Tibero RDBMS

애플리케이션 개발자 안내서

Tibero RDBMS 4 SP1

안내서 정보

안내서 제목: Tibero RDBMS 애플리케이션 개발자 안내서

발행일: 2013-02-25

소프트웨어 버전: Tibero RDBMS 4 SP1

안내서 버전: 2.1.4

내용 목차

그림 목차

예 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero RDBMS®(이하 Tibero RDBMS)에서 제공하는 각종 애플리케이션 라이브러리를 이용하여 프로그램을 개발하려는 애플리케이션 프로그램 개발자를 대상으로 기술한다.

안내서의 전제 조건

본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- Java 프로그래밍의 이해

안내서의 제한 조건

본 안내서는 Tibero RDBMS를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않다. 따라서 설치, 환경설정 등 운용 및 관리에 대해서는 각 제품 안내서를 참고하기 바란다.

참고

Tibero RDBMS의 설치 및 환경 설정에 관한 내용은 "Tibero RDBMS 설치 안내서"를 참고한다.

안내서 구성

Tibero RDBMS 애플리케이션 개발자 안내서는 총 4개의 장과 Appendix로 이루어져 있다.

각 장의 주요 내용은 다음과 같다.

- 제1장: 데이터 타입의 사용

  Tibero RDBMS에서 제공하는 데이터 타입과 이를 사용하는 방법을 소개한다.

- 제2장: tbJDBC의 사용

  tbJDBC를 사용하는 방법을 기술한다.

- 제3장: 트리거의 사용

  트리거를 사용하는 방법을 기술한다.

- 제4장: XA의 사용

  분산 트랜잭션 처리에 필요한 XA를 사용하는 방법을 기술한다.

- Appendix A.: tbJDBC 예제 프로그램 소스 코드

  tbJDBC를 이용하여 작성한 기본 프로그램의 전체 소스 코드를 기술한다.

- Appendix B.: Tibero와 Tuxedo 연동 예제 프로그램 소스 코드

  Tibero와 Tuxedo 연동 예제 프로그램의 전체 소스 코드와 각종 스크립트를 기술한다.

안내서 규약

표기	의미
<AaBbCc123>	프로그램 소스 코드의 파일명, 디렉터리
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
[그림 1.1]	그림 이름
[표 1.1]	표 이름
`AaBbCc123`	명령어, 명령어 수행 후 화면에 출력된 결과물, 예제코드
{ }	필수 인수 값
[ ]	옵션 인수 값
|	선택 인수 값

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i (PA-RISC, ia64)
	Solaris (SPARC 9/Solaris 9)
	AIX (PPC 5L/AIX 5.3)
	GNU (X86, 64, IA64)
	Linux kernel 2.6 이상
Hardware	최소 1.5GB 하드디스크 공간
	512MB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero RDBMS 설치 안내서	설치 시 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero RDBMS tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero RDBMS External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero RDBMS JDBC 개발자 안내서	Tibero RDBMS에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero RDBMS tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero RDBMS tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero RDBMS tbPSM 안내서	저장 프로시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, tbPSM 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero RDBMS tbPSM 참조 안내서	저장 프로시저 모듈인 tbPSM의 패키지를 소개하고, 이러한 패키지에 포함된 각 프로시저와 함수의 프로토타입, 파라미터, 예제 등을 기술한 참조 안내서이다.
Tibero RDBMS 관리자 안내서	Tibero RDBMS의 동작과 주요 기능의 원활한 수행을 보장하기 위해 DBA가 알아야 할 관리 방법을 논리적 또는 물리적 측면에서 설명하고, 관리를 지원하는 각종 도구를 기술한 안내서이다.
Tibero RDBMS tbAdmin 안내서	SQL/PSM 처리와 DBA를 위한 시스템 관리 기능을 제공하는 GUI 기반의 툴인 tbAdmin을 소개하고, 설치 및 사용 방법을 기술한 안내서이다.
Tibero RDBMS 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero RDBMS	Tibero RDBMS를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.

안내서	설명
에러 참조 안내서	
Tibero RDBMS 참조 안내서	Tibero RDBMS의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.
Tibero RDBMS SQL 참조 안내서	데이터베이스 작업을 수행하거나 애플리케이션 프로그램을 작성할 때 필요한 SQL 문장을 기술한 참조 안내서이다.

연락처

Korea

TIBERO Co., Ltd
272-6 Tmax Building 3th floor, Seohyeon-dong, Bundang-gu,
Seongnam-si, Gyeonggi-do, 463-824
South Korea
Tel: +82-31-779-7113
Fax: +82-31-779-7119
Email: `tibero@tibero.com`
Web (Korean): http://www.tibero.com
기술지원: http://technet.tmaxsoft.com

USA

TmaxSoft, Inc.
560 Sylvan Avenue Englewood Cliffs, NJ 07632
U.S.A
Tel: +1-201-567-8266
Fax: +1-201-567-7339
Email: `info@tmaxsoft.com`
Web (English): http://www.tmaxsoft.com

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: `info@tmaxsoft.co.jp`
Web (Japanese): http://www.tmaxsoft.co.jp

China

TmaxSoft China Co., Ltd.
Beijing Silver Tower, RM 1508, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
China
Tel: +86-10-6410-6145~8
Fax: +86-10-6410-6144
Email: info.cn@tmaxsoft.com
Web (Chinese): http://www.tmaxsoft.com.cn

제1장 데이터 타입의 사용

본 장에서는 각종 애플리케이션 라이브러리를 사용하기 앞서 Tibero RDBMS가 기본적으로 제공하는 데이터 타입을 소개하고 사용하는 방법을 설명한다.

1.1. 개요

데이터 타입은 데이터베이스에 스키마 객체를 생성할 때 필요하며, 일반적인 클라이언트 프로그램에서도 모든 데이터 타입에 대응되는 변수를 사용할 수 있다.

Tibero RDBMS에서 제공하는 데이터 타입은 다음과 같다.

구분	데이터 타입	설명
문자형	CHAR, VARCHAR, VARCHAR2, CLOB, LONG	문자열을 표현하는 데이터 타입이다.
숫자형	NUMBER	정수나 실수의 숫자를 저장하는 데이터 타입이다.
날짜형	DATE, TIMESTAMP, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND	시간이나 날짜를 저장하는 데이터 타입이다.
바이너리	RAW, BLOB, LONG RAW	대용량의 바이너리 데이터를 저장하는 데이터 타입이다.
내재형	ROWID	사용자가 명시적으로 선언하지 않아도 Tibero RDBMS가 자동으로 삽입되는 로우마다 포함하는 컬럼의 타입이다.

참고

데이터 타입에 대한 자세한 내용은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

1.2. 문자형

본 절에서는 각 데이터 타입을 사용하는 방법을 설명한다.

CHAR

CHAR 타입은 일반 문자열을 저장하는 데이터 타입으로, 고정 길이 문자열이다.

사용 방법은 다음과 같다.

```
CHAR[(size)]
```

옵션	설명
size	size는 선택적으로 사용할 수 있다. (최댓값: 2,000byte) - size를 입력한 경우: size만큼의 고정 길이를 갖는다. - size를 입력하지 않은 경우: 기본값인 1byte의 고정 길이를 갖는다.

VARCHAR

VARCHAR 타입은 일반 문자열을 저장하는 데이터 타입으로, 가변 길이 문자열이다.

사용 방법은 다음과 같다.

```
VARCHAR(size)
```

항목	설명
size	size에 정의된 만큼의 길이를 갖는다. (최댓값: 4,000byte)

VARCHAR2

VARCHAR2 타입은 일반 문자열을 저장하는 데이터 타입으로, 가변 길이 문자열이다. 테이블 생성시 VARCHAR 로 표시된다.

사용 방법은 다음과 같다.

```
VARCHAR2(size)
```

항목	설명
size	size에 정의된 만큼의 길이를 갖는다. (최댓값: 4,000byte)

CLOB

CLOB 타입은 읽을 수 있는 문자열을 데이터베이스에 저장하는 데이터 타입으로, 대용량 데이터를 저장한다. 사용 방법은 다음과 같다.

```
CLOB
```

대용량 데이터는 4GB까지 저장할 수 있다.

LONG

LONG 타입은 대용량 데이터를 저장하기 위한 데이터 타입이다. 단, Oracle과의 호환성을 위해서만 사용한다. 그 외에는 CLOB 타입을 사용할 것을 권장한다.

사용 방법은 다음과 같다.

```
LONG
```

1.3. 숫자형

NUMBER

NUMBER 타입은 정수 또는 실수를 저장하는 데이터 타입이다. 실제로 데이터베이스에 저장될 때는 숫자의 크기에 따라 가변 길이로 저장된다.

사용 방법은 다음과 같다.

```
NUMBER[(precision | precision , scale)]
```

옵션	설명
precision	정밀도이며, 정수의 자릿수를 설정한다.
scale	스케일이며, 소수점의 자릿수를 설정한다.

1.4. 날짜형

DATE

DATE 타입은 년, 월, 일의 날짜를 저장하는 데이터 타입이다. 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

사용 방법은 다음과 같다.

```
DATE
```

TIMESTAMP

TIMESTAMP 타입은 DATE 타입을 확장한 것으로 시간까지 저장하는 데이터 타입이다. 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

사용 방법은 다음과 같다.

```
TIMESTAMP[(frac_sec_prec)]
```

옵션	설명
frac_sec_prec	시간 값의 소수 초 정밀도(fractional second precision)를 설정한다. - 0부터 9까지 사용할 수 있다. (기본값: 6)

INTERVAL YEAR TO MONTH

INTERVAL YEAR TO MONTH 타입은 년과 월의 차이를 저장하는 데이터 타입이다. 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

사용 방법은 다음과 같다.

```
INTERVAL YEAR [(year_prec)] TO MONTH
```

옵션	설명
year_prec	year precision를 설정한다. - 0부터 9까지 사용할 수 있다. (기본값: 2) - 년의 자릿수를 제한할 수 있다.

INTERVAL DAY TO SECOND

INTERVAL DAY TO SECOND 타입은 날짜와 시간의 차이를 저장하는 데이터 타입이다. 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

사용 방법은 다음과 같다.

```
INTERVAL DAY [(day_prec)] TO SECOND [(frac_sec_prec)]
```

옵션	설명
day_prec	day precision를 설정한다. - 0부터 9까지 사용할 수 있다. (기본값: 2)
frac_sec_prec	TIMESTMAP 타입에 정의된 옵션과 같다.

이 타입은 day_prec 옵션과 frac_sec_prec 옵션을 사용하여 날짜의 자릿수, 초 이하의 자릿수를 제한할 수 있다.

1.5. 바이너리

RAW

RAW 타입은 가변 길이를 갖는 바이너리 데이터를 저장하는 데이터 타입이다.

사용 방법은 다음과 같다.

```
RAW(size)
```

항목	설명
size	size에 정의된 만큼의 길이를 갖는다. (최댓값: 2,000byte)

BLOB

BLOB 타입은 대용량의 바이너리 데이터를 저장하는 데이터 타입이다.

사용 방법은 다음과 같다.

```
BLOB
```

대용량의 바이너리 데이터는 4GB까지 저장할 수 있다.

LONG RAW

LONG RAW 타입은 대용량의 바이너리 데이터를 저장하는 데이터 타입이다. 단, Oracle과의 호환성을 위해서만 사용한다. 그 외에는 BLOB 타입을 사용할 것을 권장한다.

사용 방법은 다음과 같다.

```
LONG RAW
```

1.6. 내재형

ROWID

ROWID 타입은 데이터베이스 테이블의 컬럼 주소를 저장하는 데이터 타입이다. 실제로 데이터베이스에 저장될 때는 고정 길이로 저장된다.

사용 방법은 다음과 같다.

```
ROWID
```

제2장 tbJDBC의 사용

Tibero RDBMS에서는 Java 프로그램 안에서 SQL 문장을 실행하기 위해 데이터베이스를 연결해주는 애플리케이션 프로그램의 인터페이스를 제공한다. 이러한 인터페이스를 **tbJDBC**(Tibero의 Java database connectivity)라 한다.

tbJDBC는 JDK 버전이 1.3 이상의 환경에서 동작한다. 단, JDK 버전이 1.3인 경우에는 **tibero4-jdbc-13.jar** 파일의 형태로 제공되며, 그 외 나머지 버전은 **tibero4-jdbc.jar** 파일의 형태로 제공된다.

참고

본 안내서에서는 JDBC 프로그래밍에 대해 간략히 설명한다. tbJDBC에 대한 자세한 내용은 "Tibero RDBMS JDBC 개발자 안내서"를 참고한다.

2.1. JDK 설치

tbJDBC를 사용하기 위해서는 **JDK 1.3** 이상이 반드시 설치되어 있어야 한다.

다음 위치에서 JDK를 다운로드 할 수 있다.

```
http://java.sun.com/javase/downloads/index.jsp
```

만약 시스템이 Sun Microsystems사의 JDK를 사용하지 않는다면, 각각의 시스템에 적합한 JDK를 찾아 설치한다. 예를 들어 HP-UX는 HP, AIX는 IBM에서 JDK를 다운로드 받아 설치하면 된다.

각 시스템별 JDK를 설치하는 방법은 다음 위치에서 확인할 수 있다.

```
http://java.sun.com/j2se/1.5.0/install.html
```

참고

벤더별 JDK 설치 방법은 각 벤더에서 제공하는 설치 안내서를 참고한다.

2.2. JDBC의 표준 기능

tbJDBC는 JDBC 3.0 표준을 준수한다. 다만, 표준에 벤더별로 구현해도 된다고 명시한 추가 기능에 대해서는 지원하지 않을 수도 있다.

본 절에서는 tbJDBC에서 구현되는 표준 기능을 JDBC 버전 별로 간략히 설명한다. 보다 자세한 내용은 Sun Microsystems사에서 발행한 JDBC 3.0 Specification 문서를 참고한다.

2.2.1. JDBC 1.0 및 JDBC 2.0

본 절에서는 JDBC 1.0 및 JDBC 2.0의 표준 기능을 지원 여부에 따라 각각 설명한다.

지원하는 기능

- SQL-99 데이터 타입

  tbJDBC는 SQL-99에서 새롭게 추가된 데이터 타입 중에 BLOB와 CLOB만을 지원한다. 따라서 이러한 데이터 타입을 Tibero RDBMS에서 사용할 때는 다음과 같은 규칙을 따른다.

  – Blob, Clob 인터페이스는 해당 트랜잭션이 수행 중인 동안만 유효하다.

  – Blob, Clob 인터페이스는 위치 지시자(LOCATOR)를 통해 구현된다.

    위치 지시자는 데이터베이스 서버에 존재하는 데이터의 논리적인 지시자(POINTER)이다.

- 인터페이스 메소드

  지원하는 인터페이스 메소드(Interface Method)는 다음과 같다.

  – java.sql.Driver

  – java.sql.DatabaseMetaData

  – java.sql.ResultSetMetaData

  – java.sql.CallableStatement

  – java.sql.Connection

  – java.sql.PreparedStatement

  – java.sql.ResultSet

  – java.sql.Statement

  – java.sql.Blob

  – java.sql.Clob

  – javax.sql.ConnectionEventListener

  – javas.sql.ConnectionPoolDataSource

- javax.sql.DataSource
- javax.sql.PooledConnection
- javax.sql.XAConnection
- javax.sql.XADataSource

- 정적 초기화를 구현한 드라이버

  java.sql.Driver 인터페이스 메소드를 구현한 드라이버 클래스는 정적 초기화 블록(Static block)에서 Drivermanager.registerDriver 메소드를 호출함으로써, 새로 생성한 드라이버 인스턴스를 드라이버 관리자(Driver Manager)에 등록할 수 있다.

- SQL-92 Entry Level의 확장 기능

  DROP TABLE과 ESCAPE 문을 사용할 수 있다.

- 스칼라 함수(Scalar Functions)

  데이터베이스 서버에서 지원하는 모든 스칼라 함수를 사용할 수 있다. 스칼라 함수는 **DatabaseMetaData** 클래스를 통해 확인할 수 있다.

- 멀티스레딩(Multithreading)

  여러 개의 스레드가 java.sql과 javax.sql 패키지 내에 존재하는 객체에 모든 연산을 동시에 수행할 수 있다.

- 스크롤 가능한 결과 집합(Scrollable ResultSet)

  결과 집합을 전 방향 또는 후 방향으로 탐색할 수 있다. 또한, 결과 집합 내에서 상대적이거나 절대적인 위치 이동도 할 수 있다.

- 수정 가능한 결과 집합(Updatable ResultSet)

  결과 집합을 수정할 수 있다.

- 민감한 결과 집합(Sensitive ResultSet)

  결과 집합의 커서가 열린 후에 발생하는 데이터의 변화를 반영할 수 있다.

- 일괄 수정 작업(batch updates)

  여러 번의 수정 작업을 한꺼번에 처리할 수 있다.

지원하지 않는 기능

- SQL-99 데이터 타입

  tbJDBC는 SQL-99 데이터 타입 중에서 Array, Ref, Struct를 지원하지 않는다

- 인터페이스 메소드

  지원하지 않는 인터페이스 메소드는 다음과 같다.

  – java.sql.Array

  – java.sql.Ref

  – java.sql.SQLInput

  – java.sql.SQLOutput

  – java.sql.Struct

- 잘린(Truncated) 입력 파라미터의 값에 대한 예외 상황

  입력 파라미터의 값이 잘린 경우, JDBC 드라이버는 **Data Truncation**이라는 예외 상황(Exception)을 발생시킨다.

- 위치 설정을 통한 수정 및 삭제(Positioned Updates and Deletes)

  결과 집합의 커서에서 설정한 현재 위치의 레코드를 수정하거나 삭제하는 기능이다. 이 기능은 **수정 가능한 결과 집합**이라는 기능으로 대체가 가능하다.

2.2.2. JDBC 3.0

본 절에서는 JDBC 3.0의 표준 기능을 지원 여부에 따라 각각 설명한다.

지원하는 기능

- 인터페이스

  지원하는 인터페이스는 다음과 같다.

  – DatabaseMetaData

  – ResultSetMetaData

- 파라미터 메타데이터

ParameterMetaData 인터페이스를 구현한 JDBC 클래스는 준비된 문장(Prepared Statement)에서 사용한 파라미터 개수의 정보를 제공한다. 단 데이터 타입 및 속성(Property)에 대한 메타데이터(metadata)는 제공하지 않는다.

- 저장점

  Savepoint 인터페이스를 구현하여 트랜잭션에 대한 저장점 설정과 커밋 및 롤백 기능을 제공한다. 단, 특정한 저장점을 해제하는 **Connection.releaseSavepoint** java.sql 메소드는 제공하지 않는다.

지원하지 않는 기능

- 결과 집합의 유지성(ResultSet Holdability)

  결과 집합이 열려있는 동안에 커밋이 발생했을 때, 결과 집합을 그대로 유지할지 아니면 닫을지를 설정한다.

- 준비된 문장의 풀링(pooling)

  애플리케이션 서버를 통해 준비된 문장의 풀링을 구현하고, 여러 애플리케이션의 접속이 이를 재사용한다.

- 접속 풀링(connection pooling)의 설정

  JDBC 3.0에서는 최대 풀 크기, 최소 풀 크기, 초기 풀 크기 등과 같이 접속 풀링에 사용되는 다양한 파라미터를 설정할 수 있는 API를 제공한다.

- 자동 생성 키

  SQL 문장을 실행한 후 결과 집합으로부터 `getGeneratedKeys` 함수를 사용하여 결과 로우에 대한 키를 얻는다.

- 복수개의 결과 집합 반환

  JDBC 3.0에서는 한 SQL 문장이 여러 개의 결과 집합을 열 수 있도록 지원한다.

- 추가된 데이터 타입

  – java.sql.Types.DATALINK

    URL과 같은 외부 리소스의 접근을 제공한다.

  – java.sql.Types.BOOLEAN

    BIT 타입과 논리적으로 동일하다.

- REF 객체에 의해 참조된 객체의 검색 및 수정

REF 객체에 의해 참조된 객체를 검색하고, 수정할 수 있는 기능을 제공한다.

- BLOB와 CLOB 객체에 존재하는 데이터의 수정

  BLOB와 CLOB 객체에 포함된 일부 데이터를 수정하는 기능을 제공한다.

- BLOB과 CLOB과 ARRAY 및 REF 데이터 타입의 컬럼 값 수정

  updateBlob와 updateClob와 updateArray 및 updateRef를 통해 해당 데이터 타입의 컬럼 값을 수정한다.

- 그룹 변환 및 데이터 타입의 매핑

  JDBC API를 통해 사용자 정의 데이터 타입(UDT:User Defined Types)과 Java 클래스 간의 매핑 관계를 설정할 수 있다. 그러나, Tibero RDBMS에서는 사용자 정의 데이터 타입을 지원하지 않는다.

2.3. 기본 프로그래밍

본 절에서는 tbJDBC에서 제공하는 인터페이스 메소드를 통해 기본적인 Java 프로그램을 작성하는 방법을 설명한다.

참고

본 절에서 설명하고 있는 기본 프로그램의 전체 소스 코드는 “Appendix A. tbJDBC 예제 프로그램 소스 코드”를 참고한다.

다음은 public class가 포함된 JdbcTest 클래스 파일의 일부이다.

[예 2.1] tbJDBC를 사용한 기본 프로그래밍

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;

public class JdbcTest
{
    Connection conn;

    public static void main(String[] args) throws Exception
   {
```

```
        JdbcTest test = new JdbcTest();

        test.connect();                      ... ⓐ ...
        test.executeStatement();             ... ⓑ ...
        test.executePreparedStatement();     ... ⓒ ...
        test.executeCallableStatement();     ... ⓓ ...
        test.disconnect();                   ... ⓔ ...
    }

    /* ... 기능별 멤버 함수를 구현한다. */
}
```

ⓐ JdbcTest 라는 클래스를 통해 데이터베이스에 접속한다.

ⓑ ~ ⓓ 여러 종류의 멤버 함수를 수행한다.

ⓔ 데이터베이스 접속을 해제하는 작업을 수행한다.

다음 절부터는 위 예에서 ⓐ ~ ⓔ 사이에 구현된 멤버 함수를 각각 설명한다.

2.3.1. 접속

connect 멤버 함수는 기본 드라이버 관리자를 이용하여 데이터베이스에 접속(connection)하는 기능(ⓐ)을 수행한다. 단, 이 멤버 함수를 사용하려면 반드시 **java.sql.DriverManager** 인터페이스 메소드가 선언되어 있어야 한다.

[예 2.2] tbJDBC - 접속

```
private void connect() throws SQLException, ClassNotFoundException
{
    Class.forName("com.tmax.tibero.jdbc.TbDriver");

    conn = DriverManager.getConnection(
            "jdbc:tibero:thin:@localhost:8629:tibero",
            "tibero", "tmax");

   if (conn == null)
   {
        System.out.println("connection failure!");
        System.exit(-1);
    }
    System.out.println("Connection success!");
}
```

구현된 내용을 차례대로 설명하면 다음과 같다.

1. Class.forName 메소드를 호출한다.

com.tmax.tibero.jdbc.TbDriver 클래스를 로드 한다. 즉, 사용자가 사용할 기본 드라이버가 등록된다.

2. DriverManager.getConnection 메소드를 호출한다.

   등록된 드라이버를 통해 데이터베이스 접속을 요청한다. 사용자는 DriverManager.getConnection 메소드의 파라미터로 접속할 데이터베이스의 URL를 입력한다. 이때 DriverManager 클래스는 데이터베이스 접속이 가능한지 드라이버를 검사한다.

3. conn 객체에 `getConnection` 메소드로 새로운 데이터베이스 접속을 생성한다.

   접속을 해제할 때는 `close` 멤버 함수를 사용한다.

2.3.2. 실행

tbJDBC에서는 다른 데이터베이스의 클라이언트에서처럼 문장(statement)에 대한 개념이 존재한다.

`executeStatement` 멤버 함수는 이러한 문장을 실행(execution)하는 기능(ⓑ)을 수행한다.

[예 2.3] tbJDBC - 실행

```
private void executeStatement() throws SQLException
{
    String dropTable   = "drop table emp";
    String createTable = "create table emp (id number, "+
                         " name varchar(20), salary number)";
    String InsertTable = "insert into emp values(1000, 'Park', 5000)";

    Statement stmt = conn.createStatement();

    try {
        stmt.executeUpdate(dropTable);
    } catch(SQLException e) {
        // if there is not the table
    }

    stmt.executeUpdate(createTable);
    stmt.executeUpdate(InsertTable);

    stmt.close();
}
```

구현된 내용을 차례대로 설명하면 다음과 같다.

1. dropTable, createTable, InsertTable를 각각 String 객체 참조형 변수로 만든다.

   객체 참조형 변수에 각각 특정 명령을 실행할 SQL 문장을 만든다.

2. conn 객체의 `createStatement` 메소드로 새로운 문장을 생성한다.

3. `executeUpdate(str)` 메소드를 이용하여 특정 명령을 실행한다.

또한, 문장을 생성한 후에는 파라미터를 바인딩하여 원하는 작업을 수행할 수 있다.

`executePreparedStatement` 멤버 함수는 파라미터가 바인딩이 된 질의를 실행하는 기능(ⓒ)을 수행한다.

[예 2.4] tbJDBC - 준비된 문장의 선언과 파라미터 바인딩

```
private void executePreparedStatement() throws SQLException
{
    PreparedStatement pstmt = conn
        .prepareStatement("select name from emp where id = ?");    ... ① ...

    pstmt.setString(1, "1000");

    ResultSet rs = pstmt.executeQuery();

    while (rs.next()) {
        System.out.println(rs.getString(1));
    }

    pstmt.close();
}
```

구현된 내용을 차례대로 설명하면 다음과 같다.

1. PreparedStatement 클래스로 준비된 문장(prepared statement)를 선언한다.

2. conn 객체의 `prepareStatement` 메소드를 이용하여 PreparedStatement를 생성한다. 이렇게 생성된 준비된 문장은 execute로 결과를 얻을 수 있다.

3. `setString(bind_no, bind_value)` 메소드를 이용하여 바인딩 할 파라미터의 순서를 정하여 값을 설정한다. **pstmt.setString(1, "1000")** 은 준비된 문장의 첫 번째 파라미터(①)에 대응되는 **"1000"** 이라는 문자열을 바인딩한다.

4. 파라미터의 바인딩이 완료된 후 `executeQuery` 메소드를 실행하면 결과 집합(ResultSet)를 얻게 된다. 결과 집합은 개념적으로 tbCLI나 tbESQL의 커서와 동일하다. 결과 집합에 `getString(result_no)` 메소드를 사용하면 실행 결과를 문자열로 확인할 수 있다.

2.3.3. 호출

tbJDBC를 이용하여 작성한 프로그램에서 PL/SQL를 호출(call)할 수 있다. 이러한 문장을 **호출 가능 문장**(callable statement)이라고 한다.

`executeCallableStatement` 멤버 함수는 호출 가능 문장을 실행하는 기능(ⓓ)을 수행한다.

[예 2.5] tbJDBC - 호출

```
private void executeCallableStatement() throws SQLException
{
    String callSQL =
        " CREATE PROCEDURE testProc "+
        " (ID_VAL IN NUMBER, SAL_VAL IN OUT NUMBER) as " +
        " BEGIN" +
        "   update emp" +
        "      set salary = SAL_VAL" +
        "    where id = ID_VAL;" +
        "   select salary into SAL_VAL" +
        "     from emp" +
        "    where id = ID_VAl;" +
        " END;";
    String dropProc = "DROP PROCEDURE testProc";

    Statement stmt = conn.createStatement();

    try {
        stmt.executeUpdate(dropProc);
    } catch(SQLException e) {
        // if there is not the procedure
    }

    stmt.executeUpdate(callSQL);

    CallableStatement cstmt = conn.prepareCall("{call testProc(?, ?)}");
    cstmt.setInt(1, 1000);
    cstmt.setInt(2, 7000);
    cstmt.registerOutParameter(2, Types.INTEGER);
    cstmt.executeUpdate();

    int salary = cstmt.getInt(2);
    System.out.println(salary);

    stmt.close();
    cstmt.close();
}
```

구현된 내용을 차례대로 설명하면 다음과 같다.

1. CREATE PROCEDURE 문장을 이용하여 프로시저를 만든다.

   DDL 문장은 다른 문장처럼 동일하게 `executeUpdate(query_str)` 메소드를 이용하여 실행한다.

2. 호출 가능 문장은 **CallableStatement** 클래스를 통해 conn 객체의 `prepareCall(str)` 메소드를 이용하여 생성한다.

   이렇게 생성된 호출 가능 문장은 준비된 문장과 동일하게 파라미터를 바인딩한다.

   파라미터를 바인딩하는 방법은 `setInt(bind_no, bind_value)` 같은 바인딩 메소드(binding method)를 사용한다. 즉 `setInt`등의 메소드를 사용하고 나서 `executeUpdate` 메소드를 호출하면 된다.

3. CallableStatement 클래스의 `registerOutParameter(bind_no, type)` 메소드를 이용하여 바인딩 된 파라미터 중에서 출력 파라미터를 등록한다.

   등록할 출력 파라미터는 1번에서 생성된 프로시저의 두 번째 파라미터(**SAL_VAL IN OUT NUMBER**)이다. 이 파라미터는 입출력용으로 선언되었으므로 출력 파라미터로 등록할 수 있다.

4. `executeUpdate` 메소드를 호출하여 문장을 실행한 후, 출력 파라미터의 타입에 맞는 메소드로 실행 결과를 가져온다. 프로시저의 출력 파라미터는 integer 타입으로 CallableStatement 클래스의 `getInt(bind_no)` 메소드를 이용하여 salary에 결과 값을 출력한다.

2.3.4. 커밋과 롤백

트랜잭션은 하나 이상의 SQL 문장으로 이루어져 있다. SQL 문장이 실행되면서 전체가 커밋(Commit) 되든지 롤백(Rollback)된다.

tbJDBC는 Connection 객체에서 트랜잭션을 설정할 수 있다. 기본값은 **auto-commit**으로 설정되어 있다.

트랜잭션의 처리 모드를 변경하려면 다음과 같이 소스 코드를 추가해야 한다.

```
conn.setAutoCommit(false);
conn.rollback();
conn.commit();
conn.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITED)
```

2.3.5. 접속 해제

`disconnect` 멤버 함수는 데이터베이스 접속을 해제하는 기능(ⓔ)을 수행한다.

[예 2.6] tbJDBC - 접속 해제

```
private void disconnect() throws SQLException
{
    if (conn != null)
```

```
        conn.close();
}
```

conn 객체가 존재하는 경우, `close` 멤버 함수를 사용하여 데이터베이스 접속을 해제할 수 있다.

제3장 트리거의 사용

본 장에서는 트리거의 기본 개념과 이를 생성하는 방법을 설명한다.

3.1. 개요

트리거(Trigger)는 스키마 객체의 일종으로, 데이터베이스가 미리 정해 놓은 특정 조건이 만족되거나 어떤 동작이 수행되면 자동으로 실행되도록 정의한 동작이다. 예를 들어 데이터베이스에 특정한 이벤트가 발생되거나 사용자가 설정한 DDL 문장이 수행될 때 트리거가 실행될 수 있다.

트리거의 내용은 PSM으로 이루어져 있다. 트리거는 이미 정의된 PSM 객체를 호출하여 사용하거나 트리거를 생성할 때 이름 없는 블록을 함께 선언해주는 방식을 모두 이용할 수 있다.

3.1.1. 트리거의 구성요소

트리거는 다음과 같은 세 가지 구성요소를 갖춰야만 생성할 수 있다.

- 트리거가 실행될 조건이 되는 문장이나 이벤트
- 실행 조건의 제약
- 실행될 내용

다음은 **alarm_for_balance**라는 트리거를 생성하는 예이다.

```
CREATE OR REPLACE TRIGGER alarm_for_balance
BEFORE INSERT OR UPDATE ON balance_tab        ... ⓐ ...
FOR EACH ROW
WHEN (new.balance < 3000)                     ..  ⓑ ...
    CALL alarm_for_balance_fn()               ..  ⓒ ...
```

ⓐ 테이블 balance_tab의 balance 컬럼에 로우가 삽입되거나 수정이 발생했을 때 ⓑ로 이동한다.

ⓑ 해당 로우의 값이 3000 이하인지를 검사한다.

ⓒ 검사한 결과가 맞으면 `alarm_for_balance_fn` 함수를 호출하고, 함수에 정의한 동작이 실행된다.

3.1.2. 트리거의 타입

트리거의 타입은 다음과 같이 나눌 수 있다.

- 로우(row) 및 문장(statement)

타입	설명
로우	테이블에 INSERT, UPDATE, DELETE가 발생하는 로우마다 트리거의 내용이 실행되는 타입이다. 이 타입의 트리거는 각 로우에 연산이 발생할 때마다 연산 직전 또는 직후에 트리거가 실행된다.
문장	로우의 개수에 상관없이 문장 단위로 한 번만 실행되는 타입이다.

- BEFORE 및 AFTER

타입	설명
BEFORE	조건 문장이 실행되기 전에 트리거의 내용이 실행되는 타입이다.
AFTER	조건 문장이 실행된 후 트리거의 내용이 실행되는 타입이다.

트리거는 두 종류의 타입 중에서 하나씩을 각각 가질 수 있다. 즉, BEFORE 로우(BEFORE row), BEFORE 문장(BEFORE statement), AFTER 로우(AFTER row), AFTER 문장(statement)의 타입으로 생성할 수 있다.

3.2. 트리거의 생성

트리거를 생성하는 방법은 다음과 같다.

```
CREATE [OR REPLACE] TRIGGER 트리거_이름
{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON 테이블_이름
[FOR EACH ROW]
WHEN (조건_제약)
{[선언부]
BEGIN
    ...
END;} |
CALL 함수_혹은_프로시저_이름
```

참고

트리거의 문법에 대한 자세한 내용은 "Tibero RDBMS SQL 참조 안내서"를 참고한다.

다음은 로우 타입의 트리거로, 테이블 DECK_TBL의 COUNT 컬럼에 로우 값이 1000을 초과할 때마다 로그를 기록하도록 작성한 예이다.

[예 3.1] 트리거의 생성

```
CREATE OR REPLACE TRIGGER Log_overflow
AFTER UPDATE ON Deck_tbl
FOR EACH ROW
WHEN (new.count > 1000)
BEGIN
    INSERT
    INTO Deck_log (Deck_id, Timestamp, New_count, Action)
    VALUES (:new.Deck_no, SYSTIMESTAMP, :new.count, 'overflow');
END;
```

CREATE 문장은 Tibero RDBMS 내부에서는 BEGIN – END 사이의 부분을 PSM으로 인식한다. 이 문장이 컴파일 되면 PSM 스키마 객체가 생성되고 이를 데이터베이스에 저장한다. 만약 컴파일 에러가 발생한다면 정적 뷰를 통해 에러의 내용을 확인할 수 있다.

제4장 XA의 사용

본 장에서는 분산 트랜잭션(distributed transaction)를 처리하는 데 사용되는 XA를 설명한다.

4.1. 분산 트랜잭션

하나의 데이터베이스 인스턴스 내에서는 한 트랜잭션으로 묶인 여러 개의 SQL 문장이 모두 커밋되거나 롤백된다.

네트워크로 연결된 여러 개의 데이터베이스 인스턴스가 참여하는 트랜잭션에서도 마찬가지로 각각 다른 데이터베이스 인스턴스에서 수행한 SQL 문장이 모두 동시에 커밋되거나 롤백될 수 있는 방법이 필요하다. 이렇게 여러 개의 노드 또는 다른 종류의 데이터베이스가 참여하는 하나의 트랜잭션을 **분산 트랜잭션**(distributed transaction)이라 한다.

참고

분산 트랜잭션에 대한 자세한 내용은 "Tibero RDBMS 관리자 안내서"를 참고한다.

4.1.1. Two-phase commit

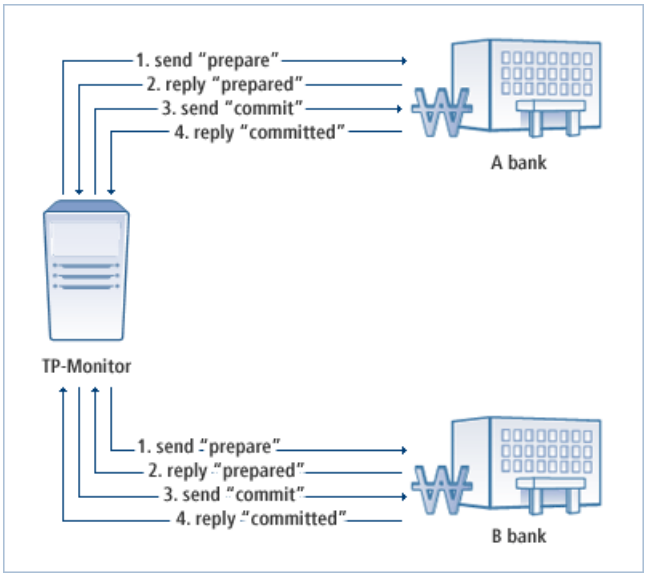
Tibero RDBMS는 X/Open DTP(Distributed Transaction Processing) 규약의 XA를 지원한다. XA는 Two-phase commit를 이용하여 분산 트랜잭션을 처리한다.

분산 환경에서 트랜잭션의 무결성을 보장하기 위해서 사용하는 커밋 방법은 **Two-phase commit**이다.

보통 두 개 이상의 노드가 특정 트랜잭션을 함께 수행하고 있다면, 일반적으로 사용자의 요청을 받아 트랜잭션을 시작한 노드가 코디네이터가 된다. TP-Monitor가 있는 시스템일 경우에는 TP-Monitor가 그 역할을 한다.

다음은 Two-phase commit의 일반적인 예를 나타내는 그림이다.

[그림 4.1] Two-phase commit의 일반적인 예



Two-phase commit mechanism은 크게 두 단계로 작업이 이루어진다. 위 예제를 기준으로 Two-phase commit를 설명하자면 다음과 같다.

1. **First Phase (또는 prepare phase)**

   커밋을 준비하는 단계로 다음의 세부 과정으로 실행된다.

단계	설명
send "prepare"	각 노드는 코디네이터 노드로부터 커밋을 준비하라는 메시지를 받는다.
reply "prepared"	각 노드는 메시지를 받은 후 복구를 위해 로그 등에서 커밋이 가능한지를 검사한다. 또는 필요에 따라 자체적으로 롤백을 한다. 만약 커밋이 가능하다면 최종으로 커밋한 로그를 제외한 모든 작업을 수행한 후, 커밋이 준비되었다는 메시지를 코디네이터 노드로 전달한다.

2. **Second Phase (또는 commit phase)**

   실제로 커밋한 기록을 저장하는 단계로 Second Phase은 다음의 세부 과정으로 실행된다.

단계	설명
send "commit"	코디네이터 노드는 모든 노드에서 커밋이 되었다는 메시지를 전달받는다. 이를 확인한 후 실제로 커밋을 실행하라는 메시지를 각 노드에 보낸다.
reply "committed"	각 노드는 커밋을 기록한 후 커밋이 완료 되었다는 메시지를 코디네이터 노드로 전달한다.

4.1.2. In-doubt 트랜잭션

Two-phase commit mechanism에 의해 첫 번째 prepare 메시지를 받으면 데이터베이스는 분산 트랜잭션에 해당하는 리소스를 잠금을 설정하거나 로그를 남김으로써 커밋할 준비를 한다. 그런데 **prepare**까지 마친 상태에서 네트워크의 이상으로 다음 메시지(커밋 혹은 롤백)를 받지 못하는 경우가 발생할 수 있다. 이 경우에 데이터베이스는 해당 트랜잭션을 커밋해야 할 지 롤백 해야 할지 판단할 수 없다.

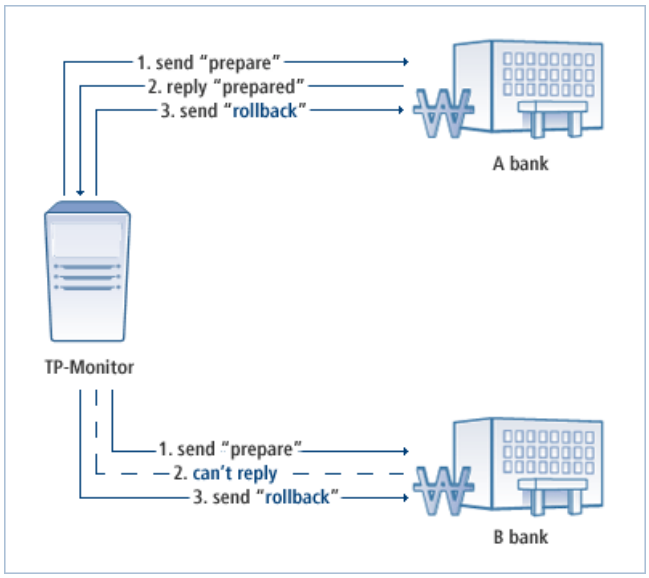
따라서 다음 메시지가 올 때까지 **prepare**된 리소스에 잠금 설정을 한 채로 기다리게 된다. 이렇게 prepare는 되었는데 그 다음 메시지를 받지 못한 채 리소스만 소유하고 기다리고 있는 트랜잭션을 **In-doubt 트랜잭션**이라 한다.

예를 들어, First Phase에서 코디네이터 노드가 커밋을 준비하라는 메시지를 보냈음에도 불구하고 어떤 특정 노드의 서버가 다운되었거나 네트워크 상태가 불안정하여 그 메시지를 못 받았거나 또는 메시지는 받았지만 커밋이 준비되었다는 답변을 받지 못했을 때 In-doubt 트랜잭션이 발생한다.

이러한 경우 코디네이터 노드는 다른 모든 노드의 응답을 받았어도 한 노드의 응답을 받지 못했으므로, 이 트랜잭션을 In-doubt 트랜잭션으로 표시하고, 모든 노드에 Second Phase의 커밋 명령을 실행하라는 메시지를 보내는 대신에 롤백 하라는 메시지를 보낸다.

예를 들면 다음 그림과 같다.

[그림 4.2] In-doubt 트랜잭션이 발생하는 예



또한, 모든 노드에서 커밋이 준비되었다는 메시지를 받아서 확인했더라도 그 이후에 코디네이터 노드가 보낸 Second Phase의 커밋 명령을 실행하라는 메시지를 특정 노드가 받지 못했거나, 그 노드가 커밋 명령은 잘 수행하였으나 커밋이 완료되었다는 응답을 코디네이터 노드에 전달하지 못했을 수도 있다. 이와 같은 경우도 마찬가지로 In-doubt 트랜잭션으로 분류된다.

트랜잭션은 커밋이나 롤백이 확정되지 않은 채로 데이터베이스 리소스에 대해 잠금(Lock)이 설정된 상태 즉 정체(pending) 상태가 된다. 따라서 이를 해결하려면 첫 번째 경우처럼 코디네이터 노드가 자동으로 전체 트랜잭션을 롤백 해 준다거나 DBA가 이러한 트랜잭션을 추출하여 자체적으로 판단하여 수동으로 처리할 수 있다.

이때 DBA가 In-doubt 트랜잭션을 추출하기 위해 정보를 볼 수 있는 테이블과 뷰로는 **VT_XA_BRANCH**와 **DBA_2PC_PENDING**이 있다.

참고

VT_XA_BRANCH는 현재 데이터베이스 서버에서 활동 중인 모든 XA 트랜잭션 브랜치 (XA transaction branch)의 정보를 실시간으로 볼 수 있는 테이블이다. 그리고 DBA_2PC_PENDING은 현재 정체되고 있는 XA 트랜잭션 브랜치의 정보를 보여주는 뷰이다. 이에 대한 자세한 내용은 각각 "Tibero RDBMS 참조 안내서"와 "Tibero RDBMS 관리자 안내서"를 참고한다.

4.2. XA API

TP-Monitor를 이용하여 분산 트랜잭션을 수행하기 위해서는 XA API를 사용해야 한다.

Tibero RDBMS는 X/Open DTP 규약의 XA를 지원하기 때문에 표준에 맞는 XA 애플리케이션 프로그램을 작성할 수 있다.

4.2.1. XA 함수

Tibero RDBMS에서는 XA를 지원하기 위해 다음과 같은 XA 함수를 제공한다. 단, 이 함수는 **C 프로그래밍 언어용**으로만 제공된다.

다음은 XA 함수를 나열한 표이다.

함수	설명
xa_open	리소스 매니저(Resource Manager)에 접속한다.
xa_close	리소스 매니저에서 데이터베이스 접속을 해제한다.
xa_start	XID에 새로운 트랜잭션을 시작하거나, 이미 존재하는 트랜잭션에 현재 프로세스를 연결한다.
xa_end	XID에서 현재 프로세스를 분리한다
xa_rollback	XID의 트랜잭션을 롤백한다.
xa_prepare	XID에 커밋을 준비한다. Two-phase commit의 First Phase이다.
xa_commit	XID에 커밋을 완료한다. Two-phase commit의 Second Phase이다.
xa_recover	prepare 상태인 트랜잭션의 목록을 검사하여 커밋이나 롤백을 수행한다.
xa_forget	XID의 트랜잭션이 이미 처리된 경우 로그 기록을 삭제한다.

4.2.2. xa_open 함수의 속성

다음은 xa_open 함수를 호출하는 데 필요한 속성이다.

이름	필수	작성 예	설명
user	O	user=tibero	접속할 사용자의 이름이다.
pwd	O	pwd=1234	접속할 사용자의 패스워드이다.
sestm	O	sestm=10	시스템에 의해 중단(abort)되기 전까지의 트랜잭션의 inactive time limit이다. 클라이언트로부터 한 요청에 대한 답변을 전달한 후 그 다음 요청이 오기 전까지 대기하는 시간이다. 그 이상의 시간이 지나면 클라이언트 측에서 문제가 있다고 판단하고 해당 xtb를 롤백한다.
db	X	db=sample	접속할 데이터베이스의 DSN 이름(tbdsn.tbr 파일 내의 DSN 이름)이다.
LogDir	X	LogDir=home/db/tibero/log	XA API가 에러, 트레이스 메시지를 저장할 디렉터리의 이름이다.
Loose_Coupling	O	Loose_Coupling=false	loosely coupling flag로 true/false설정한다. 다른 브랜치이지만 동일한 글로벌 트랜잭션끼리 같은 리소스를 사용하는지 여부를 설정한다. loose coupling이면 (1,0)과 (1,2)는 서로 다른 내부 리소스를 사용한다. 예를 들어 TX가 해당된다. 이와는 반대로 tight coupling이면 두 xtb는 하나의 TX를 서로 잠금 처리를 설정해가며 공유하여 사용한다.
seswt	X	seswt=30	XA_RETRY가 반환되기 전까지 데이터베이스 서버가 트랜잭션을 기다리는 대기 시간이다. flag 내에 TMNOWAIT이 설정되어 있지 않을 경우에는 클라이언트의 요청을 데이터베이스가 곧바로 처리해줄 수 없을 때 해당 리소스를 사용할 수 있을 때까지 기다린 후에 클라이언트에 답변을 주게 된다. 이때 데이터베이스가 sesswt로 설정한 시간까지 답변을 줄 수 없는 경우 XARETRY를 반환한다.

4.2.3. XA 애플리케이션 프로그래밍

일반적으로 C 프로그래밍 언어에서 XA 애플리케이션 프로그램을 작성할 때 ESQL을 이용하여 개발하는 예가 많다.

다음은 Tibero RDBMS에서 제공하는 **tbESQL**를 이용하여 XA 애플리케이션 프로그램을 작성한 예이다.

```
#define XA_CONN_STR_TIGHT "TIBERO_XA:user=tibero, pwd=tmax," \
                          "db=sample, sestm=60"

/* 동일한 글로벌 트랜잭션에서 다른 브랜치로 xa_start했을 경우
* Tight-Coupling
*/
void test_xa_2branch_2pc_tight()
{
    int rc;
    XID xid1;
    XID xid2;
    struct xa_switch_t *tbxa = &XA_SWITCH_NAME;
    char *conn_str = XA_CONN_STR_TIGHT; /* tightly coupled */
    long gtrid = _GTRID_BASE;
    long bqual = 1;

    EXEC SQL BEGIN DECLARE SECTION;
        int cnt;
    EXEC SQL END DECLARE SECTION;

    /* xa_open */
    tbxa->xa_open_entry (conn_str, 0, TMNOFLAGS);
    xid1.formatID = 1;
    xid1.gtrid_length = sizeof(gtrid);
    xid1.bqual_length = sizeof(bqual);
    memcpy(&xid1.data[0], &gtrid, sizeof(gtrid));
    memcpy(&xid1.data[sizeof(gtrid)], &bqual, sizeof(bqual));

    bqual = 2;
    xid2.formatID = 1;
    xid2.gtrid_length = sizeof(gtrid);
    xid2.bqual_length = sizeof(bqual);
    memcpy(&xid2.data[0], &gtrid, sizeof(gtrid));
    memcpy(&xid2.data[sizeof(gtrid)], &bqual,
    sizeof(bqual));

    EXEC SQL DELETE FROM PERSON;
    EXEC SQL COMMIT WORK;

    /* (1, 1) 시작 */
```

```
    /* xa_start -- sql statements starts */
    tbxa->xa_start_entry (&xid1, 0, TMNOFLAGS);
    EXEC SQL INSERT INTO PERSON VALUES ('1', 'LEE');
    EXEC SQL INSERT INTO PERSON VALUES ('2', 'KIM');

    /* xa_end -- */
    tbxa->xa_end_entry (&xid1, 0, TMSUCCESS);

    /* (1, 2) 시작 */
    /* xa_start -- sql statements starts */
    tbxa->xa_start_entry (&xid2, 0, TMNOFLAGS);
    EXEC SQL INSERT INTO PERSON VALUES ('2', 'PARK');
    EXEC SQL INSERT INTO PERSON VALUES ('3', 'JAKE');
    EXEC SQL INSERT INTO PERSON VALUES ('4', 'KID');
    EXEC SQL INSERT INTO PERSON VALUES ('5', 'CHANHO');

    /* xa_end -- */
    tbxa->xa_end_entry (&xid2, 0, TMSUCCESS);

    /* xa_prepare */
    tbxa->xa_prepare_entry (&xid1, 0, TMNOFLAGS);

    /* Tightly-Coupled 가정 */
    /* xa_prepare */
    tbxa->xa_prepare_entry (&xid2, 0, TMNOFLAGS);

    /* xa_commit */
    tbxa->xa_commit_entry (&xid1, 0, TMNOFLAGS);

    /* xa_commit */
    tbxa->xa_commit_entry (&xid2, 0, TMNOFLAGS);
    EXEC SQL SELECT COUNT(*) into :cnt FROM PERSON;
    CuAssertIntEq(tc, cnt, 6);

    /* xa_close */
    tbxa->xa_close_entry ("", 0, TMNOFLAGS);
    return;
}

/* 같은 Global Trasaction의 다른 Branch로 xa_start했을 경우
* Loose-Coupling
*/
void test_xa_2branch_2pc_loose(CuTest *tc)
{
    int rc;
    XID xid1;
    XID xid2;
```

```
struct xa_switch_t *tbxa = &XA_SWITCH_NAME;
char *conn_str = XA_CONN_STR_LOOSE;
long gtrid = _GTRID_BASE;
long bqual = 1;

EXEC SQL BEGIN DECLARE SECTION;
    int cnt;
EXEC SQL END DECLARE SECTION;

/* xa_open */
tbxa->xa_open_entry (conn_str, 0, TMNOFLAGS);
xid1.formatID = 1;
xid1.gtrid_length = sizeof(gtrid);
xid1.bqual_length = sizeof(bqual);
memcpy(&xid1.data[0], &gtrid, sizeof(gtrid));
memcpy(&xid1.data[sizeof(gtrid)], &bqual, sizeof(bqual));

bqual = 2;
xid2.formatID = 1;
xid2.gtrid_length = sizeof(gtrid);
xid2.bqual_length = sizeof(bqual);
memcpy(&xid2.data[0], &gtrid, sizeof(gtrid));
memcpy(&xid2.data[sizeof(gtrid)], &bqual,
sizeof(bqual));

EXEC SQL DELETE FROM PERSON;
EXEC SQL COMMIT WORK;

/* (1, 1) 시작 */
/* xa_start -- sql statements starts */
tbxa->xa_start_entry (&xid1, 0, TMNOFLAGS);
EXEC SQL INSERT INTO PERSON VALUES ('1', 'LEE');
EXEC SQL INSERT INTO PERSON VALUES ('2', 'KIM');

/* xa_end -- */
tbxa->xa_end_entry (&xid1, 0, TMSUCCESS);

/* (1, 2) 시작 */
/* xa_start -- sql statements starts */
tbxa->xa_start_entry (&xid2, 0, TMNOFLAGS);
EXEC SQL INSERT INTO PERSON VALUES ('2', 'PARK');
EXEC SQL INSERT INTO PERSON VALUES ('3', 'JAKE');
EXEC SQL INSERT INTO PERSON VALUES ('4', 'KID');
EXEC SQL INSERT INTO PERSON VALUES ('5', 'CHANHO');

/* xa_end -- */
tbxa->xa_end_entry (&xid2, 0, TMSUCCESS);
```

```
    /* xa_prepare */
    tbxa->xa_prepare_entry (&xid1, 0, TMNOFLAGS);

    /* Loosely-Coupled 가정 */
    /* xa_prepare */
    tbxa->xa_prepare_entry (&xid2, 0, TMNOFLAGS);

    /* xa_commit */
    tbxa->xa_commit_entry (&xid1, 0, TMNOFLAGS);

    /* xa_commit */
    tbxa->xa_commit_entry (&xid2, 0, TMNOFLAGS);
    EXEC SQL SELECT COUNT(*) into :cnt FROM PERSON;
    CuAssertIntEq(tc, cnt, 6);

    /* xa_close */
    tbxa->xa_close_entry ("", 0, TMNOFLAGS);
    return;
}
```

4.3. JDBC에서의 XA 지원

본 절에서는 XA(Extended Architecture)에서 지원하는 인터페이스와 이를 이용하여 작성한 프로그램에 대해 기술한다.

4.3.1. XA 인터페이스

Tibero RDBMS에서 지원하는 JDBC의 XA 인터페이스는 다음과 같다.

- XA Datasource Interface
- XA Connection Interface
- XA Exception Interface
- XA XID Interface

다음은 Tibero RDBMS에서 구현된 XA 인터페이스의 목록이다.

표준 XA 인터페이스(JDK 1.3)	**Tibero RDBMS의 XA 인터페이스**
javax.sql.XADataSource	com.tmax.tibero.jdbc.ext.TbXADataSource

표준 XA 인터페이스(JDK 1.3)	Tibero RDBMS의 XA 인터페이스
javax.sql.XAConnection	com.tmax.tibero.jdbc.ext.TbXAConnection
javax.transaction.xa.XAException	com.tmax.tibero.jdbc.ext.TbXAException
javax.transaction.xa.Xid	com.tmax.tibero.jdbc.ext.TbXid

4.3.2. XA 인터페이스 프로그래밍

다음은 tbJDBC 환경에서 XA 인터페이스를 이용하여 프로그래밍한 예이다.

```
package test.com.tmax.tibero.cases;
import com.tmax.tibero.jdbc.ext.*;
import test.com.tmax.tibero.AbstractBase;
import javax.sql.XAConnection;
import javax.transaction.xa.XAResource;
import java.sql.*;

public class TestXATwoBranch extends AbstractBase{
    int formatID = 1;
    int row_count = 0;
    int pre1 , pre2 = 0;

    byte[] gtid1 = new byte[1];
    byte[] bq1 = new byte[1];
    byte[] gtid2 = new byte[1];
    byte[] bq2 = new byte[1];

    public TestXATwoBranch (String name) {
    super(name);
    }

    /* Tightly Coupled 일 때 */
    public void test_xa_2branch_2pc_tight () throws Exception {
        debug("test_xa_2branch_2pc_tight " + this._getClassName());

        create_table_for_xa();

        gtid1[0] = (byte)1; bq1[0] = (byte)1;
        gtid2[0] = (byte)1; bq2[0] = (byte)2;

        TbXADataSource xads1 = new TbXADataSource();
        xads1.setUrl(getXAurl());
        xads1.setUser(getXAuser());
        xads1.setPassword(getXApasswd());
```

```
TbXADataSource xads2 = new TbXADataSource();
xads2.setUrl(getXAurl());
xads2.setUser(getXAuser());
xads2.setPassword(getXApasswd());

XAConnection xacon1 = xads1.getXAConnection();
XAConnection xacon2 = xads2.getXAConnection();

Connection conn1 = xacon1.getConnection();
Connection conn2 = xacon2.getConnection();

XAResource xars1 = xacon1.getXAResource();
XAResource xars2 = xacon2.getXAResource();

/* XID (1.1) 생성 */
TbXid xid1 = new TbXid(formatID, gtid1,bq1 );

/* XID (1.2) 생성 */
TbXid xid2 = new TbXid(formatID, gtid2, bq2);
try {
    /* (1.1) 시작 */
    xars1.start(xid1, XAResource.TMNOFLAGS);

    PreparedStatement pstmt1;
    pstmt1 = conn1.prepareStatement(
                "insert into author values (?,?)");

    pstmt1.setInt(1, 1);
    pstmt1.setString(2, "FOSCHIA");
    pstmt1.executeUpdate();

    pstmt1.setInt(1,2);
    pstmt1.setString(2, "AGNOS");
    pstmt1.executeUpdate();

    /* (1.1) 종료 */
    xars1.end(xid1, XAResource.TMSUCCESS);

    /* (1.2) 시작 */
    xars2.start(xid2, XAResource.TMNOFLAGS);
    PreparedStatement pstmt2;
    pstmt2 = conn2.prepareStatement(
                "insert into author values (?,?)");

    pstmt2.setInt(1, 3);
    pstmt2.setString(2, "JELLA");
    pstmt2.executeUpdate();
```

```
        pstmt2.setInt(1,4);
        pstmt2.setString(2, "THIPHILO");
        pstmt2.executeUpdate();

        /* (1.2) 종료 */
        xars2.end(xid2, XAResource.TMSUCCESS);

        /* (1,1) prepare */
        pre1= xars1.prepare(xid1);
        assertEquals(pre1, XAResource.XA_RDONLY);

        /* (1,2) prepare */
        pre2 = xars2.prepare(xid2);
        assertEquals(pre2, XAResource.XA_OK);

        /* (1.1) commit */
        try {
            xars1.commit(xid1, false);
        } catch(TbXAException e) {}

        /* (1.2) commit */
        try {
            xars2.commit(xid2, false);
        } catch(TbXAException e) {}

        Statement stmt1 = conn1.createStatement();
        ResultSet rs1 =
            stmt1.executeQuery("select * from author");

        while (rs1.next())
            row_count ++;

        assertEquals(4, row_count);

        rs1.close(); rs1 = null;
        stmt1.close(); stmt1 = null;

        pstmt1.close(); conn1.close(); xacon1.close();
        pstmt2.close(); conn2.close(); xacon2.close();

        pstmt1= null; conn1= null; xacon1=null;
        pstmt2= null; conn2= null; xacon2=null;
    } catch (TbXAException e) {}
}

/* Loosely Coupled 일 때 */
```

```
public void test_xa_2branch_2pc_loose () throws Exception {
    debug("test_xa_2branch_2pc_loose " + this._getClassName());

    create_table_for_xa();

    gtid1[0] = (byte)1; bq1[0] = (byte)1;
    gtid2[0] = (byte)1; bq2[0] = (byte)2;

    TbXADataSource xads1 = new TbXADataSource();

    xads1.setUrl(getXAurl());
    xads1.setUser(getXAuser());
    xads1.setPassword(getXApasswd());

    TbXADataSource xads2 = new TbXADataSource();

    xads2.setUrl(getXAurl());
    xads2.setUser(getXAuser());
    xads2.setPassword(getXApasswd());

    XAConnection xacon1 = xads1.getXAConnection();
    XAConnection xacon2 = xads2.getXAConnection();

    Connection conn1 = xacon1.getConnection();
    Connection conn2 = xacon2.getConnection();

    XAResource xars1 = xacon1.getXAResource();
    XAResource xars2 = xacon2.getXAResource();

    TbXid xid1 = new TbXid(formatID, gtid1,bq1 );
    TbXid xid2 = new TbXid(formatID, gtid2, bq2);

    try {
        xars1.start(xid1, TbXAResource.TBRTRANSLOOSE);

        PreparedStatement pstmt1;
        pstmt1 = conn1.prepareStatement(
                    "insert into author values (?,?)");

        pstmt1.setInt(1, 1);
        pstmt1.setString(2, "FOSCHIA");
        pstmt1.executeUpdate();

        pstmt1.setInt(1,2);
        pstmt1.setString(2, "AGNOS");
        pstmt1.executeUpdate();
```

```
            xars1.end(xid1, XAResource.TMSUCCESS);

            xars2.start(xid2, TbXAResource.TBRTRANSLOOSE);

            PreparedStatement pstmt2;
            pstmt2 = conn2.prepareStatement(
                        "insert into author values (?,?)");

            pstmt2.setInt(1, 3);
            pstmt2.setString(2, "JELLA");
            pstmt2.executeUpdate();

            pstmt2.setInt(1,4);
            pstmt2.setString(2, "THIPHILO");
            pstmt2.executeUpdate();

            xars2.end(xid2, XAResource.TMSUCCESS);

            pre1= xars1.prepare(xid1);
            assertEquals(pre1, XAResource.XA_OK);

            pre2 = xars2.prepare(xid2);
            assertEquals(pre2, XAResource.XA_OK);

            xars1.commit(xid1, false);
            xars2.commit(xid2, false);

            Statement stmt1 = conn1.createStatement();
            ResultSet rs1 =
                stmt1.executeQuery("select * from author");

            while (rs1.next())
            row_count ++;

            assertEquals(4, row_count);

            rs1.close(); rs1 = null;
            stmt1.close(); stmt1 = null;

            pstmt1.close(); conn1.close(); xacon1.close();
            pstmt2.close(); conn2.close(); xacon2.close();

            pstmt1= null; conn1= null; xacon1=null;
            pstmt2= null; conn2= null; xacon2=null;
        } catch (TbXAException e) {}
    }
}
```

4.4. TP-Monitor와 Tibero 연동

TP-Monitor(Transaction Processing Monitor)는 각종 프로토콜에서 동작하는 세션과 시스템 및 데이터베이스 사이의 최소 처리 단위인 트랜잭션을 감시하여 일관성 있게 보관 및 유지하는 역할을 하는 트랜잭션 관리 미들웨어이다. 본 절에서는 대표적인 상용 TP-Monitor인 Tmax와 Tuxedo를 Tibero RDBMS와 연동하는 예제를 제시한다.

4.4.1. Tmax 와 Tibero 연동

Tmax는 Transaction Maximization의 약어로 트랜잭션 처리 극대화를 의미한다. Tmax는 시스템의 분산 환경에서 이기종 컴퓨터 간의 트랜잭션 처리를 완벽히 보장하면서 부하를 분산시키고 에러 발생 시 적절한 조치를 담당하는 TP-Monitor이다. 트랜잭션의 특성을 지원하면서 사용자에게는 최적의 개발환경을 제공하고, 클라이언트/서버 환경에서 최적의 솔루션을 제공하며, 성능 개선은 물론 모든 장애에 완벽하게 대처한다.

Tmax는 분산 트랜잭션 프로세싱의 국제 표준인 X/Open DTP(Distributed Transaction Processing) 모델을 준수하고 국제 표준기구인 OSI(Open Systems Interconnection group)의 DTP 서비스에 대한 기능적 분산과 기능 구성 요소 간 API 및 시스템 인터페이스 정의에 따라 개발되었다. 또한, 분산 환경에서 이기종 간의 투명한 업무 처리 및 OLTP(On-Line Transaction Processing)를 지원하고 트랜잭션 처리의 ACID(Atomic, Consistent, Isolated, Durable: transaction properties) 특성을 만족하게 한다.

아래에서 소개할 Tmax 와 Tibero 연동 예제 프로그램을 테스트해보기 위해서는 Tmax와 Tibero가 정상적으로 설치되어 있어야 한다.

참고

1. 연동할 Tmax와 Tibero 가 다른 머신에 설치된 경우, Tmax가 설치된 머신에서 Tibero 클라이언트를 따로 설치하여 Tibero 서버에 정상적으로 접속할 수 있는 환경이 구축되어야 한다.

2. Tmax 설치 및 관리에 대한 자세한 내용은 "Tmax Installation Guide"나 "Tmax Administration Guide"를 참조한다. Tibero 로 설치 및 관리에 관한 자세한 내용은 "Tibero RDBMS 설치 안내서"와 "Tibero RDBMS 관리자 안내서"를 참고한다. .

여기서 소개하는 예제 프로그램은 Tmax를 설치할 때 인스톨러가 기본으로 제공하는 것이다. 클라이언트가 Tmax 서버를 통하여 Tibero DB를 접속하여 특정 테이블의 데이터를 조회, 추가, 변경, 삭제하는 작업을 한다.

테스트 환경과 프로그램에 사용된 각종 파일들은 다음과 같다.

- 테스트 환경

파일	설명
운영체제	Ubuntu Linux 2.6.32-24-server x86-64
쉘	bash

파일	설명
$TMAXDIR	Tmax 설치 디렉터리

- 프로그램 파일

파일	설명
sample.m	Tmax 환경 설정 파일 ($TMAXDIR/config)
tms_tbr.mk	Tibero용 TMS makefile($TMAXDIR/sample/server)
tbrtest.c	서버 프로그램 tbESQL/C 파일($TMAXDIR/sample/server)
tbrtest.h	서버 프로그램 헤더 파일($TMAXDIR/sample/server)
Makefile.tbr	서버 프로그램 makefile ($TMAXDIR/sample/server)
compile	서버 프로그램 빌드 스크립트 ($TMAXDIR/sample/server)
tbr_main.c	클라이언트 프로그램 파일($TMAXDIR/sample/client)
Makefile.c	클라이언트 프로그램 makefile ($TMAXDIR/sample/client)
compile	클라이언트 프로그램 빌드 스크립트 ($TMAXDIR/sample/client)

다음에 제시된 순서대로 따르면 Tmax와 Tibero가 연동하는 것을 확인할 수 있다.

1. Tmax 기본 환경 설정
2. TMS 컴파일
3. TMS 컴파일
4. 서버 프로그램 컴파일
5. 클라이언트 프로그램 컴파일
6. DB 테이블 생성
7. 예제 프로그램 실행

Tmax 기본 환경 설정

- Tmax 시스템 환경 파일 설정

  $TMAXDIR/config 디렉터리에 있는 sample.m은 Tmax를 기동시킬 때 필요한 각종 정보들이 들어있는 Tmax 시스템 환경 파일이다. ASCII 파일 형태로 작성하며, cfl 유틸리티로 컴파일 하여 이진 파일을 생성한다. 생성된 이진 파일은 Tmax 기동 및 종료할 때 참조된다.

  Tibero 서버와 연동하는 서비스를 활성화 시키기 위하여 sample.m 환경 파일의 SVRGROUP 절, SERVER 절, SERVICE 절 항목을 아래와 같이 수정한다.

```
### tms for Tibero ###
svg4            NODENAME = "integrity", DBNAME = TIBERO,
                OPENINFO = "TIBERO_XA:user=tibero,pwd=tmax,sestm=60,db=tibero",
                TMSNAME  = tms_tbr

### server for Tibero sample program ###
tbrtest         SVGNAME = svg4

### services for tbrtest ###
TBRINS          SVRNAME = tbrtest
TBRSEL          SVRNAME = tbrtest
TBRUPT          SVRNAME = tbrtest
TBRDEL          SVRNAME = tbrtest
```

SVRGROUP 절의 NODENAME은 Tmax 설치시에 hostname 값으로 자동 설정된다. DB 벤더를 구분하기 위한 DBNAME은 TIBERO로 설정되어 있다. OPENINFO는 XA 모드 설정을 위한 TIBERO_XA가 앞쪽에 쓰여 있고 그 뒤에 “4.2.2. xa_open 함수의 속성”이 나열되어 있다.TMSNAME에는 XA를 담당할 모듈 이름이 설정되어 있다.

SERVER 절에는 예제 서버 프로그램의 이름인 tbrtest와 예제 서버 프로그램이 제공하는 서비스 4가지가 설정되어 있다.

참고

1.현재 Tibero에서는 xa_open 함수의 속성 중에 LogDir 속성을 지원하지 않고 있다. TMSNAME에는 XA를 담당할 모듈 이름이 설정되어 있다.

2. Tmax 시스템 환경 파일 설정에 대한 자세한 내용은 "Tmax Administration Guide"를 참조한다.

- Tmax 시스템 환경 파일 컴파일

  수정한 sample.m 파일을 아래와 같은 명령으로 컴파일한다.

  ```
  cfl -i sample.m
  ```

  성공적으로 컴파일 된 후에는 다음과 같은 메시지가 출력된다.

  ```
  CFL is done successfully for node(<nodename>)
  ```

- 서비스 테이블 생성

  서비스 테이블은 각각의 Tmax 시스템내 서버 프로세스가 생성될 때 필요한 파일로서 각각의 프로세스들이 어떤 서비스를 처리하는지에 대한 정보가 담겨 있다. 아래와 같은 명령으로 서비스 테이블을 생성한다.

  ```
  gst
  ```

  성공적으로 처리되면 다음과 같은 메시지가 출력된다.

```
SVC tables are successfully generated GST is successfully done
```

- 구조체 정의 이진 파일 및 필드키 정의 이진 파일 생성

  서버나 클라언트 프로그램내에서 구조체나 필드키를 쓰는 경우 이와 관련된 이진 파일을 생성해 주어야 한다. 하지만 이 예제 프로그램에서는 구조체나 필드키를 사용하지 않으므로 그 과정을 생략한다.

TMS 컴파일

TMS(Transaction Management Server)는 Tmax 시스템의 구성요소로서 데이터베이스 관리 및 분산 트랜잭션 처리를 담당하는 프로세스이다. Tibero용 TMS를 컴파일하기 전에 Tibero 관련 환경 변수 TB_HOME, TB_SID, LD_LIBRARY_PATH, PATH 등이 제대로 설정되었는지 확인한다.

확인한 다음 아래와 같이 $TMAXDIR/sample/server 디렉터리로 이동하여 Tibero용 TMS makefile을 이용하여 TMS를 컴파일한다.

```
cd $TMAXDIR/sample/server
make –f tms_tbr.mk all
```

서버 프로그램 컴파일

$TMAXDIR/sample/server 디렉터리로 이동하여 실제적으로 서비스를 제공하는 서버 프로그램을 빌드 스크립트를 이용하여 컴파일한다.

```
cd $TMAXDIR/sample/server
./compile tbc tbrtest
```

클라이언트 프로그램 컴파일

$TMAXDIR/sample/client 디렉터리로 이동하여 서비스를 요청하는 클라이언트 프로그램을 빌드 스크립트를 이용하여 컴파일한다.

```
cd $TMAXDIR/sample/client
./compile c tbr_main
```

DB 테이블 생성

Tibero 서버에 tibero/tmax 계정으로 접속하여 아래와 같은 emp 테이블을 생성한다.

```
tbsql tibero/tmax

create table emp (
    empno number,
```

```
   ename char(16),
   job char(16),
   hiredate char(16),
   sal number
);
```

예제 프로그램 실행

- Tmax 시스템 기동

  다음과 명령으로 Tmax를 기동한다.

```
tmboot
```

  성공적으로 기동되면 다음과 같은 메시지가 출력된다.

```
TMBOOT for node(<nodename>) is starting:
Welcome to Tmax demo system: it will expire 2012/3/11
Today: 2012/1/13
        TMBOOT: TMM is starting: Fri Jan 13 14:18:31 2012
        TMBOOT: CLL is starting: Fri Jan 13 14:18:31 2012
        TMBOOT: CLH is starting: Fri Jan 13 14:18:31 2012
(I) CLH9991 Current Tmax Configuration: Number of client handler(MINCLH) = 1
                Supported maximum user per node = 680
                Supported maximum user per handler = 680 [CLH0125]
        TMBOOT: TLM(tlm) is starting: Fri Jan 13 14:18:31 2012
        TMBOOT: TMS(tms_tbr) is starting: Fri Jan 13 14:18:31 2012
(I) TMS0211 General Infomation : transaction recovery will be started [TMS0221]
(I) TMS0211 General Infomation : transaction recovery was completed [TMS0222]
        TMBOOT: TMS(tms_tbr) is starting: Fri Jan 13 14:18:31 2012
        TMBOOT: SVR(tbrtest) is starting: Fri Jan 13 14:18:31 2012
```

- 클라이언트 프로그램 실행

  클라이언트 프로그램의 명령 옵션은 다음과 같다.

```
Usage: ./tbr_main empno loop_cnt ins_flag upt_flag del_flag
flag : 1|0
```

  원하는 옵션을 선택하여 수행시키면 아래와 같은 결과가 출력된다.

```
./tbr_main 12 3 1 0 0

 LOOP COUNT = 1
>> INSERT : COMMIT TEST
[./tbr_main] [[TBRINS] emp Insert Success]
[./tbr_main] [[TBRSEL] emp Select Success [1]]
```

```
 LOOP COUNT = 2
>> INSERT : COMMIT TEST
[./tbr_main] [[TBRINS] emp Insert Success]
[./tbr_main] [[TBRSEL] emp Select Success [2]]

 LOOP COUNT = 3
>> INSERT : COMMIT TEST
[./tbr_main] [[TBRINS] emp Insert Success]
[./tbr_main] [[TBRSEL] emp Select Success [3]]
```

4.4.2. Tuxedo 와 Tibero 연동

Tuxedo는 분산 트랜잭션 처리를 위한 플랫폼으로서, C, C++ 및 COBOL로 작성된 소프트웨어를 위한 개방형의 분산 시스템을 토대로 메인프레임급 확장성 및 성능을 제공하며, 메인스트림 하드웨어를 토대로 메인프레임 애플리케이션을 “리호스팅” 할 수 있는 플랫폼이다. Tuxedo는 비용 효과적인 신뢰성과 초 당 수십만 건의 트랜잭션을 지원할 수 있는 탁월한 확장성을 제공하는 것은 물론, SOA와 같은 혁신적 아키텍처의 일부로서 기존 IT 자산의 수명을 연장함으로써 투자 보호의 이점을 제공한다. Oracle Tuxedo는 Oracle Fusion Middleware의 전략적 트랜잭션 처리 제품이다

아래에서 소개할 Tuxedo 와 Tibero 연동 예제 프로그램을 테스트해보기 위해서는 Tuxedo와 Tibero가 정상적으로 설치되어 있어야 한다.

참고

1. 연동할 Tmax와 Tibero 가 다른 머신에 설치된 경우, Tmax가 설치된 머신에서 Tibero 클라이언트를 따로 설치하여 Tibero 서버에 정상적으로 접속할 수 있는 환경이 구축되어야 한다.

2. Tuxedo 설치 및 관리에 대한 자세한 내용은 "Tuxedo Documenation" 참조한다. Tibero 로 설치 및 관리에 관한 자세한 내용은 "Tibero RDBMS 설치 안내서"와 "Tibero RDBMS 관리자 안내서"를 참고한다.

여기서 소개하는 예제 프로그램은 클라이언트가 Tuxedo 서버를 통하여 Tibero DB를 접속하여 특정 테이블의 데이터를 조회, 추가하는 작업을 한다. 테스트 환경과 프로그램에 사용된 각종 파일들은 다음과 같다. 편의상 테스트 서버는 tux_machine이라는 호스트네임을 가지고 있으며, Tibero와 Tuxedo는 각각 path/to/tibero와 /path/to/tuxedo에 설치되어 있다고 가정한다. 그리고 예제 프로그램 파일들이 작업 디렉터리 /path/to/example에 있다고 가정한다.

- 테스트 환경

구분	설명
운영체제	AIX
호스트 네임	tux_machine

구분	설명
쉘	bash
Tibero 설치 홈 디렉터리	/path/to/tibero
Tuxedo 설치 홈 디렉터리	/path/to/tuxedo
예제 프로그램 홈 디렉터리	/path/to/example

- 프로그램 파일

파일	설명
tb_tux.env	시스템 환경 변수 설정 파일
tb_tux.conf.m	Tuxedo 환경 설정 파일
tmax32.fld	필드 테이블 파일
trans_fml32.tbc	서버 프로그램 tbESQL/C 파일
builds.sh	서버 프로그램 빌드 스크립트
insert.c	INSERT 클라이언트 프로그램 파일
select.c	SELECT 클라이언트 프로그램 파일
buildc.sh	클라이언트 프로그램 빌드 스크립트
create_table.sql	테스트를 위한 DB 테이블 생성 파일
run.sh	Tuxedo 시스템 기동 스크립트

참고

본 절에서 설명하고 있는 기본 프로그램의 전체 소스 코드는 “Appendix B. Tibero와 Tuxedo 연동 예제 프로그램 소스 코드”를 참고한다.

다음에 제시된 순서대로 따르면 Tuxedo와 Tibero의 연동하는 것을 확인할 수 있다. 운영 체제나 시스템 환경에 따라서 예제 파일을 수정하여 테스트해야 한다.

1. 시스템 환경 변수 설정
2. Tuxedo 기본 환경 설정
3. TMS 컴파일
4. 서버 프로그램 컴파일
5. 클라이언트 프로그램 컴파일
6. DB 테이블 생성
7. 예제 프로그램 실행

시스템 환경 변수 설정

Tibero와 Tuxedo 연동 테스트를 하기 위해 필요한 시스템 환경 변수 설정은 아래와 같다.

- Tibero를 위한 기본 환경 변수 설정

```
export TB_HOME=/path/to/tibero
export TB_SID=tibero
export PATH=$TB_HOME/bin:$TB_HOME/client/bin:$TB_HOME/scripts:$PATH
export LD_LIBRARY_PATH=$TB_HOME/client/lib:$TB_HOME/lib:$LD_LIBRARY_PATH
export LIBPATH=$TB_HOME/client/lib:$TB_HOME/lib:$LIBPATH
```

- Tuxedo를 위한 기본 환경 변수 설정

```
export TUXDIR=/path/to/tuxedo
export JAVA_HOME=$TUXDIR/jre
export JVMLIBS=$JAVA_HOME/lib/amd64/server:$JAVA_HOME/jre/bin
export PATH=$TUXDIR/bin:$JAVA_HOME/bin:$PATH
export COBCPY=:$TUXDIR/cobinclude; export COBCPY
export COBOPT="-C ANS85 -C ALIGN=8 -C NOIBMCOMP -C TRUNC=ANSI -C OSEXT=cbl"
export SHLIB_PATH=$TUXDIR/lib:$JVMLIBS:$SHLIB_PATH
export LIBPATH=$TUXDIR/lib:$JVMLIBS:$LIBPATH
export LD_LIBRARY_PATH=$TUXDIR/lib:$JVMLIBS:$LD_LIBRARY_PATH
export WEBJAVADIR=$TUXDIR/udataobj/webgui/java
```

- Tuxedo를 위한 추가 환경 변수 설정

  연동 테스트를 위해서 추가 설정해 주어야 할 환경 변수들로서 상황에 맞게 설정해 준다.

```
export TUXCONFIG=/path/to/tuxedo/tuxconf
export FLDTBLDIR32=/path/to/tuxedo
export FIELDTBLS32=tmax32.fld
export TLOGDEVICE=/path/to/tuxedo/TLOG
export ULOGPFX=/path/to/tuxedo/ULOG
```

Tuxedo 기본 환경 설정

- Tuxedo 시스템 환경 파일 설정

  tb_tux.conf.m은 Tuxedo를 기동시킬 때 필요한 각종 정보들이 들어있는 Tuxedo 시스템 환경 파일이다. ASCII 파일 형태로 작성하며, tmloadcf 유틸리티로 컴파일하여 이진 파일을 생성한다. 생성된 이진 파일은 Tuxedo 기동 및 종료할 때 참조된다.

  Tibero 서버와 연동하는 서비스를 활성화시키기 위하여 tb_tux.conf.m 환경 파일의 RESOURCES 절, MACHINES 절, GROUPS 절, SERVERS 절, SERVICES 절 항목을 아래와 같이 설정하였다.

```
*RESOURCES
IPCKEY          68300
DOMAINID        tbrdomain
MASTER          tbrtest
MAXACCESSERS    10
MAXSERVERS      5
MAXSERVICES     10
MODEL           SHM
LDBAL           N

*MACHINES
DEFAULT:
                TUXDIR="path/to/tuxedo"
                APPDIR="path/to/example"
                TUXCONFIG="path/to/example/tuxconf"

tux_machine     LMID=tbrtest

*GROUPS
TBXA            LMID=tbrtest GRPNO=1
                TMSNAME=tms_tibero
                OPENINFO="TIBERO_XA:TIBERO_XA:user=sys,pwd=tibero,sestm=60,db=tibero"

*SERVERS
DEFAULT:
                CLOPT="-A -r"

trans_fml32     SRVGRP=TBXA  SRVID=1

*SERVICES
SELECT_FML32
INSERT_FML32
~
```

RESOURCES 절과 MACHINE 절은 일반적인 Tuxedo 시스템 환경 파일 설정처럼 한다.

MACHINE 절의 tux_machine은 서버의 호스트네임이므로 테스트 환경에 따라 다르게 설정 해주야 한다. LMID 와 MASTER는 임의로 tbrtest 라고 정하였고, DOMAINID로 임의로 tbrdomain 이라고 정하였으로 원하는 대로 설정해주면 된다.

GROUPS 절도 일반적인 Tuxedo 시스템 환경 파일 설정처럼 한다. TMSNAME에는 Tibero 서버와 XA 통신을 담당할 모듈의 이름을 설정한다. OPENINFO는 XA 모드 설정을 위한 TIBERO_XA:TIBERO_XA: 를 앞쪽에 쓰고 그 뒤에 “4.2.2. xa_open 함수의 속성”이 나열되어야 한다.

SERVERS 절에는 예제 서버 프로그램인 trans_fml32가 설정되어 있다. SERVICE 절에는 예제 서버 프로그램이 제공하는 서비스인 SELECT_FML32, INSERT_FML32가 설정되어 있다.

참고

1. 현재 Tibero에서는 xa_open 함수의 속성 중에 LogDir 속성을 지원하지 않고 있다.

2. Tuxedo 시스템 환경 파일 설정에 대한 자세한 설명은 "Tuxedo Documentation"를 참조한다.

- Tuxedo 시스템 환경 파일 컴파일 파일을 아래와 같은 명령으로 컴파일한다.

```
tmloadcf -y tb_tux.conf.m
```

- 필드 테이블 파일을 헤더 파일로 변환한다.

  서버나 클라언트 프로그램간에 데이터를 주고받을때 공용 자료 구조체 정의해서 쓴다.

  이번 연동 테스트에서는 FML(Field Manipulation Language)을 이용하여 아래 같은 필드 테이블 파일 tmax32.fld 을 새로 정의해서 쓴다.

```
#name          number          type            flag    comment
OUTPUT         302             string          0       -
EMPNO          901             long            0       -
ENAME          902             string          0       -
JOB            903             string          0       -
MGR            904             long            0       -
SAL            905             float           0       -
COMM           906             float           0       -
DEPTNO         907             long            0       -
```

  이 필드 테이블 파일 tmax32.fld로 부터 헤더 파일을 생성하는 방법은 다음과 같다.

```
mkfldhdr32 tmax32.fld
```

  이 결과 tmax32.fld.h이라는 헤더 파일이 생성되며, 예제 클라이언트 프로그램과 서버 프로그램에서 가져다 쓴다. 그 밖의 다른 구조체를 정의하여 파일을 만드는 방법은 "Tuxedo Documentation"을 참조한다.

TMS 컴파일

다음 아래와 같은 명령어를 이용하여 Tibero용 TMS를 컴파일한다.

```
buildtms -o $TUXDIR/bin/tms_tibero -v -r TIBERO_XA
```

서버 프로그램 컴파일

실제적으로 서비스를 제공하는 서버 프로그램을 아래와 같은 빌드 스크립트 builds.sh를 이용하여 컴파일 한다.

```
#### transaction server precompile ####
PRECOMP=$TB_HOME/client/bin/tbpc
PRECOMPFLAGS="UNSAFE_NULL=YES"
LIB=$TB_HOME/client/lib
INC=$TB_HOME/client/include
CFLAGS="-ltbcli -ltbxa -lm -lrt -lpthread -ltbertl -g "
CC=gcc
rm -f trans_fml32.c
$PRECOMP INCLUDE=$TUXDIR/include UNSAFE_NULL=YES INCLUDE=$INC $PRECOMPFLAGS ONAME=trans_fml3

#### transaction server build ####
buildserver -o trans_fml32 -v -f trans_fml32.c -s INSERT_FML32,SELECT_FML32 -r TIBERO_XA
```

참고

프리컴파일 옵션에 대한 자세한 설명은 "Tibero RDBMS tbESQL/C 안내서"를 참조한다.

클라이언트 프로그램 컴파일

insert와 select 서비스를 요청하는 클라이언트 프로그램을 아래와 같은 빌드 스크립트 buildc.sh를 이용하여 컴파일한다.

```
buildclient -o insert -v -f insert.c
buildclient -o select -v -f select.c
```

DB 테이블 생성

Tibero 서버에 tibero/tmax 계정으로 접속하여 아래와 같은 emp 테이블을 생성한다.

```
tbsql tibero/tmax

drop table emp;
CREATE TABLE emp (
    empno           NUMBER,
    ename           VARCHAR2(10),
    job             VARCHAR2(9),
    mgr             NUMBER(4),
    hiredate        DATE,
    sal             NUMBER(7,2),
```

```
    comm            NUMBER(7,2),
    deptno          NUMBER(2)
);
```

예제 프로그램 실행

- Tuxedo 시스템 기동

  다음 명령으로 기동한다.

  ```
  tmboot -y
  ```

  성공적으로 기동되면 다음과 같은 메시지가 출력된다.

  ```
  Booting all admin and server processes in /path/to/example/tuxconf
  INFO: Oracle Tuxedo, Version 10.3.0.0, 64-bit, Patch Level (none)

  Booting admin processes ...

  exec BBL -A :
          process id=3457104 ... Started.

  Booting server processes ...

  exec tms_tibero -A :
          process id=4046910 ... Started.
  exec tms_tibero -A :
          process id=9265576 ... Started.
  exec tms_tibero -A :
          process id=1863802 ... Started.
  exec trans_fml32 -A -r :
          process id=3719284 ... Started.
  5 processes started.
  ```

- 클라이언트 프로그램 실행

  emp 테이블에 Empyee 정보를 추가하고 조회하는 클라이언트 프로그램은 다음과 같이 실행시켜 보자.

  ```
  ./insert
  ****************************************
  | Employee Number : 1
  | Employee Name   : Kim
  | Employee Job    : Manager
  ```

```
****************************************
```

insert 프로그램을 실행시키고 Employee Number, Employ Name, Employ Job를 위와 같이 입력하면 Tuxedo 서버 프로그램을 통하여 Tibero 서버의 emp 테이블에 레코드가 추가된다.

```
./select
****************************************
| Employee Number : 1
****************************************

EMPNO: 1
ENAME: Kim
JOB: Manager
```

select 프로그램을 실행시키고 Employee Number를 위와 같이 입력하면 Tuxedo 서버 프로그램을 통하여 Tibero서버의 emp 테이블로부터 해당 레코드 정보를 가져와서 출력한다.

Appendix A. tbJDBC 예제 프로그램 소스 코드

A.1. JdbcTest.class

다음은 tbJDBC를 이용하여 **JdbcTest 클래스 파일**을 작성한 프로그램 소스 코드이다.

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;

public class JdbcTest
{
    Connection conn;

    public static void main(String[] args) throws Exception
    {
        JdbcTest test = new JdbcTest();

        test.connect();
        test.executeStatement();
        test.executePreparedStatement();
        test.executeCallableStatement();
        test.disconnect();
    }

    private void connect() throws SQLException, ClassNotFoundException
    {
        Class.forName("com.tmax.tibero.jdbc.TbDriver");

        conn = DriverManager.getConnection(
              "jdbc:tibero:thin:@localhost:6666:tibero",
              "tibero", "tmax");

        if (conn == null)
        {
            System.out.println("connection failure!");
            System.exit(-1);
```

```
        }
        System.out.println("Connection success!");
    }

    private void executeStatement() throws SQLException
    {
        String dropTable   = "drop table emp";
        String createTable = "create table emp (id   number, "+
                             " name varchar(20), salary number)";
        String InsertTable = "insert into emp values(1000, 'Park', 5000)";

        Statement stmt = conn.createStatement();

        try {
            stmt.executeUpdate(dropTable);
         } catch(SQLException e) {
           // if there is not the table
         }

         stmt.executeUpdate(createTable);
         stmt.executeUpdate(InsertTable);

         stmt.close();
    }

    private void executePreparedStatement() throws SQLException
    {
        PreparedStatement pstmt = conn
        .prepareStatement("select name from emp where id = ?");

        pstmt.setString(1, "1000");

        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            System.out.println(rs.getString(1));
        }
        pstmt.close();
    }

    private void executeCallableStatement() throws SQLException
    {
        String callSQL = " CREATE PROCEDURE testProc "+
                         " (ID_VAL IN NUMBER, SAL_VAL IN OUT NUMBER) as " +
                         " BEGIN" +
                         "   update emp" +
                         "      set salary = SAL_VAL" +
```

```
                        "    where id = ID_VAL;" +
                        "   select salary into SAL_VAL" +
                        "     from emp" +
                        "    where id = ID_VAl;" +
                        " END;";

        String dropProc = "DROP PROCEDURE testProc";

        Statement stmt = conn.createStatement();

        try {
            stmt.executeUpdate(dropProc);
        } catch(SQLException e) {
            // if there is not the procedure
        }

        stmt.executeUpdate(callSQL);

        CallableStatement cstmt = conn.prepareCall("{call testProc(?, ?)}");
        cstmt.setInt(1, 1000);
        cstmt.setInt(2, 7000);
        cstmt.registerOutParameter(2, Types.INTEGER);
        cstmt.executeUpdate();

        int salary = cstmt.getInt(2);
        System.out.println(salary);

        stmt.close();
        cstmt.close();
    }

    private void disconnect() throws SQLException
    {
        if (conn != null)
            conn.close();
    }
}
```

Appendix B. Tibero와 Tuxedo 연동 예제 프로그램 소스 코드

B.1. tb_tux.env

다음은 시스템 환경 변수 설정 파일이다.

```
# for tibero
export TB_HOME=/path/to/tibero
export TB_SID=tibero
export PATH=$TB_HOME/bin:$TB_HOME/client/bin:$TB_HOME/scripts:$PATH
export LD_LIBRARY_PATH=$TB_HOME/client/lib:$TB_HOME/lib:$LD_LIBRARY_PATH
export LIBPATH=$TB_HOME/client/lib:$TB_HOME/lib:$LIBPATH

# for tuxedo
export TUXDIR=/path/to/tuxedo
export JAVA_HOME=$TUXDIR/jre
export JVMLIBS=$JAVA_HOME/lib/amd64/server:$JAVA_HOME/jre/bin
export PATH=$TUXDIR/bin:$JAVA_HOME/bin:$PATH
export COBCPY=:$TUXDIR/cobinclude; export COBCPY
export COBOPT="-C ANS85 -C ALIGN=8 -C NOIBMCOMP -C TRUNC=ANSI -C OSEXT=cbl"
export SHLIB_PATH=$TUXDIR/lib:$JVMLIBS:$SHLIB_PATH
export LIBPATH=$TUXDIR/lib:$JVMLIBS:$LIBPATH
export LD_LIBRARY_PATH=$TUXDIR/lib:$JVMLIBS:$LD_LIBRARY_PATH
export WEBJAVADIR=$TUXDIR/udataobj/webgui/java

export TUXCONFIG=/path/to/tuxedo/tuxconf
export FLDTBLDIR32=/path/to/tuxedo
export FIELDTBLS32=tmax32.fld
export TLOGDEVICE=/path/to/tuxedo/TLOG
export ULOGPFX=/path/to/tuxedo/ULOG
```

B.2. tb_tux.conf.m

다음은 Tuxedo 환경 설정 파일이다.

```
*RESOURCES
IPCKEY          68300

DOMAINID        tbrdomain
```

```
MASTER          tbrtest
MAXACCESSERS    10
MAXSERVERS      5
MAXSERVICES     20
MODEL           SHM
LDBAL           N

*MACHINES
DEFAULT:
                TUXDIR="/data1/apmqam/oracle/tuxedo/tuxedo10gR3"
                APPDIR="/data1/apmqam/tibero_tuxedo_test"
                TUXCONFIG="/data1/apmqam/tibero_tuxedo_test/tuxconf"
                TLOGDEVICE="/data1/apmqam/tibero_tuxedo_test/TLOG"

tmaxi4          LMID=tbrtest

*GROUPS
TBXA            LMID=tbrtest GRPNO=1
                TMSNAME=tms_tibero
                OPENINFO="TIBERO_XA:TIBERO_XA:user=sys,pwd=tibero,sestm=60,db=tibero"

*SERVERS
DEFAULT:
                CLOPT="-A -r"

trans_fml32     SRVGRP=TBXA  SRVID=1

*SERVICES
SELECT_FML32
INSERT_FML32
```

B.3. tmax32.fld

다음은 필드 테이블 파일이다.

```
#name          number          type            flag    comment
OUTPUT         302             string          0       -
EMPNO          901             long            0       -
ENAME          902             string          0       -
JOB            903             string          0       -
MGR            904             long            0       -
SAL            905             float           0       -
COMM           906             float           0       -
DEPTNO         907             long            0       -
```

B.4. trans_fml32.tbc

서버 프로그램 tbESQL/C 파일

```
#include <stdio.h>
#include <atmi.h>
#include <userlog.h>
#include <Uunix.h>
#include <fml32.h>
#include "fml32.fld.h"
#include <tx.h>
#include "sqlca.h"

EXEC SQL include SQLCA.H;

EXEC SQL begin declare section;
    int h_empno;
    char h_ename[10];
    char h_job[10];
EXEC SQL end declare section;

void INSERT_FML32(rqst)
TPSVCINFO *rqst;
{
    FBFR32 *sndbuf;
    char msgbuf[256];
    FLDLEN32 flen;
    XID *xid;
    TXINFO info;
    char xidstring[1000];
    char str[100];

    int i;

    sndbuf = (FBFR32 *)rqst->data;

    tx_info(&info);
    xid = &(info.xid);

    memset( xidstring, 0x00, 1000);

    for( i = 0 ; i < xid->gtrid_length+xid->bqual_length ; i++ )
    {
        sprintf(xidstring+strlen(xidstring),"%0x\0",xid->data[i]);
    }

    Fprint32(sndbuf);
```

```
    memset( &h_empno, 0x00, sizeof ( h_empno ) );
    memset( h_ename, 0x00, sizeof ( h_ename ) );
    memset( h_job, 0x00, sizeof ( h_job ) );

    Fget32(sndbuf, EMPNO, 0, (char *)&h_empno, &flen);
    Fget32(sndbuf, ENAME, 0, (char *)h_ename, &flen);
    Fget32(sndbuf, JOB, 0, (char *)h_job, &flen);

    printf("SVR: EMPNO %d ENAME %s JOB %s\n", h_empno, h_ename, h_job);
    printf("SVR: INSERT_FML32 XID:%d.%d. %0x.%s - %s\n\n",
            xid->gtrid_length, xid->bqual_length,
            xid->formatID, xidstring, h_ename);

    EXEC SQL INSERT
    INTO emp( empno, ename, job )
    VALUES ( :h_empno, :h_ename, :h_job );

    if ( sqlca.sqlcode != 0 ){
        sprintf(msgbuf, "insert fail: sqlcode = %d(%s)\n",
                        sqlca.sqlcode, sqlca.sqlerrm.sqlerrmc);
        Fchg32(sndbuf, OUTPUT, 0, msgbuf, 0);
        tpreturn(TPFAIL, -1, (char *)sndbuf, 0, 0);
    }

    strcpy(msgbuf, "insert success!" );
    Fchg32(sndbuf, OUTPUT, 0, msgbuf, 0);
    tpreturn(TPSUCCESS, 0, rqst->data, strlen(rqst->data), 0);
}

void SELECT_FML32(rqst)
TPSVCINFO *rqst;
{
    FBFR32 *sndbuf;
    char msgbuf[256];
    FLDLEN32 flen;

    sndbuf = (FBFR32 *)rqst->data;
    Fprint32(sndbuf);
    Fget32(sndbuf, EMPNO, 0, (char *)&h_empno, &flen);

    EXEC SQL SELECT NVL(ename,' '), NVL(job,' ')
    INTO :h_ename,:h_job
    FROM emp
    WHERE empno = :h_empno;

    if ( sqlca.sqlcode != 0 ){
        sprintf(msgbuf, "select failed sqlcode = %d(%s)\n",
```

```
                        sqlca.sqlcode, sqlca.sqlerrm.sqlerrmc);
        Fchg32(sndbuf, OUTPUT, 0, msgbuf, 0);
        tpreturn(TPFAIL, -1, (char *)sndbuf, 0, 0);
    }

    printf("#### %d \n", h_empno);
    Fchg32(sndbuf, ENAME, 0,(char *)h_ename, 0);
    Fchg32(sndbuf, JOB, 0, (char *)h_job, 0);
    Fchg32(sndbuf, EMPNO, 0, (char *)&h_empno, 0);

    strcpy(msgbuf, "select success!" );
    Fchg32(sndbuf, OUTPUT, 0, msgbuf, 0);

    Fprint32(sndbuf);
    tpreturn(TPSUCCESS, 0,(char *)sndbuf, sizeof(sndbuf), 0);
}
```

B.5. builds.sh

다음은 서버 프로그램 빌드 스크립트이다.

```
#### transaction server precompile ####
PRECOMP=$TB_HOME/client/bin/tbpc
PRECOMPFLAGS="UNSAFE_NULL=YES"
LIB=$TB_HOME/client/lib
INC=$TB_HOME/client/include
CFLAGS="-ltbcli -ltbxa -lm -lrt -lpthread -ltbertl -g "
CC=gcc
rm -f trans_fml32.c
$PRECOMP INCLUDE=$TUXDIR/include UNSAFE_NULL=YES INCLUDE=$INC $PRECOMPFLAGS ONAME=trans_fml3

#### transaction server build ####
buildserver -o trans_fml32 -v -f trans_fml32.c -s INSERT_FML32,SELECT_FML32 -r TIBERO_XA
```

B.6. insert.c

다음은 INSERT 클라이언트 프로그램 파일이다.

```
#include <stdio.h>
#include <atmi.h>
#include <fml32.h>
#include <string.h>
#include "fml32.fld.h"

struct temp_str{
```

```
        int   empno;
        char  ename[11];
        char  job[10];
};
typedef struct temp_str *str;

main(argc, argv)
char *argv[];
{
    FBFR32 *sendbuf;
    FBFR32 *recvbuf;
    long recvlen = 0;
    int ret;
    str tmpbuf;
    char msgbuf[30];
    FLDLEN32 flen;

    tmpbuf = malloc(sizeof(struct temp_str));

    if (tpinit((TPINIT *) NULL) == -1) {
        fprintf(stderr, "Tpinit failed\n");
        exit(1);
    }

    if((sendbuf = (FBFR32 *) tpalloc("FML32", NULL, 0)) == NULL) {
        fprintf(stderr,"Error allocating send buffer\n");
        tpterm();
        exit(1);
    }

    if((recvbuf = (FBFR32 *) tpalloc("FML32", NULL, 0)) == NULL) {
        fprintf(stderr,"Error allocating receive buffer\n");
        tpfree((char *)sendbuf);
        tpterm();
        exit(1);
    }

    tmpbuf = malloc(sizeof(struct temp_str));

    printf("\n****************************************\n");
    printf( "|  Employee Number : " ); scanf ( "%d", &tmpbuf->empno );
    printf( "|  Employee Name   : " ); scanf ( "%s", tmpbuf->ename );
    printf( "|  Employee Job    : " ); scanf ( "%s", tmpbuf->job );
    printf("****************************************\n\n");

    tpbegin(10, 0);
    Fchg32(sendbuf,EMPNO,0,(char *)&tmpbuf->empno,0);
```

```
    Fchg32(sendbuf,ENAME,0,(char *)tmpbuf->ename,0);
    Fchg32(sendbuf,JOB,0,(char *)tmpbuf->job,0);

    ret = tpcall("INSERT_FML32", (char *)sendbuf, 0, (char **)&recvbuf,
                 &recvlen, (long)0);
    if(ret == -1) {
        fprintf(stderr, "tperrno = %d (%s) \n", tperrno, tpstrerror(tperrno));
        tpfree((char *)sendbuf);
        tpfree((char *)recvbuf);
        tpterm();
        exit(1);
    }

    flen = sizeof( msgbuf );
    Fget32(recvbuf, OUTPUT, 0, (char *)msgbuf, &flen);
    printf("%s\n", msgbuf);

    tpcommit(0);

    free(tmpbuf);
    tpfree((char *)sendbuf);
    tpterm();
}
```

B.7. select.c

다음은 SELECT 클라이언트 프로그램 파일이다.

```
#include <stdio.h>
#include <atmi.h>
#include <fml32.h>
#include <string.h>
#include "fml32.fld.h"


main(argc, argv)
char *argv[];
{
    FBFR32 *sendbuf;
    FBFR32 *recvbuf;
    long recvlen;
    int ret;
    int empno;
    FLDLEN32 flen;
    int h_empno;
    char h_ename[10];
```

```
    char h_job[10];
    char msgbuf[256];

    if (tpinit((TPINIT *) NULL) == -1) {
        fprintf(stderr, "Tpinit failed\n");
        exit(1);
    }

    if((sendbuf = (FBFR32 *) tpalloc("FML32", NULL, 0)) == NULL) {
        fprintf(stderr,"Error allocating send buffer\n");
        tpterm();
        exit(1);
    }

    if((recvbuf = (FBFR32 *) tpalloc("FML32", NULL, 0)) == NULL) {
        fprintf(stderr,"Error allocating receive buffer\n");
        tpfree((char *)sendbuf);
        tpterm();
        exit(1);
    }

    printf("\n*****************************************\n");
    printf( "|  Employee Number : " ); scanf ( "%d", &empno );
    printf("*****************************************\n\n");

    Fchg32(sendbuf,EMPNO,0,(char *)&empno,0);

    ret = tpcall("SELECT_FML32", (char *)sendbuf, 0, (char **)&recvbuf,
                 &recvlen, (long)0);
    if(ret == -1) {
        fprintf(stderr, "tperrno = %d (%s)\n", tperrno, tpstrerror(tperrno));
        flen = sizeof( msgbuf );
        Fget32(recvbuf, OUTPUT, 0, (char *)msgbuf, &flen);
        printf("error msg: %s\n", msgbuf);
        tpfree((char *)sendbuf);
        tpfree((char *)recvbuf);
        tpterm();
        exit(1);
    }

    flen = sizeof(msgbuf);
    Fget32(recvbuf, OUTPUT, 0, (char *)msgbuf, &flen);
    printf("%s\n", msgbuf);

    flen = sizeof(&h_empno);
    Fget32(recvbuf, EMPNO, 0, (char *)&h_empno, &flen);
    printf("EMPNO; %d \n", h_empno);
```

```
    flen = sizeof(h_ename);
    Fget32(recvbuf, ENAME, 0, (char *)h_ename, &flen);
    printf("ENAME: %s \n", h_ename);
    flen = sizeof(h_job);
    Fget32(recvbuf, JOB, 0, (char *)h_job, &flen);
    printf("JOB: %s \n", h_job);

    tpfree((char *)recvbuf);
    tpfree((char *)sendbuf);
    tpterm();
}
~
```

B.8. buildc.sh

다음은 클라이언트 프로그램 빌드 스크립트이다.

```
buildclient -o insert -v -f insert.c
buildclient -o select -v -f select.c
```

B.9. create_table.sql

다음은 SELECT 클라이언트 프로그램 파일이다.

```
drop table emp;
CREATE TABLE emp (
    empno           NUMBER,
    ename           VARCHAR2(10),
    job             VARCHAR2(9),
    mgr             NUMBER(4),
    hiredate        DATE,
    sal             NUMBER(7,2),
    comm            NUMBER(7,2),
    deptno          NUMBER(2)
);
~
```

B.10. run.sh

다음은 Tuxedo 시스템 기동 스크립트이다.

```
#!/bin/sh

tmshutdown -y
```

```
rm ULOG* > /dev/null 2>&1
rm xa* > /dev/null 2>&1

tmloadcf -y tb_tux.conf.m
rm /data1/apmqam/tibero_tuxedo_test/TLOG* > /dev/null 2>&1
rm /data1/apmqam/tibero_tuxedo_test/ULOG* > /dev/null 2>&1

tmadmin -c << EOF
crdl -z /data1/apmqam/tibero_tuxedo_test/TLOG -b 1000
q
EOF

tmboot -y
```

색인

V

X

ㄱ

ㄴ

ㄷ

ㅁ

ㅂ

ㅅ

ㅇ

ㅈ

ㅌ