128GB 까지 입니다.

- **Temp_Extract_Column_Delimiter**
  출력되는 각 컬럼에 대한 구분자를 지정하며 디폴트는 콤마(,) 입니다

- **Temp_Extract_ Row_Delimiter**
  출력되는 각 레코드에 대한 구분자를 지정하며 디폴트는 carriage return(\n) 입니다

extract-option

- **Temp_Extract_ Binary**
  설정 값이 **on**일 경우 출력되는 파일의 형태를 바이너리 형태로 설정합니다.

- **Temp_Extract_ Swap**
  설정 값이 **on**일 경우 출력되는 파일의 형태를 바이너리 **swap** 형태로 설정합니다. 단, 선행되어야 할 조건으로 **Temp_Extract_Binary**이 **on** 으로 설정되어야 합니다.

- **Temp_Extract_ Null_As_Zero**
  ASCII 형태로 **UNLOAD**할 때 **NULL** 값에 대한 표현을 어떻게 처리할 지를 결정하며 **on**으로 솔정되면 **NULL** 값에 대한 데이터를 숫자 타입은 **0**으로 문자 타입이나 날짜 계열의 타입은 **empty string ('')**으로 변환하며 **off**일 경우는 데이터 타입에 관계없이 **string 'NULL'**로 출력합니다. 기본값은 **off** 이며 만약 기본 값으로 설정하고 **UNLOAD** 한 파일을 다시 **LOAD**할 때 주의 하시기 바랍니다.

- **Temp_Extract_Append**
  설정 값이 **on**일 경우 이미 존재하는 파일의 뒤에 첨삭하여 **UNLOAD**가 되며 만약 파일이 존재하지 않는다면 새로운 파일이 만들어 집니다.

- **Temp_Extract_Quote**
  Temp_Extract_Quotes, Temp_Extract_Quotes_All과 함께 사용되며 이 두가지 옵션중에 하나라도 **on**으로 설정되면 컬럼 데이터의 양쪽을 여기서 설정한 기호로 감싸줍니다. 단, **UNLOAD** 파일 형태가 **ASCII** 일 경우만 유효합니다.

extract-option

- **Temp_Extract_Quotes**
  이 옵션이 on으로 설정된다면 UNLOAD 할 때 테이블의 컬럼중 문자나 날짜 형태의 데이터를 각 필드의 시작과 끝 양쪽에 Temp_Extract_Quote에서 설정한 기호로 감싸줍니다. Temp_Extract_Quote 옵션을 따로 설정하지 않았다면 ' 가 사용됩니다.

- **Temp_Extract_Quote_All**
  이 옵션이 on으로 설정된다면 UNLOAD 할 때 테이블의 컬럼중 문자나 날짜 형태의 데이터뿐만 아니라 모든 데이터를 각 필드의 시작과 끝 양쪽에 Temp_Extract_Quote에서 설정한 기호로 감싸줍니다. Temp_Extract_Quote 옵션을 따로 설정하지 않았다면 ' 가 사용됩니다.

extract-example

- 예)

```
SET TEMPORARY OPTION Temp_Extract_Name1 = ‘/data/orders1.dat’;
SET TEMPORARY OPTION Temp_Extract_Name2 = ‘/data/orders2.dat’;
SET TEMPORARY OPTION Temp_Extract_Name3 = ‘’; // 중요
SET TEMPORARY OPTION Temp_Extract_Size1 = ‘1024000’;
SET TEMPORARY OPTION Temp_Extract_Size2 = ‘1024000’;
SET TEMPORARY OPTION Temp_Extract_Delimiter = ‘|’ ;
SET TEMPORARY OPTION Temp_Extract_Row_Delimiter = ‘\n’ ;
SET TEMPORARY OPTION Temp_Extract_Null_As_Zero = ‘on’ ;
SELECT * FROM orders ;
SET TEMPORARY OPTION Temp_Extract_Name1 = ‘’; // extract disable
```

- **orders** 테이블을 **1,024,000KB** 크기의 두개의 파일에 나누어서 **UNLOAD** 하는데 파일은 **|** 로 필드 분리 되었으며 레코드 분리자는 **\n** 입니다. 또한 출력 데이터중 숫자 컬럼의 **NULL** 값은 **0**으로, 문자/날짜 컬럼의 **NULL** 값은 **‘’(empty string)** 으로 표현됩니다.

- **만약 TEMPORARY 옵션이 생략되었다면 어떤 일이 일어날까요?**

Transaction, Versioning and Etc.

transaction

- 단일 사건으로 처리되어야 하며 영구적으로 발생되어진 변경사항을 데이터베이스에게 반영시켜줍니다. 오직 **COMMIT** 되거나 **ROLLBACK** 될 수 있습니다.

- **autocommit mode (unchained mode)**
  - 개별적인 **INSERT, UPDATE, DELETE, SELECT**를 개별 트랜잭션으로 처리합니다.
  - 만약 여러 개의 **statement**를 한 트랜잭션으로 묶으려면 명시적으로 **BEGIN TRAN**과 **COMMIT(**혹은 **ROLLBACK)**으로 트랜잭션의 처음과 끝을 지정해 주어야 합니다.
  - **ASE**와 **isql**의 기본적인 트랜잭션 모드 입니다.

- **manual commit mode (chained mode)**
  - 서버와 연결 혹은 또 다른 트랜잭션이 종료됨과 동시에 **BEGIN TRAN**이 암시적으로 수행됩니다.
  - 사용자가 명시적으로 트랜잭션의 **COMMIT**을 통해 종료를 나타내주어야 합니다.
  - **ANSI** 표준**, SYBASE IQ, ASA, dbisql**의 디폴트 트랜잭션 모드 입니다.

- **Chained** 라는 옵션을 통해 트랜잭션 모드를 변경할 수 있습니다

```
SET [TEMPORARY] OPTION Chained = ‘off’; // unchained mode
SET [TEMPORARY] OPTION Chained = ‘on’; // chained mode
```

transaction command

- **BEGIN TRANSACTION : unchained mode**에서 사용자가 임의로 정의하는 트랜잭션의 시작을 표시합니다.
  BEGIN TRAN[SACTION] [*transaction-name*];

- **COMMIT :** 트랜잭션의 종료를 표시하며 데이터베이스의 변경사항을 취소 없이 영원히 반영합니다
  COMMIT [WORK];

- **ROLLBACK :** 트랜잭션의 종료를 표시하며 이전의 발생된 변경사항을 취소합니다
  ROLLBACK [WORK];

- **SAVEPOINT :** 현재의 트랜잭션에 **breakpoint**를 만들어 줍니다
  SAVEPOINT [*savepoint-name*];

- **ROLLBACK TO SAVEPOINT : SAVEPOINT** 이후의 변경사항만 취소합니다.
  ROLLBACK TO SAVEPOINT [*savepoint-name*];

- **CHECKPOINT :** 메모리내의 변경사항을 디스크에 반영합니다.
  CHECKPOINT;

연관된 option

- **Auto_Commit : Interactive SQL 혹은 dbisql과 같은 SYBASE IQ의 기본 클라이언트에서만 동작하는 옵션으로 모든 명령 실행 후에 묵시적으로 COMMIT의 실행여부를 조절합니다. 기본값은 off로 자동으로 COMMIT을 하지 않습니다. 단, on으로 설정할 때 compound statement, batch 뒤에서 COMMIT 실행합니다.**
  SET [TEMPORARY] OPTION Auto_Commit = 'off';

- **Chained : chained 모드와 unchained 모드의 변경을 조절하며 기본 값은 on, 즉chained 모드입니다. 각 클라이언트 툴에서의 지원여부를 점검하십시오.**
  SET [TEMPORARY] OPTION Chained = 'on';

- **Commit_On_Exit : Interactive SQL, dbisql과 같은 SYBASE IQ의 기본 클라이언트 에서만 동작하는 옵션으로 툴 종료시 COMMIT의 실행 여부를 조절합니다. 기본 값은 on, 즉 툴 종료시 자동으로 COMMIT을 실행합니다. 각 클라이언트 툴에서의 지원 여부를 점검하십시오.**
  SET [TEMPORARY] OPTION Commit_on_exit = 'on';

- **AutoPreCommit : 이 옵션은 다른 옵션과 달리 odbc 환경 파일에 저장하는 connection level 옵션이며 모든 명령어전에 COMMIT의 실행여부를 조절합니다. 기본 값은 off 즉 자동으로 COMMIT하지 않으며 Interactive SQL, dbisql과 같은 SYBASE IQ의 기본 클라이언트에서는 비록 odbc 환경하에서 동작하지만 작동하지 않습니다.**

transaction의 이해

Auto_Commit	Chained	내용	Rollback	비고

on, off	on	BEGIN
		update test set a = 1;
		rollback;
		END; ✓
			O	compound statement

on, off	off	BEGIN
		update test set a = 1;
		rollback;
		END;
			X	

on	on	update test set a = 1; ✓ rollback;	X	1 statement per 1 batch
off	on	update test set a = 1; rollback;	O	
on, off	off	update test set a = 1; rollback;	X	
on, off	on	update test set a = 1 rollback; ✓	O	2 statement per 1 batch
on, off	off	update test set a = 1 rollback;	X	

✓ Auto_Commit이 실행되는 시점

snapshot versioning

- **SYBASE IQ**의 테이블에 대한 변경을 관리하기 위한 방법으로 **table-level versioning**이라고도 합니다.

- **ANSI isolation level 3**을 지원하며 읽고 있는 동안에도 쓸 수 있는 기능을 제공합니다. 특정한 시점에 하나의 테이블에 대해 쓸 수 있는 사용자는 **SYBASE IQ**에 의해 자동으로 **1**명으로 제한되며 조절 불가능합니다.

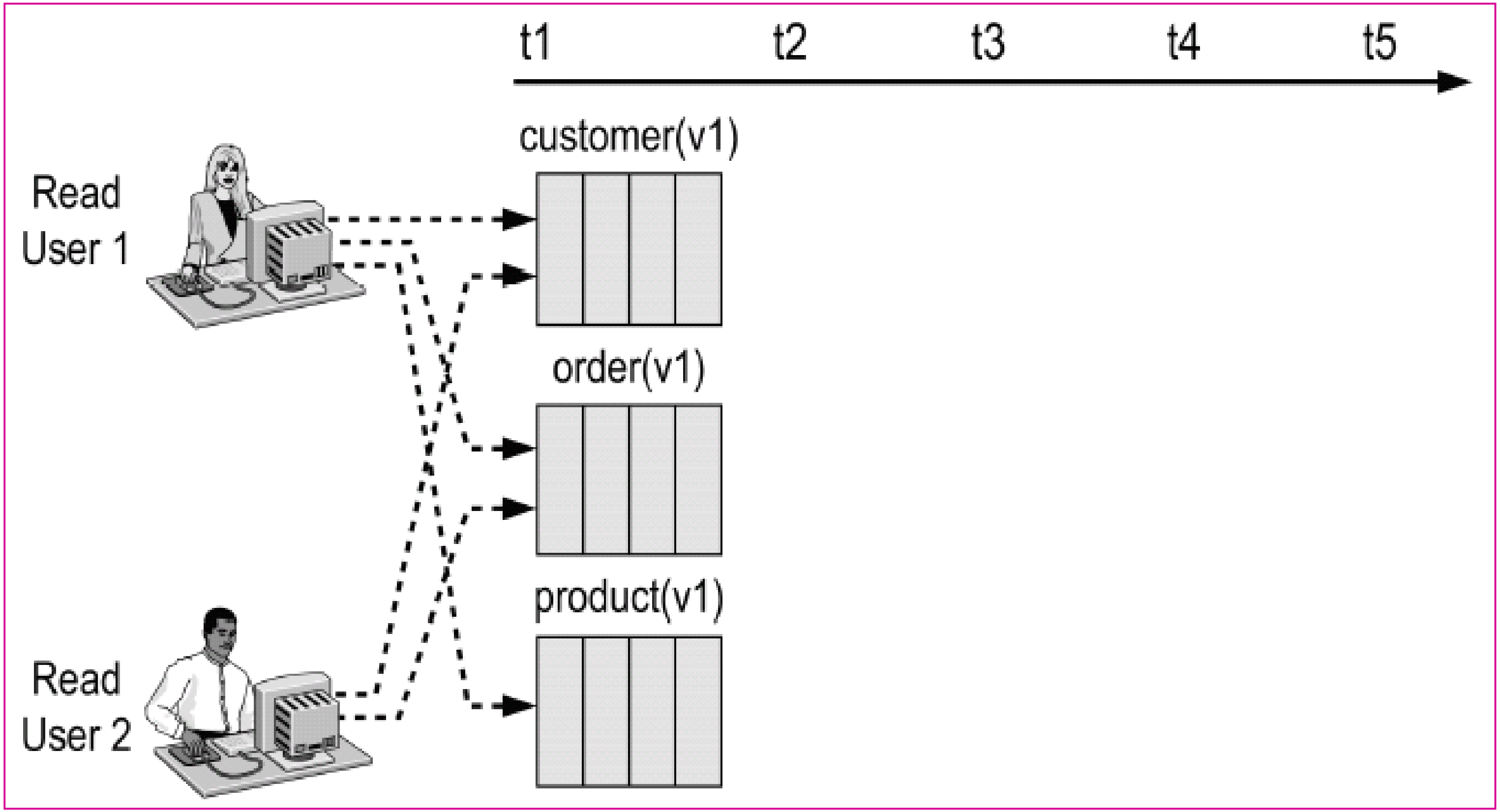
- 어떤 테이블이 수정되어지면 그 페이지들의 이전 데이터들은 그대로 남겨두고 새로운 페이지에 새로운 데이터들이 쓰여지며 각 페이지 헤더에는 **blockmap** 이라는 버전정보를 보관하는 데이터 구조가 있습니다. 그래서 이전 페이지의 **blockmap**에는 이전버전이 새로운 페이지의 **blockmap**에는 새로운 버전이 저장됩니다.

- 이전 버전들은 더 이상 이전 데이터를 보는 사용자가 존재하지 않으면 삭제되며 이런 버전관리가 다른 **RDBMS**에서의 **log**와 비슷합니다.

- 사용자가 현재 버전의 테이블 내용을 보고 싶으면 먼저 **COMMIT**을 실행시키고 질의를 하시면 됩니다. 즉 **SYBASE IQ**의 **COMMIT**에는 두 가지의 의미가 있습니다.

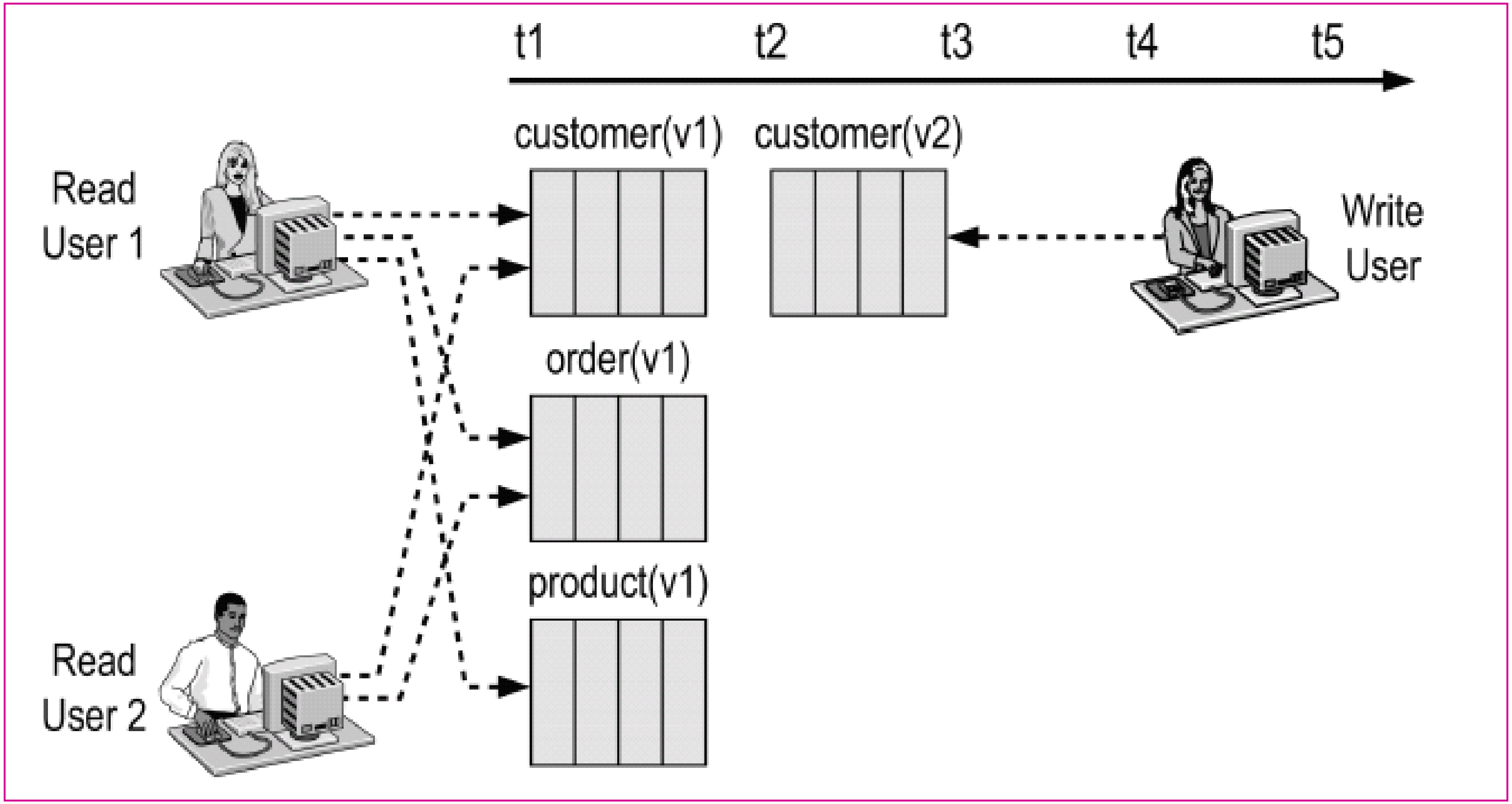
versioning example

- **t1 시점에 User1, User2는 customer,order, product 테이블로 부터 데이터를 조회합니다.**



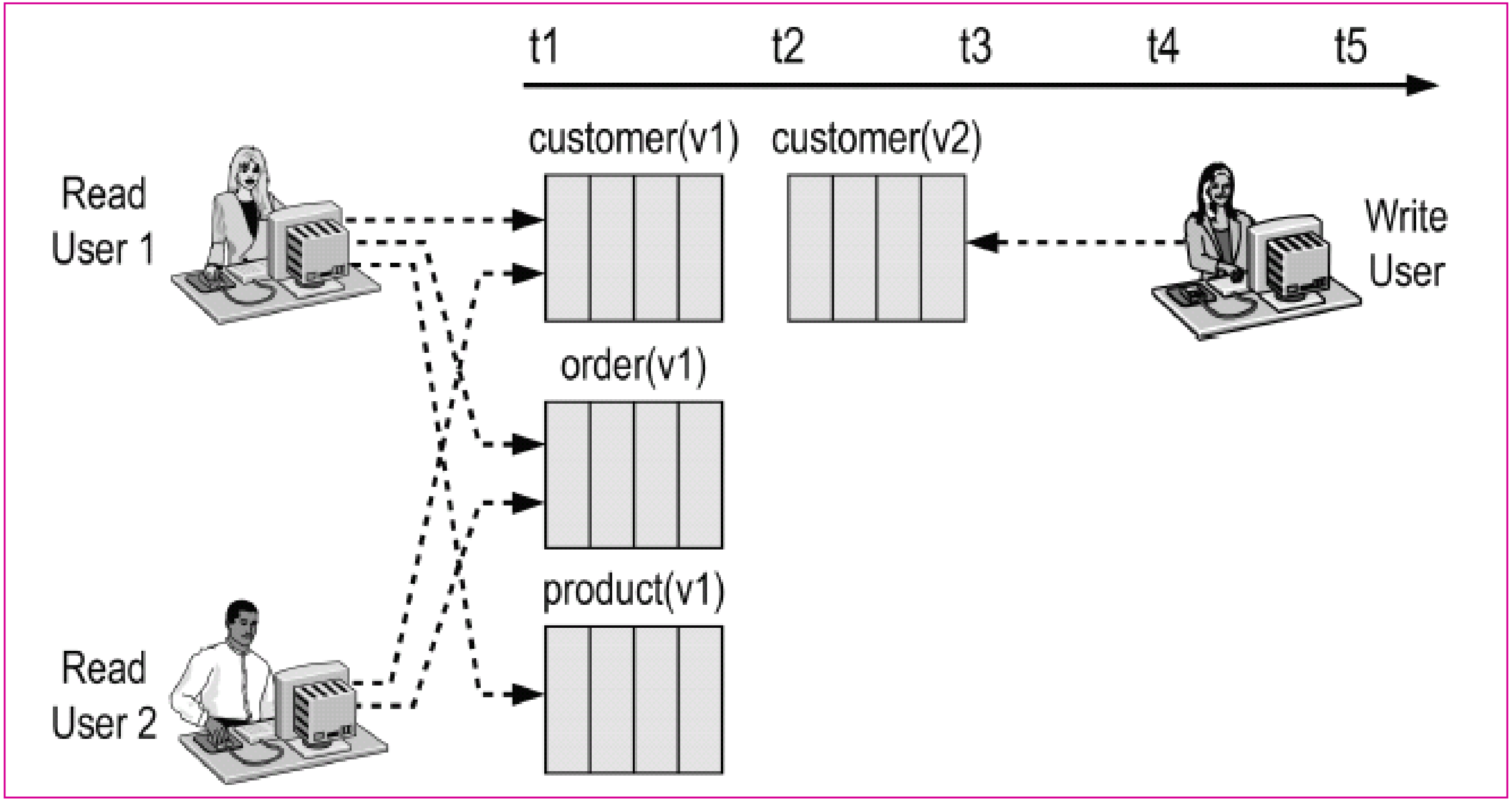
versioning example

- **t2 시점에 write user가 들어와서 customer 테이블을 수정하고 있으며 같은 시간에 다른 사용자들은 아직도 여전히 customer,order,product 테이블을 조회하고 있습니다.**



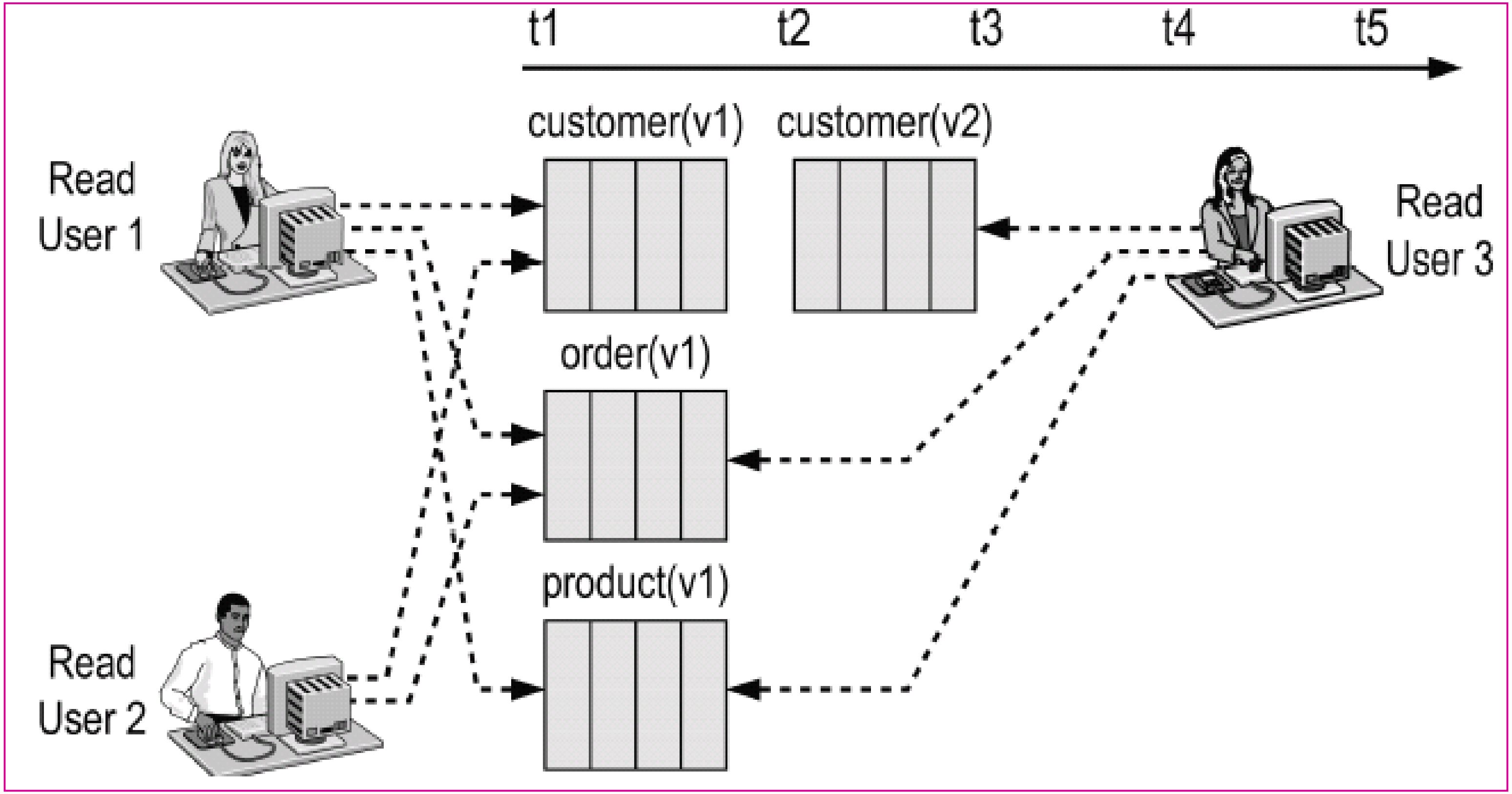
versioning example

- **t3 시점에 write user의 customer 테이블에 대한 수정이 COMMIT 되었습니다. 같은 시간에 다른 사용자들은 아직도 여전히 customer,order,product 테이블의 원래 버전을 조회하고 있습니다.**



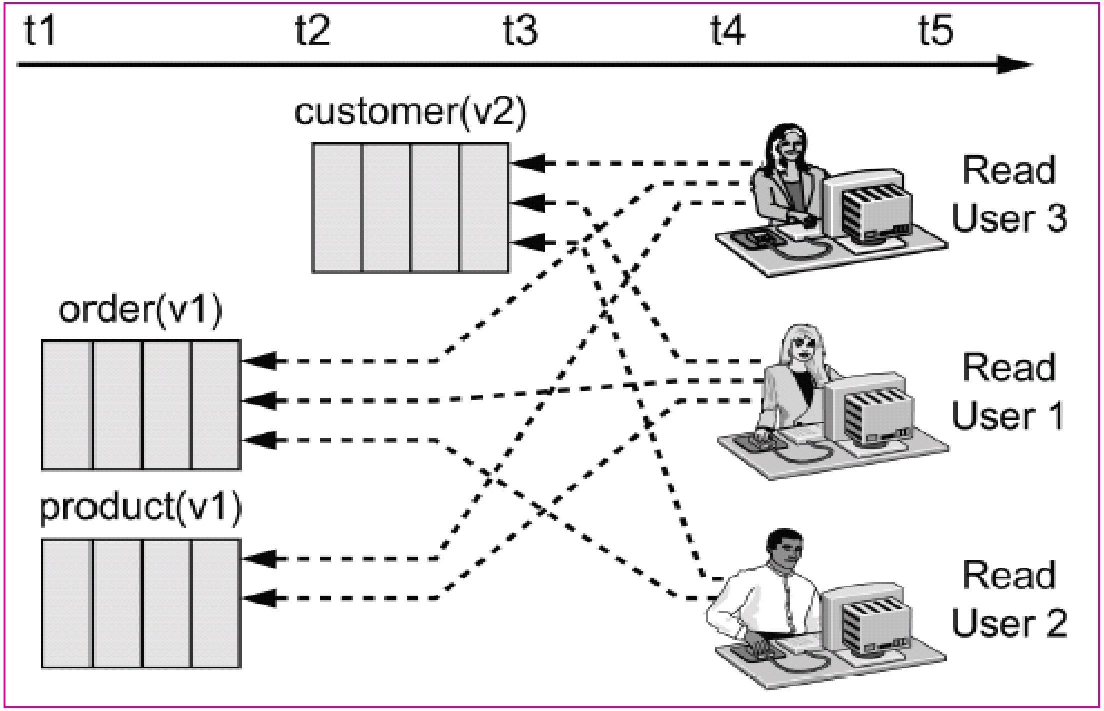
versioning example

- **t4** 시점에 **User 3**이 **login** 해서 **customer** 테이블의 새 버전과 **order, product**의 원래 버전을 조회하고 있습니다. 같은 시간에 다른 사용자들은 아직도 여전히 **customer, order, product** 테이블의 원래 버전을 조회하고 있습니다.

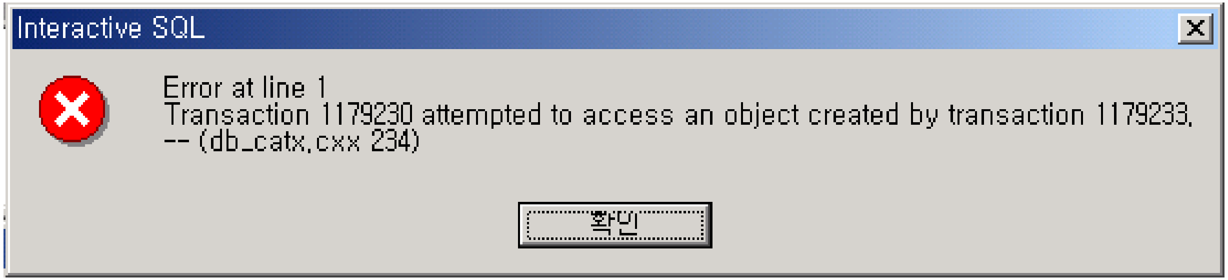


versioning example

- **t5 시점에 User 1, User 2가 COMMIT을 수행하거나 아니면 재접속 하여 새로운 버전의 customer 테이블과 원래버전의 order, product 테이블을 읽고 있습니다. 이 시점에 customer의 원래버전은 삭제됩니다.**



개발시 자주 발생하는 현상



Interactive SQL

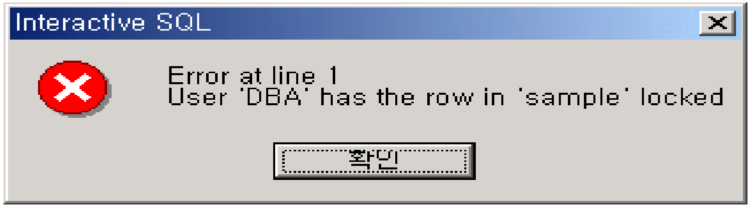
Error at line 1
Transaction 1179230 attempted to access an object created by transaction 1179233.
-- (db_catx.cxx 234)

- 재연
  - **Interactive SQL**를 이용하여 **A,B** 두 개의 세션을 동시에 연결합니다
  - **A** 세션에서 기존 테이블 **sample**에 컬럼을 하나 더 추가합니다
  - **B** 세션에서 **A**세션에서 만든 **sample** 테이블을 조회합니다
  - 두 세션 모두 **Auto_Commit** 옵션이 **off**로 설정되었다고 가정합니다**.**

- 해결방법

  이 문제는 **SYBASE IQ**의 **versioning**과 관련한 문제로 **B** 세션의 연결시점에 혹은 마지막 **COMMIT**를 실행한 후에 **sample**이라는 테이블은 없었습니다**.** 이 시점에 **B** 세션에서 **COMMIT**을 실행하면 **B** 세션에서도 **sample** 이라는 테이블을 액세스 할 수 있습니다**.**

개발시 자주 발생하는 현상



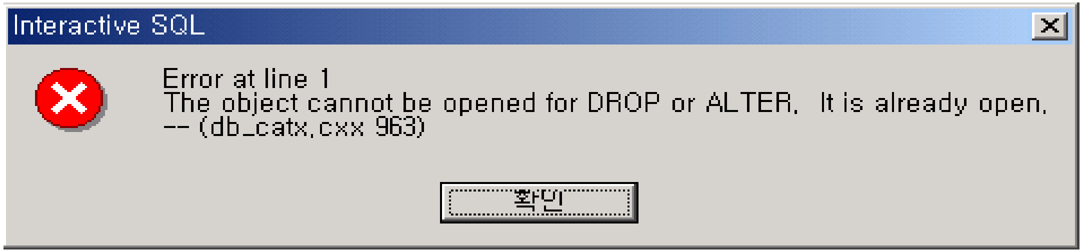
Error at line 1
User 'DBA' has the row in 'sample' locked

- 재연
  - Interactive SQL를 이용하여 A,B 두 개의 세션을 동시에 연결합니다
  - A 세션에서 sample 테이블을 SELECT 하고 그냥 그 상태를 유지합니다
  - B 세션에서 sample 테이블을 DROP 하거나 ALTER 합니다.
  - 두 세션 모두 Auto_Commit 옵션이 off로 설정되었다고 가정합니다.

- 해결 방법

  이것은 SYBASE IQ의 locking 메카니즘과 때문에 발생합니다. A 세션에서 sample 테이블을 SELECT 하고 있는 상태에서 SYBASE IQ는 sample 테이블에 lock을 하고 있습니다. 이것을 해결하려면 트랜잭션을 종료해야 합니다. 즉 트랜잭션을 종료하기 위해 COMMIT을 실행시켜야 합니다.COMMIT을 실행하였더라도 현재 SELECT 결과의 커서가 열려 있기 때문에 RESUME 이란 명령어를 통해 이 또한 닫아 주어야 합니다.

개발시 자주 발생하는 현상



Error at line 1
The object cannot be opened for DROP or ALTER. It is already open.
-- (db_catx.cxx 963)

- 재연
  - ● **Interactive SQL**를 이용하여 **A,B** 두 개의 세션을 동시에 연결합니다
  - ● **A** 세션에서 **sample** 테이블을 **SELECT**하고 **COMMIT**를 실행합니다**.**
  - ● **B** 세션에서 **sample** 테이블을 **DROP** 하거나 **ALTER** 합니다
  - ● 두 세션 모두 **Auto_Commit** 옵션이 **off**로 설정되었다고 가정합니다**.**

- 해결 방법

  이것은 **SYBASE IQ** 클라이언트 툴의 커서 방식 프로그램 기법 때문에 발생하는 현상으로 **A**세션에서 **SELECT**한 후에 **COMMIT**을 실행하면 잠긴 **sample** 테이블에 **lock** 현상은 종료가 되나 **sample** 테이블에 대한 **point**를 계속해서 유지하기 위해 커서가 열려있습니다**.** 이 커서를 종료하기 위해서는 **A** 세션에서 **RESUME** 명령어를 사용하거나 아니면 다른 **SELECT**를 사용하면 됩니다**.**혹은 **A** 세션을 종료한 후 다시 연결하면 됩니다**.**

Transaction Tip

- **READ 사용자와 WRITE 사용자를 분리하여 생성하고 WRITE 작업은 가능하면 READ 사용자가 없는 시간에 수행합니다.**

- **READ 사용자는 READ 명령 전후에 COMMIT 을 실행합니다.**
  - **dbisqlc을 사용하는 READ 사용자는 Auto_Commit='On' 으로 변경합니다**
  - **OLAP Tool을 포함한 ODBC 환경에서는 AutoPreCommit='Y'로 변경하고 각 질의 뒤에 OLAP Tool 의 후처리 기능을 이용하여 COMMIT을 실행합니다 (단, dbisqlc 는 위 조건에서 제외됩니다)**
  - **그 외 isql과 같은 OCDK 환경의 READ 사용자는 디폴트 상태로 사용합니다.**

- **위와 같이 사용하면 versioning과 transaction으로 부터 발생하는 개발 부작용을 최소화 시킬 수 있습니다**

유용한 system SP

- sp_iqconnection
- sp_iqstatus
- sp_iqtransaction
- sp_iqview
- sp_iqspaceused
- sp_iqlocks
- sp_iqcontext
- sp_iqtable
- sp_iqtablesize
- sp_iqcolumn
- sp_iqindex
- sp_iqindexsize
- sp_helptext

BI. Anytime. Anywhere

SYBASE®

www.sybase.co.kr

02-3451-5200