

- [JDBC User's Manual](#)
 - [서문](#)
 - [이 매뉴얼에 대하여](#)
 - [1.JDBC 시작하기](#)
 - [JDBC 드라이버 설치](#)
 - [데이터베이스에 연결하기](#)
 - [연결 정보](#)
 - [Statement와 ResultSet 다루기](#)
 - [JDBC Connection Failover](#)
 - [2.기본 기능](#)
 - [IPv6 접속](#)
 - [Statement, PreparedStatement 및 CallableStatement](#)
 - [내셔널 캐릭터 셋 사용](#)
 - [3.고급 기능](#)
 - [자동 생성 키](#)
 - [타임아웃](#)
 - [DataSource](#)
 - [Connection Pool](#)
 - [Multiple ResultSet](#)
 - [JDBC와 Failover](#)
 - [JDBC Escapes](#)
 - [ResultSet 사용하기](#)
 - [Atomic Batch](#)
 - [Date, Time, Timestamp](#)
 - [GEOMETRY](#)
 - [LOB](#)
 - [Autocommit 제어](#)
 - [BIT, VARBIT](#)
 - [JDBC 로깅](#)
 - [Hibernate](#)
 - [Sharding](#)
 - [SQL Plan](#)
 - [4.Tips & Recommendation](#)
 - [성능을 위한 팁](#)
 - [5.에러 메시지](#)
 - [SQL States](#)
 - [A.부록: 데이터 타입 매핑](#)
 - [데이터 타입 매핑](#)
 - [Java 데이터형을 데이터베이스 데이터형으로 변환하기](#)
 - [데이터베이스 데이터형을 Java 데이터형으로 변환하기](#)

Altibase® Application Development

JDBC User's Manual



Altibase Application Development JDBC User's Manual

Release 7.1

Copyright © 2001~ 2019 Altibase Corp. All Rights Reserved.

본 문서의 저작권은 ㈜알티베이스에 있습니다. 이 문서에 대하여 당사의 동의 없이 무단으로 복제 또는 전용할 수 없습니다.

㈜알티베이스

08378 서울시 구로구 디지털로 306 대륭포스트타워II 10층

전화: 02-2082-1114 팩스: 02-2082-1099

고객서비스포털: <http://support.altibase.com>

homepage: <http://www.altibase.com>

서문

이 매뉴얼에 대하여

이 매뉴얼은 Altibase가 제공하는 JDBC 드라이버의 사용법에 대해 설명한다. Altibase의 JDBC 드라이버는 JDBC 사양을 대부분 준수하나, 경우에 따라서 사양에서 벗어난 방식으로 동작한다. JDBC를 이용해서 애플리케이션을 작성하기 전에, 본 매뉴얼을 참고하여 JDBC 사양과 다른 부분에 대한 지식을 습득할 것을 권고한다.

대상 사용자

이 매뉴얼은 다음과 같은 Altibase 사용자를 대상으로 작성되었다.

- 데이터베이스 관리자
- 성능 관리자
- 데이터베이스 사용자
- 응용 프로그램 개발자
- 기술지원부

다음과 같은 배경 지식을 가지고 이 매뉴얼을 읽는 것이 좋다.

- 자바 프로그래밍 언어
- SQL
- Stored Procedure
- Altibase에 대한 이해

소프트웨어 환경

이 매뉴얼은 데이터베이스 서버로 Altibase 버전 7.1을 사용한다는 가정 하에 작성되었다.

이 매뉴얼의 구성

이 매뉴얼은 다음과 같이 구성되어 있다.

- 제 1장 시작하기
이 장에서는 Altibase의 JDBC 드라이버를 이용하는 기본적인 방법을 기술한다.
- 제 2 장 기본 기능
이 장에서는 Altibase의 JDBC 드라이버를 사용해서 데이터베이스의 객체를 다루는 기본적인 방법을 설명한다.
- 제 3 장 고급 기능
이 장에서는 Altibase의 JDBC 드라이버가 제공하는 보다 향상된 기능과 그 사용법을 설명한다.
- 제 4 장 Tips & Recommendation
이 장은 Altibase의 JDBC 드라이버를 효율적으로 사용하기 위한 방법을 제시한다.
- 제 5장 에러 메시지
이 장은 Altibase의 JDBC 드라이버를 사용하면서 발생할 수 있는 SQL State를 기술한다.
- 부록 A. 데이터 타입 매핑
Altibase의 데이터 타입과 JDBC 표준 데이터 타입, Java 데이터 타입간에 호환 여부를 기술한다.

문서화 규칙



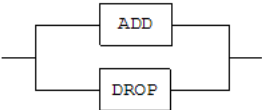
이 절에서는 이 매뉴얼에서 사용하는 규칙에 대해 설명한다. 이 규칙을 이해하면 이 매뉴얼과 설명서 세트의 다른 매뉴얼에서 정보를 쉽게 찾을 수 있다.

여기서 설명하는 규칙은 다음과 같다.

- 구문 다이어그램
- 샘플 코드 규칙

구문 다이어그램

이 매뉴얼에서는 다음 구성 요소로 구축된 다이어그램을 사용하여, 명령문의 구문을 설명한다.

구성 요소	의미
	명령문이 시작한다. 완전한 명령문이 아닌 구문 요소는 화살표로 시작한다.
	명령문이 다음 라인에 계속된다. 완전한 명령문이 아닌 구문 요소는 이 기호로 종료한다.
	명령문이 이전 라인으로부터 계속된다. 완전한 명령문이 아닌 구문 요소는 이 기호로 시작한다.
	명령문이 종료한다.
	필수 항목
	선택적 항목
	선택사항이 있는 필수 항목. 한 항목만 제공해야 한다.
	선택사항이 있는 선택적 항목
	선택적 항목. 여러 항목이 허용된다. 각 반복 앞부분에 콤마가 와야 한다.

샘플 코드 규칙

코드 예제는 SQL, Stored Procedure, iSQL 또는 다른 명령 라인 구문들을 예를 들어 설명한다.

아래 테이블은 코드 예제에서 사용된 인쇄 규칙에 대해 설명한다.

규칙	의미	예제
[]	선택 항목을 표시	VARCHAR [(size)] [[FIXED] VARIABLE]
{ }	필수 항목 표시. 반드시 하나 이상을 선택해야 되는 표시	{ ENABLE DISABLE COMPILE }
	선택 또는 필수 항목 표시의 인자 구분 표시	{ ENABLE DISABLE COMPILE } [ENABLE DISABLE COMPILE]
...	그 이전 인자의 반복 표시 예제 코드들의 생략되는 것을 표시	SQL> SELECT ename FROM employee; ENAME ----- SWNO HJNO HSCHOI ... 20 rows selected.
그 밖에 기호	위에서 보여진 기호 이 외에 기호들	EXEC :p1 := 1; acc NUMBER(11,2);
기울임 꼴	구문 요소에서 사용자가 지정해야 하는 변수, 특수한 값을 제공해야만 하는 위치	SELECT * FROM table_name; CONNECT userID/password;

규칙	의미	예제
소문자	사용자가 제공하는 프로그램의 요소들, 예를 들어 테이블 이름, 칼럼 이름, 파일 이름 등	SELECT ename FROM employee;
대문자	시스템에서 제공하는 요소들 또는 구문에 나타나는 키워드	DESC SYSTEM_.SYS_INDICES_;

관련 자료

자세한 정보를 위하여 다음 문서 목록을 참조하기 바란다.

- Administrator’s Manual
- Replication Manual
- Spatial SQL Reference

Altibase는 여러분의 의견을 환영합니다.

이 매뉴얼에 대한 여러분의 의견을 보내주시기 바랍니다. 사용자의 의견은 다음 버전의 매뉴얼을 작성하는데 많은 도움이 됩니다. 보내실 때에는 아래 내용과 함께 고객센터포털(<http://support.altibase.com/kr/>)로 보내주시기 바랍니다.

- 사용 중인 매뉴얼의 이름과 버전
- 매뉴얼에 대한 의견
- 사용자의 성함, 주소, 전화번호

이 외에도 Altibase 기술지원 설명서의 오류와 누락된 부분 및 기타 기술적인 문제들에 대해서 이 주소로 보내주시면 정성껏 처리하겠습니다. 또한, 기술적인 부분과 관련하여 즉각적인 도움이 필요한 경우에도 고객센터포털을 통해 서비스를 요청하시기 바랍니다.

여러분의 의견에 항상 감사드립니다.

1.JDBC 시작하기

이 장에서는 Altibase의 JDBC 드라이버를 이용하는 기본적인 방법을 기술한다.

JDBC 드라이버 설치

Altibase 홈페이지 (www.altibase.com)에서 다운로드 받은 Altibase 패키지를 다운로드 하여 설치한다.

Altibase JDBC 드라이버는 패키지를 설치한 후, \$ALTIBASE_HOME/lib 디렉토리에서 찾을 수 있다.

버전 호환성

Altibase 7.1 JDBC 드라이버는 Type 4 pure Java JDBC 드라이버로써, JDBC 3.0 스펙을 준수한다. 또한, JDK 1.5 이상에서 정상적으로 동작한다.

JDBC 드라이버 버전 확인

설치된 JDBC 드라이버의 버전과 드라이버가 컴파일된 JDK 버전을 아래와 같이 확인할 수 있다.

```
$ java -jar $ALTIBASE_HOME/lib/Altibase.jar
JDBC Driver Info : Altibase 7.1.0.0.0 with CMP 7.1.3 for JDBC 3.0 compiled with JDK 5
```

CLASSPATH 설정

Altibase JDBC를 사용하려면 Altibase JDBC 드라이버를 CLASSPATH 환경변수에 추가해야 한다.

Altibase는 로깅 기능을 지원하지 않는 Altibase.jar 파일과 지원하는 Altibase_t.jar 파일을 함께 제공한다.

ex) 유닉스 환경에서 bash 셸을 사용하는 경우

```
$ export CLASSPATH=$ALTIBASE_HOME/lib/Altibase.jar:.$CLASSPATH
```

LD_LIBRARY_PATH 설정

비동기적 prefetch의 auto-tuning 기능을 사용할 경우에는 JNI 모듈이 필요하며 [libaltijext.so](#) 파일이 위치한 디렉토리를 LD_LIBRARY_PATH 환경변수에 추가하여야 한다. JNI 모듈을 로딩하는데 실패하더라도 auto-tuning 이외의 기능은 정상 동작한다.

ex) 유닉스 환경에서 bash 셸을 사용하는 경우

```
$ export LD_LIBRARY_PATH=$ALTIBASE_HOME/lib:.$LD_LIBRARY_PATH
```

데이터베이스에 연결하기

이 절에서는 JDBC로 Altibase 서버에 연결하는 기본적인 방법을 프로그램 코드로 설명한다.

드라이버 로딩

Altibase JDBC 드라이버의 Driver 클래스를 로딩하고, 드라이버를 찾아오는 방법이다.

1. DriverManager에 Altibase JDBC 드라이버를 등록한다.

```
Class.forName("Altibase.jdbc.driver.AltibaseDriver");
```

2. 연결 URL을 사용해서 DriverManager에서 드라이버를 획득할 수 있다.

```
String sURL = "jdbc:Altibase://localhost:20300/mydb";  
Driver sDriver = DriverManager.getDriver( sURL );
```

연결 정보 설정

Properties 객체를 사용해서 다수의 연결 정보를 드라이버에 전달할 수 있다. 다음은 Properties 객체에 연결 속성을 셋팅하는 코드 예제이다.

```
Properties sProps = new Properties();  
sProps.put("user", "SYS");  
sProps.put("password", "MANAGER");
```

데이터베이스에 접속

연결 URL과 연결 정보를 사용해서 데이터베이스에 접속할 수 있다. 다음은 데이터베이스에 연결하고 Connection 객체를 획득하는 코드 예제이다.

```
Connection sCon = sDriver.connect(sURL, sProps);
```

위의 connect 메소드의 인자로 사용되는 연결 URL의 형식은 아래와 같다.

```
jdbc:Altibase://server_ip:server_port/dbname
```

예제

```

import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverManager;
import java.util.Properties;
public class ConnectionExample
{
    public static void main(String[] args) throws Exception
    {
        String sURL = "jdbc:Altibase://localhost:20300/mydb";
        Properties sProps = new Properties();
        sProps.put("user", "SYS");
        sProps.put("password", "MANAGER");
        Class.forName("Altibase.jdbc.driver.AltibaseDriver");
        Driver sDriver = DriverManager.getDriver(sURL);
        Connection sCon = sDriver.connect(sURL, sProps);
    }
}

```

컴파일과 실행

아래와 같이 JDBC 응용프로그램을 컴파일하고 실행할 수 있다.

```

$ javac ConnectionExample.java
$ java ConnectionExample

```

연결 정보

이 절에서는 Altibase에 접속할 때 사용 가능한 연결 속성들의 정보를 기술하고, 연결 속성을 설정하는 방법에 대해 설명한다.

연결 속성 설정하기

데이터베이스 접속에 필요한 연결 속성을 `Properties` 객체에 설정하거나 연결 URL에 명시할 수 있다.

연결 URL 사용하기

연결 URL 끝에 "?"를 붙이고 그 뒤에 "*key=value*" 형식으로 프로퍼티 값을 지정할 수 있다. 여러 개의 프로퍼티를 입력할 때는 "&"로 연결하면 된다.

다음은 연결 URL의 예제이다.

```
"jdbc:Altibase://localhost:20300/mydb?fetch_enough=0&time_zone=DB_TZ"
```

`Properties` 객체 사용하기

`Properties` 객체를 생성하고 키와 값을 입력한 다음, 연결 정보로 이용하면 된다.

아래는 `Properties` 객체를 사용하는 예제이다.

```

Properties sProps = new Properties();
sProps.put("fetch_enough", "30");
sProps.put("time_zone", "DB_TZ");

...

Connection sCon = DriverManager.getConnection( sURL, sProps );

```

연결 속성 정보

Altibase에 접속할 때 사용 가능한 연결 속성에 대해 기술한다. 각 속성에 대한 기술에는 다음의 항목들이 포함된다.

- 기본값: 명시하지 않을 경우 기본적으로 사용되는 값
- 값의 범위: 지정 가능한 값
- 필수 여부: 반드시 지정해야 하는지 여부
- 설정 범위: 설정한 속성이 시스템 전체에 영향을 미치는지 또는 해당 세션에만 영향을 미치는지 여부

- 설명: 속성에 대한 설명

alternateservers

기본값	
값의 범위	[host_name:port_number[/dbname][, host_name:port_number[/dbname]]*
필수 여부	No
설정 범위	
설명	Connection Failover 발생 시 접속할 수 있는 서버들의 리스트이다. 사용법은 3장의 "JDBC와 Failover" 절을 참고한다.

app_info

기본값	
값의 범위	임의의 문자열
필수 여부	No
설정 범위	세션
설명	V\$SESSION의 CLIENT_APP_INFO 칼럼에 저장될 문자열을 지정한다.

auto_commit

기본값	true
값의 범위	[true false]
필수 여부	No
설정 범위	세션
설명	구문의 수행이 완료될 때 트랜잭션이 자동으로 커밋될 지 여부를 지정한다.

defer_pre pares

기본값	off
값의 범위	[on off]
필수 여부	No
설정 범위	세션
설명	<p>PrepareStatement가 호출될 때 서버와의 통신을 보류할지 여부(ON, OFF)를 지정할 수 있다. 이 속성이 ON이면, PrepareStatement가 호출이 되더라도 Execute 함수가 호출될 때까지 prepare 요청이 서버로 전송되지 않는다. 그러나 이 속성이 OFF이면, PrepareStatement가 호출될 때 prepare 요청이 즉시 서버로 전송된다.</p> <p>단 PreparedStatement () 뒤에 다음의 메소드들이 호출되면, prepare 요청이 즉시 서버로 전송된다.</p> <ul style="list-style-type: none"> • getMetData • getParameterMetaData • setObject(int, Object, int) <p>또한 DBCP의 statement pool이 활성화되어 있을 경우 충돌이 발생할 수 있기 때문에 deferred 옵션이 켜져 있을 경우에는 statement pool 옵션을 꺼야 한다.</p>

ciphersuite_list

기본값	JRE가 지원하는 모든 cipher suite list를 참조한다.
값의 범위	임의의 문자열
필수 여부	No
설정 범위	N/A
설명	서버에 SSL 통신을 위해 사용할 암호 알고리즘 목록이다.

clientside_auto_commit

기본값	off
값의 범위	[on off]
필수 여부	No
설정 범위	세션
설명	<p>Autocommit의 동작을 Altibase 서버가 제어할 것인지, JDBC 드라이버가 제어할 것인지를 설정한다.</p> <p>auto_commit 속성이 true이거나 auto_commit 속성이 지정되지 않은 경우에만 이 속성의 설정 값이 적용된다.</p> <ul style="list-style-type: none"> on: JDBC 드라이버가 자동커밋 제어 off: Altibase 서버가 자동커밋 제어 <p>자세한 설명은 3장의 "Autocommit 제어" 절을 참고한다.</p>

connectionretrycount

기본값	0
값의 범위	Unsigned Integer 범위내의 숫자값 [1 - 231]
필수 여부	No
설정 범위	
설명	<p>Connection Failover 발생 시, 타 서버로 재접속을 시도하는 횟수를 지정한다.</p> <p>이 값이 1일 경우 타서버로 재시도 횟수는 한 번이므로, 총 두 번의 접속 시도를 한다.</p> <p>사용법은 3장의 "JDBC와 Failover" 절을 참고한다.</p>

connectionretrydelay

기본값	0
값의 범위	Unsigned Integer 범위내의 숫자값 [1 - 231]
필수 여부	No
설정 범위	
설명	<p>Connection Failover 발생 시, ConnectionRetryCount가 설정되어 있는 경우, 다른 서버로 재접속을 시도하는 대기 시간을 설정한다.</p> <p>단위는 초(second)이다.</p> <p>사용법은 3장의 "JDBC와 Failover" 절을 참고한다.</p>

database

기본값	mydb
값의 범위	데이터베이스 이름
필수 여부	No

기본값	mydb
설정 범위	N/A
설명	접속을 시도할 Altibase 서버에 생성한 데이터베이스의 이름

datasource

기본값	
값의 범위	[0 - 65535]
필수 여부	Datasource 사용시 필수
설정 범위	N/A
설명	DataSource 이름

date_format

기본값	ALTIBASE_DATE_FORMAT 환경 변수에 설정된 값. 환경 변수가 설정되어 있지 않으면, 날짜 타입의 입출력 형식은 서버의 설정 값을 따른다.
값의 범위	날짜 형식의 문자열
필수 여부	No
설정 범위	시스템
설명	DATE 타입의 입/출력 포맷을 지정하는 속성이다. 포맷이 설정되면, 클라이언트에서 해당 포맷을 따르지 않은 데이터를 입력할 경우, DATE 타입으로 간주하지 않고, 에러를 반환한다.

ddl_timeout

기본값	0
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	DDL문 수행 시간의 한계값을 설정한다. 수행 시간이 이 시간을 넘어가면 구문의 수행이 취소된다. 단위는 초(sec)이다. 이 값이 0이면 무한대를 의미한다.

description

기본값	
값의 범위	임의의 문자열
필수 여부	No
설정 범위	N/A
설명	DataSource의 Description 부분

fetch_async

기본값	Off
값의 범위	[off preferred]
필수 여부	No
설정 범위	

기본값	Off
설명	비동기적으로 prefetch를 수행하여 fetch 성능을 향상시킨다. off: 동기적으로 prefetch를 수행한다.(기본값) preferred: 비동기적으로 prefetch를 수행한다. 명령문마다 비동기 prefetch를 설정할 수 있지만 접속 당 하나의 명령문만 prefetch가 비동기적으로 수행된다.

fetch_auto_tuning

기본값	on (Linux 인 경우) 이외의 경우는 off
값의 범위	[on off]
필수 여부	No
설정 범위	
설명	비동기적으로 prefetch 를 수행할 경우, 네트워크 상태에 따라 auto-tuning 할 것인지 여부를 지정한다. Auto-tuning이란 네트워크 상태에 따라 prefetch row 개수를 자동으로 조절해주는 기능이다. 이 기능은 리눅스에서만 사용할 수 있다. off: auto-tuning 기능을 사용하지 않는다. (리눅스 이외의 OS 에서 기본값) on: auto-tuning 기능을 사용한다. (리눅스에서 기본값)

fetch_enough

기본값	0
값의 범위	[0 - 2147483647]
필수 여부	No
설정 범위	세션
설명	현재 세션의 FetchSize를 설정한다. 한 번의 Fetch 연산으로 서버에서 한 번에 가져올 행의 개수를 의미한다. 이 값이 0이면, JDBC 드라이버는 한 네트워크 패킷에 담을 수 있는 최대 크기만큼 서버로부터 데이터를 가지고 온다.

fetch_timeout

기본값	60
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	SELECT문 수행 시간의 한계값을 설정한다. 질의 수행 시간이 설정된 시간을 넘어가면 질의는 자동으로 종료된다. 단위는 초(sec)이다. 이 값이 0이면 무한대를 의미한다.

idle_timeout

기본값	0
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	아무런 작업도 하지 않은 채로 Connection이 연결을 유지하는 시간의 한계값을 설정한다. 이 시간을 넘어가면 접속은 자동으로 해제된다. 단위는 초(sec)이다. 이 값이 0이면 무한대를 의미한다.

isolation_level

기본값	
값의 범위	[2 4 8]
필수 여부	No
설정 범위	시스템
설명	2 : TRANSACTION_READ_COMMITTED 4 : TRANSACTION_REPEATABLE_READ 8: TRANSACTION_SERIALIZABLE

keystore_password

기본값	N/A
값의 범위	임의의 문자열
필수 여부	No
설정 범위	N/A
설명	keystore_url에 비밀번호를 지정한다.

keystore_type

기본값	JKS
값의 범위	[JKS, JCEKS, PKCS12 외]
필수 여부	No
설정 범위	N/A
설명	keystore_url의 keystore 타입을 설정한다.

keystore_url

기본값	N/A
값의 범위	임의의 문자열
필수 여부	No
설정 범위	N/A
설명	KeyStore의 경로를 지정한다. KeyStore는 개인 인증서와 공개 인증서를 가지고 있다.

lob_cache_threshold

기본값	8192
값의 범위	[0 - 524288]
필수 여부	No
설정 범위	세션
설명	클라이언트에 캐시할 수 있는 LOB 데이터의 최대 크기를 설정한다.

login_timeout

기본값	
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션

기본값	
설명	로그인 대기 최대 시간을 설정한다. 자세한 내용은 3장의 "타임아웃" 절을 참고한다.

max_statements_per_session

기본값	
값의 범위	Signed Short 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	한 세션에서 실행 가능한 구문(statement)의 최대 개수를 지정한다. 이 값이 0이면 무한대를 의미한다.

ncharliteralreplace

기본값	false
값의 범위	[true false]
필수 여부	No
설정 범위	세션
설명	SQL문 내에 NCHAR 문자열(리터럴)의 존재 여부를 클라이언트에서 검사할지를 지정한다.

prefer_ipv6

기본값	false
값의 범위	[true false]
필수 여부	No
설정 범위	
설명	IPv6 주소를 그대로 사용할 것인지, IPv4로 변환해서 사용할 것인지를 지정한다. 자세한 설명은 "IPv6 접속" 절을 참고하라.

password

기본값	
값의 범위	
필수 여부	Yes
설정 범위	N/A
설명	사용자 ID의 비밀번호

port

기본값	20300
값의 범위	[0 - 65535]
필수 여부	No
설정 범위	N/A
설명	접속을 시도할 Altibase 서버의 포트번호를 지정한다. ssl_enable의 값이 false이면 20300, true이면 20443이 사용된다.

privilege

기본값	
값의 범위	[normal sysdba]
필수 여부	No
설정 범위	N/A
설명	접속 모드 normal: 일반 모드 sysdba: DBA 모드

query_timeout

기본값	600
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	질의 수행 시간의 한계값을 설정한다. 수행 시간을 넘어서는 질의는 자동으로 종료된다. 단위는 초(sec)이다. 이 값이 0이면 무한대를 의미한다.

remove_redundant_transmission

기본값	0
값의 범위	[0 1]
필수 여부	No
설정 범위	세션
설명	CHAR, VARCHAR, NCHAR, NVARCHAR 타입의 문자열에 중복 데이터 압축 방식을 사용할 것인지를 설정한다.

response_timeout

기본값	
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	응답 대기 최대 시간을 설정한다. 자세한 내용은 3장의 "타임아웃" 절을 참고한다.

sessionfailover

기본값	off
값의 범위	[on off]
필수 여부	No
설정 범위	
설명	STF(Session Time Failover) 사용 여부를 설정한다. 사용법은 3장의 "JDBC와 Failover" 절을 참고한다.

server

기본값	localhost
값의 범위	본 매뉴얼의 2장 기본 기능의 "IPv6 접속" 절을 참고한다.

기본값	localhost
필수 여부	Yes
설정 범위	N/A
설명	접속을 시도할 Altibase 서버의 IP 주소 또는 호스트명

sock_rcvbuf_block_ratio

기본값	0
값의 범위	[0 - 216]
필수 여부	No
설정 범위	
설명	<p>소켓 수신 버퍼의 크기를 32K 단위로 설정한다.</p> <p>만약 이 속성의 값이 2로 설정되었다면 소켓 수신 버퍼의 크기는 64K 가 된다.</p> <p>이 속성의 값이 설정되지 않았을 경우, ALTIBASE_SOCKET_RCVBUF_BLOCK_RATIO 환경 변수를 참조하여 값을 설정한다.</p> <p>TCP kernel parameter 중 최대 소켓 수신 버퍼 크기가 이 속성값에 의해 설정된 소켓 수신 버퍼 크기 미만으로 설정되어 있을 경우, 이 속성 은 OS에 따라 무시되거나 에러를 발생시킬 수 있다. (Linux OS 인 경우, 'net.core.rmem_max' TCP kernel parameter에 해당된다)</p>

ssl_enable

기본값	false
값의 범위	[true false]
필수 여부	No
설정 범위	세션
설명	<p>서버에 SSL 통신을 사용해서 접속할지 여부를 설정한다.</p> <p>자세한 내용은 SSL/TLS User's Guide를 참조한다.</p>

time_zone

기본값	DB_TZ(데이터베이스에 설정된 타임존을 이용)
값의 범위	
필수 여부	No
설정 범위	세션
설명	<p>타임존을 설정한다.</p> <p>자세한 내용은 <i>General Reference</i>의 TIME_ZONE 프로퍼티를 참고하도록 한다.</p>

truststore_password

기본값	N/A
값의 범위	임의의 문자열
필수 여부	No
설정 범위	N/A
설명	truststore_url의 비밀번호를 지정할 수 있다.

truststore_type

기본값	JKS
값의 범위	[JKS, JCEKS, PKCS12 외]

기본값	JKS
필수 여부	No
설정 범위	N/A
설명	truststore_url의 TrustStore 타입을 설정한다.

truststore_url

기본값	N/A
값의 범위	임의의 문자열
필수 여부	No
설정 범위	N/A
설명	TrustStore의 경로를 지정한다. TrustStore는 CA의 인증서를 갖고있는 KeyStore이다.

user

기본값	
값의 범위	
필수 여부	Yes
설정 범위	N/A
설명	접속할 데이터베이스의 사용자 ID

utrans_timeout

기본값	3600
값의 범위	Unsigned Integer 범위내의 숫자값
필수 여부	No
설정 범위	세션
설명	UPDATE문 수행 시간의 한계값을 설정한다. 수행 시간이 이 시간을 넘어가면 구문이 자동으로 종료된다. 단위는 초(sec)이다. 이 값이 0이면 무한대를 의미한다.

verify_server_certificate

기본값	true
값의 범위	[true false]
필수 여부	No
설정 범위	N/A
설명	서버의 CA 인증서를 인증할지 여부를 설정한다. 이 값을 FALSE로 설정하면, 클라이언트의 애플리케이션은 서버의 CA 인증서를 인증하지 않는다.

Statement와 ResultSet 다루기

이 절에서는 JDBC로 Altibase 서버에 연결해서 SQL문을 실행하는 기본적인 방법을 코드 예제로 설명한다. 편의상 exception 처리는 생략한다.

예제

```

import java.util.Properties;
import java.sql.*;

//...

String sURL      = "jdbc:Altibase://localhost:20300/mydb";
String sUser     = "SYS";
String sPassword = "MANAGER";

Connection sCon = null;

//Set properties for Connection
Properties sProps = new Properties();
sProps.put( "user",    sUser);
sProps.put( "password", sPassword);

// Load class to register Driver into DriverManager
Class.forName("Altibase.jdbc.driver.AltibaseDriver");

// Create a Connection Object
sCon = DriverManager.getConnection( sURL, sProps );

// Create a Statement Object
Statement sStmt = sCon.createStatement();

// Execute DDL Type Query
sStmt.execute("CREATE TABLE TEST ( C1 VARCHAR (10) )");

// Execute Inserting Query
sStmt.execute("INSERT INTO TEST VALUES ('ABCDE')");

// Execute Selecting Query
// Get Result Set from the Statement Object
ResultSet sRs = sStmt.executeQuery("SELECT * FROM TEST");

// Get ResultSetMetaData Object
ResultSetMetaData sRsMd = sRs.getMetaData();

// Retrieve ResultSet
while(sRs.next())
{
    for(int i=1; i<=sRsMd.getColumnCount(); i++)
    {
        // Get Actual Data and Printout
        System.out.println(sRs.getObject(i));
    }
}

// Eliminate ResultSet Resource
sRs.close();

// Execute Updating Query
sStmt.execute("UPDATE TEST SET C1 = 'abcde'");

// Execute Selecting Query
// Get Result Set from the Statement Object
sRs = sStmt.executeQuery("SELECT * FROM TEST");

// Get ResultSetMetaData Object
sRsMd = sRs.getMetaData();

// Retrieve ResultSet
while(sRs.next())
{
    for(int i=1; i<=sRsMd.getColumnCount(); i++)
    {
        // Get Actual Data and Printout
        System.out.println(sRs.getObject(i));
    }
}

// Eliminate ResultSet Resource
sRs.close();

// Execute Deleting Query
sStmt.execute("DELETE FROM TEST");

```



```
// Execute Selecting Query
// Get Result Set from the Statement Object
sRs = sStmt.executeQuery("SELECT * FROM TEST");

// Get ResultSetMetaData Object
sRsMd = sRs.getMetaData();

// Retrieve ResultSet
while(sRs.next())
{
    for(int i=1; i<=sRsMd.getColumnCount(); i++)
    {
        // Get Actual Data and Printout
        System.out.println(sRs.getObject(i));
    }
}

// Eliminate Resources
sRs.close();
sStmt.close();
```

JDBC Connection Failover

여러 개의 Altibase 서버를 운영하는 환경에서 한 서버의 종료 또는 네트워크 장애 등으로 인해 Altibase JDBC 드라이버를 사용해서 구현한 응용프로그램의 서비스가 불가능해질 수 있다.

이런 장애 상황이 발생하면, 장애가 발생한 서버로 접속하던 클라이언트는 장애 상황을 감지하고, 자동으로 다른 서버로 접속해서 실행 중이던 명령(Statement)들을 처리할 수 있는데 이를 Fail-Over라고 한다.

JDBC 애플리케이션에서 Fail-Over 기능을 사용하는 방법은 *Replication Manual* 의 4장을 참고하기 바란다.

2.기본 기능

Altibase JDBC 드라이버를 사용해서 데이터베이스의 객체를 다루는 기본 방법은 JDBC 표준 인터페이스를 사용하는 방법과 다르지 않다.

이 장에서는 IPv6 주소를 사용해서 데이터베이스 서버에 접속하는 방법과 JDBC 응용 프로그램에서 사용할 수 있는 세 가지 Statement를 비교 설명한다.

IPv6 접속

Altibase의 JDBC 드라이버는 JDBC URL에 IPv6 주소와 IPv6 주소로 바뀌어지는 호스트명의 사용을 지원한다.

개요

URL에 IPv6 주소를 명시하려면, 사각 괄호 ("[]")로 주소를 에워싸야 한다. 예를 들어 localhost를 IP주소로 지정할 때, IPv4 주소 127.0.0.1를 쓸 때는 괄호를 사용하지 않는다. 반면 IPv6 주소 [::1]를 사용하려면 괄호를 반드시 사용해야 한다. IPv6 주소 표기법에 대한 자세한 설명은 *Administrator's Manual*을 참고하기 바란다.

전제 조건

java.net.preferIPv4Stack

IPv6 주소를 사용해서 접속하려면, 클라이언트 실행시 java.net.preferIPv4Stack 속성을 FALSE로 지정해야 한다.

이 속성을 TRUE로 지정하면, 클라이언트 응용프로그램은 IPv6 주소를 사용해서 데이터베이스 서버에 접속할 수 없다.

```
$ java -Djava.net.preferIPv4Stack=false sample [::1]
```

java.net.preferIPv6Addresses

java.net.preferIPv6Addresses 속성은 TRUE 또는 FALSE 어느 것을 지정하여도 Altibase JDBC 드라이버에 아무런 영향을 주지 않는다.

PREFER_IPV6

URL의 server_ip 속성에 호스트명을 입력하면, PREFER_IPV6 속성에 지정한 값에 따라 JDBC 드라이버가 호스트명을 IPv4 주소 또는 IPv6주소로 변환한다.

이 속성이 TRUE이고 server_ip프로퍼티에 호스트명이 입력되면, 클라이언트 응용프로그램은 먼저 호스트명을 IPv6주소로 바꾼다.

그러나 이 속성을 생략하거나 FALSE로 설정하면, 클라이언트 응용프로그램은 먼저 호스트명을 IPv4주소로 바꾼다.

클라이언트 응용프로그램이 처음 접속에 실패하면, 처음과 다른 버전의 IP 주소를 사용해서 접속을 다시 시도할 것이다.

사용법

IPv6 포맷의 IP 주소의 경우 주소값 그대로 지정할 수 있다.

호스트명을 JDBC 드라이버가 IPv4 주소로 변환할지 IPv6주소로 변환할 것인지는 PREFER_IPV6 속성에 지정할 수 있다.

```
Properties sProps = new Properties();
...
sProps.put( "PREFER_IPV6", "FALSE");
```

위와 같이 PREFER_IPV6 속성을 FALSE로 지정하면, JDBC 드라이버는 호스트명을 IPv4 주소형으로 변환한다. 만약 PREFER_IPV6 속성이 TRUE이고 호스트명이 주어지면, 클라이언트 응용프로그램은 먼저 호스트명을 IPv6주소로 바꾼다.

이 속성을 생략하거나 FALSE로 설정하면, 클라이언트 응용프로그램은 먼저 호스트명을 IPv4주소로 바꾼다. JDBC드라이버의 기본 동작은 호스트명을 IPv4주소로 바꿔서 시도하는 것이다.

클라이언트 응용프로그램이 선호하는 버전의 IP주소를 사용해서 처음 접속에 실패하면, 다른 버전의 IP주소를 사용해서 접속을 재시도한다.

예제

```
Connection sCon = null;
Properties sProps = new Properties();

sProps.put( "user", "SYS");
sProps.put( "password", "MANAGER");
sProps.put( "PREFER_IPV6", "FALSE");

String sURL = "jdbc:Altibase://localhost:20300/mydb";
Connection sCon = DriverManager.getConnection( sURL, sProps );
```

Statement, PreparedStatement 및 CallableStatement

JDBC에는 직접 SQL 구문 실행 여부 또는 SQL문에 IN/OUT 파라미터 사용 여부에 따라 사용 가능한 명령문(Statement) 객체가 따로 존재한다. 아래는 각 Statement별 PREPARE 가능 여부와 입/출력 파라미터 사용 가능 여부를 표로 나타낸 것이다.

	PREPARE	IN 파라미터	OUT 파라미터
Statement	X	X	X
PreparedStatement	O	O	X

	PREPARE	IN 파라미터	OUT 파라미터
CallableStatement	O	O	O

Statement

Statement는 정적인 SQL문을 직접 수행할 때 주로 사용된다.

PreparedStatement

PreparedStatement는 SQL구문을 먼저 준비(PREPARE)해 두고, 수행하고자 할 때 주로 사용된다. 같은 구문을 여러 번 수행할 때, Statement 대신 PreparedStatement를 사용하면 성능 향상을 기대할 수 있다.

Altibase JDBC 드라이버는 PreparedStatement 객체가 생성될 때 서버에서 구문을 PREPARE할 것을 지시한다. 이 때 서버가 PREPARE에 실패하면 예외를 반환하고, JDBC 드라이버는 예외를 던진다.

PreparedStatement의 경우 Statement와 달리 입력 파라미터를 사용할 수 있다. 파라미터는 SQL문 내에서 "?" 문자로 나타내며, setXXX() 메소드를 이용해서 값을 설정할 수 있다.

예제

아래는 IN 파라미터와 함께 PreparedStatement를 사용하는 코드 예제이다.

```
PreparedStatement sPrepStmt = sConn.prepareStatement("INSERT INTO t1 VALUES (?, ?)");
sPrepStmt.setInt(1, 1);
sPrepStmt.setString(2, "string-value");
sPrepStmt.execute();
sPrepStmt.close();
```

CallableStatement

CallableStatement는 입력 또는 출력 파라미터를 함께 사용할 수 있다. CallableStatement는 저장 프로시저 또는 저장 함수 호출에 주로 사용된다.

예제

아래는 입력 파라미터와 출력 파라미터를 함께 CallableStatement를 사용하는 코드 예제이다.

```
CallableStatement sCallStmt = connection().prepareCall("{call p1(?, ?)}");
sCallStmt.setInt(1, 1);
sCallStmt.registerOutParameter(2, Types.VARCHAR);
sCallStmt.execute();

String sOutVal = sCallStmt.getString(2);
// todo something ...

sCallStmt.close();
```

내셔널 캐릭터 셋 사용

여기에서는 JDBC에서 NCHAR 및 NVARCHAR 타입 같은 유니코드 타입에 내셔널 캐릭터 문자열을 사용하는 방법을 설명한다.

데이터 조회 및 변경

JDBC를 이용하여 NCHAR, NVARCHAR 타입의 데이터를 조회 및 변경하는 방법은 CHAR, VARCHAR 타입과 동일하다. 즉 CHAR 타입에서 사용했던 getString, setString 등의 메소드를 그대로 사용하면 된다.

상수 문자열 사용

SQL 구문에서 내셔널 캐릭터를 가지는 상수 문자열을 사용하는 방법은 다음과 같다.

- 서버와 연결할 때 NcharLiteralReplace 속성을 true로 설정한다.
- 내셔널 캐릭터의 문자열을 SQL 구문에서 상수 문자열로 사용하기 위해서는 해당 문자열 바로 앞에 'N'을 붙인다.

예제

```
// create table t1 (c1 nvarchar(1000));
Properties sProps;
sProps.put( "user", "SYS");
sProps.put( "password", "MANAGER");
sProps.put( "NcharLiteralReplace", "true");
Connection sCon = DriverManager.getConnection( sURL, sProps );

Statement sStmt = sCon.createStatement();
sStmt.execute("insert into t1 values (N'AB가나')");
ResultSet sRS = sStmt.executeQuery( "select * from t1 where c1 like N'%가나%'");
```

3.고급 기능

이 장에서는 Altibase JDBC 드라이버가 제공하는 보다 향상된 기능들을 소개하고, 사용법을 설명한다.

자동 생성 키

자동 생성 키(Auto-generated Keys)란 테이블의 각 행을 유일하게 가리킬 수 있는 값으로, 데이터베이스에서 자동으로 생성된다.

Altibase에서는 시퀀스(sequence)로 자동 생성 키 역할을 할 수 있다. 이 절에서는 JDBC에서 자동 생성 키 값을 얻는 방법을 설명한다.

사용법

자동 생성 키를 얻기 위해서 먼저 어떤 칼럼에 대해 자동 생성된 키를 얻을 것인지를 명시하는 메소드를 사용해서 Statement 객체를 실행한다. 그리고 getGeneratedKeys() 메소드로 자동 생성된 키의 결과셋을 가져올 수 있다.

또는 어떤 칼럼이 자동 생성된 키 값을 얻을 것인지를 명시하는 메소드를 사용해서 PreparedStatement 객체를 생성하고 실행한 다음, getGeneratedKeys() 메소드로 자동 생성된 키의 결과셋을 가져올 수 있다.

다음은 자동으로 생성된 키를 가져올 수 있는 SQL문을 실행하는 Statement의 메소드들이다.

```
public boolean execute(String aSql, int aAutoGeneratedKeys) throws SQLException;
public boolean execute(String aSql, int[] aColumnIndexes) throws SQLException;
public boolean execute(String aSql, String[] aColumnNames) throws SQLException;
```

다음은 자동으로 생성된 키를 가져올 수 있는 PreparedStatement 객체를 생성하는 Connection의 메소드들이다.

```
public PreparedStatement prepareStatement(String aSql, int aAutoGeneratedKeys) throws SQLException;
public PreparedStatement prepareStatement(String aSql, int[] aColumnIndexes) throws SQLException;
public PreparedStatement prepareStatement(String aSql, String[] aColumnNames) throws SQLException;
```

위의 두 가지 방법 중 하나를 사용해서 SQL문을 실행한 후, Statement의 아래 메소드를 이용해서 자동 생성된 키를 ResultSet 객체로 얻을 수 있다.

```
public ResultSet getGeneratedKeys() throws SQLException;
```

제약 사항

Altibase에서 자동으로 생성된 키를 얻을 때 아래의 제약 사항이 있다.

- 단순 INSERT 문에 대해서만 지원한다.
- Altibase는 AUTO INCREMENT 속성의 칼럼을 지원하지 않기 때문에, 오직 시퀀스로부터만 자동 생성 키를 얻을 수 있다.

다음은 자동 생성 키를 얻을 수 있는 SQL문의 예제이다.

```
INSERT INTO t1 (id, val) VALUES (t1_id_seq.nextval, ?);
```

다음은 자동 생성 키를 얻을 수 없는 SQL문의 예제이다.

```
SELECT * FROM t1;  
EXEC p1;
```

자동 생성 키를 만들지 않는 SQL문을 자동 생성 키 생성 플래그(`Statement.RETURN_GENERATED_KEYS`)와 함께 실행하면 해당 플래그는 무시되고, `getGeneratedKeys()` 메소드는 빈 결과셋을 반환한다.

예제

```
sStmt.executeUpdate(sqStr, Statement.RETURN_GENERATED_KEYS);  
ResultSet sKeys = sStmt.getGeneratedKeys();  
while (sKeys.next())  
{  
    int sKey = sKeys.getInt(1);  
  
    // do somethings...  
}  
sKeys.close();  
sStmt.close();
```

타임아웃

이 절에서는 Altibase 서버와 연결된 클라이언트 세션에서 발생할 수 있는 타임아웃을 설명하고, 타임아웃과 관련된 프로퍼티를 설정하는 방법을 코드 예제로 보여준다.

로그인 타임아웃

`Connection` 객체의 `connect` 메소드를 호출한 후, 서버로부터 응답을 받을 때까지 대기하는 최대 시간을 초과할 때 발생하는 타임아웃이다. 대기하는 최대 시간을 설정하는 속성은 `login_timeout`이며, 설정 값의 단위는 초(second)이다.

코드 예제

아래는 속성을 설정하는 두 가지 방법을 코드 예제로 보여준다.

1. 타임아웃 속성을 추가한 `Properties` 객체를 사용해서 `Connection` 객체를 생성한다.

```
Properties sProps = new Properties();  
...  
sProps("login_timeout", "100");  
...  
Connection sCon = DriverManager.getConnection( sUrl, sProps );
```

2. 타임아웃 속성을 명시한 연결 URL을 사용해서 `Connection` 객체를 생성한다.

```
String sUrl = "jdbc:Altibase://localhost:20300/mydb?login_timeout=100";  
Connection sCon = DriverManager.getConnection( sUrl );
```

응답 타임아웃

Altibase 서버로부터 응답을 기다리는 최대 시간을 초과할 때 발생하는 타임아웃이다. 대기하는 최대 시간을 설정하는 속성은 `response_timeout`이며, 설정 값의 단위는 초(second)이다.

이 값은 서버와 통신하는 모든 메소드 호출에 적용된다.

코드 예제

아래는 응답 타임아웃의 속성을 설정하는 방법을 코드 예제로 보여준다.

1. 타임아웃 속성을 추가한 `Properties` 객체를 사용해서 `Connection` 객체를 생성한다.

```

Properties sProps = new Properties();
...
sProps("response_timeout", "100");
...
Connection sCon = DriverManager.getConnection( sUrl, sProps );

```

2. 타임아웃 속성을 명시한 연결 URL을 사용해서 Connection 객체를 생성한다.

```

String sUrl = "jdbc:Altibase://localhost:20300/mydb?response_timeout=100";
Connection sCon = DriverManager.getConnection( sUrl );

```

3. 애플리케이션 실행 시 인자로 전달한다.

```

java ... -DALTIbase_RESPONSE_TIMEOUT=100 ...

```

4. 환경 변수를 설정한다.

```

// Linux
export ALTIbase_RESPONSE_TIMEOUT=100

```

DataSource

Altibase JDBC 드라이버는 연결 설정을 담고 있는 파일을 사용해서 데이터베이스에 접속하는 방법을 제공한다. 이 때 설정 파일에 포함되어 있는 데이터베이스 서버로의 연결 정보들의 집합을 DataSource라고 한다.

DataSource 설정 방법

DataSource는 altibase_cli.ini 파일에 아래의 형식으로 설정한다.

```

# comment

[ datasource_name ]
Server=localhost # comment
Port=20300
User=sys
Password=manager

```

만약 더 추가할 연결 속성이 있다면, “*key=value*” 형태의 문자열을 줄 단위로 적으면 된다.

JDBC 드라이버는 아래 순서의 경로대로 altibase_cli.ini 파일을 찾는다.

1. ./altibase_cli.ini
2. \$HOME/altibase_cli.ini
3. \$ALTIBASE_HOME/conf/altibase_cli.ini

DataSource를 이용한 접속

DataSource를 사용해서 서버에 접속하려면, IP 주소와 포트 번호 대신에 altibase_cli.ini 파일에 명시한 DSN(Data Source Name)을 연결 URL에 명시하면 된다.

다음은 DSN을 이용한 연결 URL의 예제이다.

```

jdbc:Altibase://datasource_name
jdbc:Altibase://datasource_name:20301
jdbc:Altibase://datasource_name:20301?sys=user&password=pwd

```

연결 URL에 DSN을 명시할 때, port 또는 다른 속성을 추가로 지정할 수 있다. 단, altibase_cli.ini 파일에 지정한 속성을 연결 URL에 중복해서 지정하면, 파일의 값은 무시되고 연결 URL에 지정한 속성값이 사용된다.

Connection Pool

다음의 방법으로 연결 풀 (Connection Pool)을 설정하고 관리할 수 있다.

- AltibaseConnectionPoolDataSource 사용: WAS에서 연결 풀을 사용할 때, 이 클래스를 WAS의 JDBC연결 풀 설정에서 지정한다. 6.3.1 버전 이하의 Altibase

JDBC 드라이버에서 이 클래스의 이름은 `ABConnectionPoolDataSource`였다.

`AltibaseConnectionPoolDataSource`에서 설정하는 속성 정보는 아래와 같다.

프로퍼티 이름	설명
databaseName	데이터베이스 이름
dataSourceName	dataSource 이름
loginTimeout	데이터베이스 로그인을 위한 최대 대기 시간
logWriter	dataSource를 위한 Log writer
password	데이터베이스 패스워드
portNumber	데이터베이스 포트 넘버
serverName	데이터베이스 서버명
URL	Altibase 연결을 위한 Connection string 정보 (대문자 주의)
user	데이터베이스 사용자 ID

WAS (Web Application Server) 설정

Altibase는 아래의 웹 애플리케이션 서버와 함께 사용할 수 있다.

- Tomcat 8.x
 - Code example

```
Context initContext = new InitialContext();
Context envContext = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/altihdb");
Connection conn = ds.getConnection();
// ...
```

- WebLogic 12.x
- Jeus 6.x

웹 애플리케이션 서버에서 JDBC 드라이버와 연결 풀을 설정하고 사용하는 방법에 대해서는 각각의 제품별 매뉴얼을 참조하기 바란다.

Tomcat 8.x

Apache Tomcat의 설치 및 설정 방법에 대한 자세한 내용은 <http://tomcat.apache.org/tomcat-8.0-doc/index.html>를 참조하기 바란다.

Context configuration

아래와 같이 Context에 JNDI DataSource를 추가한다.

```
<Context>

<Resource name="jdbc/altihdb" auth="Container" type="javax.sql.DataSource"
driverClassName="Altibase.jdbc.driver.AltibaseDriver"
url="jdbc:Altibase://localhost:20300/mydb"
username="SYS" password="MANAGER"
maxTotal="100" maxIdle="30" maxWaitMillis="10000" />

</Context>
```

web.xml configuration

```

<!-- web.xml -->
<resource-ref>
  <description>Altibase Datasource example</description>
  <res-ref-name>jdbc/altihdb</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

```

Code example

```

Context initContext = new InitialContext();
Context envContext = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/altihdb");
Connection conn = ds.getConnection();
// ...

```

WebLogic 12.x

WebLogic Server의 설치 및 설정 방법에 대한 자세한 내용은 <http://docs.oracle.com/middleware/1213/wls/index.html>를 참조하기 바란다.

아래의 링크를 참조하여 JDBC datasource와 Connection Pool을 설정할 수 있다.

http://docs.oracle.com/middleware/1213/wls/WLACH/taskhelp/jdbc/jdbc_datasources/CreateDataSources.html

<http://docs.oracle.com/middleware/1213/wls/WLACH/pagehelp/JDBCjdbcdatasourcesjdbcdatasourceconfigconnectionpooltitle.html>

Altibase를 위한 설정 옵션은 아래와 같다.

- 데이터베이스 유형(Database Type): 기타(Other)
- 드라이버 클래스 이름(Driver Class Name): Altibase.jdbc.driver.AltibaseDriver
- URL: jdbc:Altibase://localhost:20300/mydb

Jeus 6.x

\$JEUS_HOME/config/JeusMain.xml 파일의 <data-source> 요소를 편집하여 연결 풀을 설정한다.


```

<!-- JeusMain.xml -->
<resource>
  <data-source>
    <database>
      <vendor>others</vendor>
      <export-name>jdbc/altihdb</export-name>
      <data-source-class-name>
        Altibase.jdbc.driver.AltibaseConnectionPoolDataSource
      </data-source-class-name>
      <data-source-type>ConnectionPoolDataSource</data-source-type>
      <auto-commit>true</auto-commit>
      <property>
        <name>PortNumber</name>
        <type>java.lang.Integer</type>
        <value>20300</value>
      </property>
      <property>
        <name>Password</name>
        <type>java.lang.String</type>
        <value>MANAGER</value>
      </property>
      <property>
        <name>ServerName</name>
        <type>java.lang.String</type>
        <value>localhost</value>
      </property>
      <property>
        <name>ConnectionAttributes</name>
        <type>java.lang.String</type>
        <value>;create=true</value>
      </property>
      <property>
        <name>DatabaseName</name>
        <type>java.lang.String</type>
        <value>mydb</value>
      </property>
      <property>
        <name>User</name>
        <type>java.lang.String</type>
        <value>SYS</value>
      </property>
    </database>
  </data-source>
</resource>

```

Multiple ResultSet

Altibase용 PSM(저장 프로시저 및 저장 함수)은 여러 개의 결과셋을 클라이언트에 반환할 수 있다. 이 절에서는 여러 개의 결과셋을 반환하는 PSM의 예제를 가지고, JDBC 애플리케이션에서 이러한 결과셋들을 다루는 방법을 코드 예제로 살펴본다.

다음은 여러 개의 결과셋을 반환하는 PSM 예제이다.

```

CREATE TYPESET my_type
AS
  TYPE my_cur IS REF CURSOR;
END;

CREATE PROCEDURE p1 (p1 OUT MY_TYPE.MY_CUR, p2 out MY_TYPE.MY_CUR)
AS
BEGIN
  OPEN p1 FOR 'SELECT * FROM t1';
  OPEN p1 FOR 'SELECT * FROM t2';
END;

```

다음은 JDBC 애플리케이션에서 PSM 호출로 반환된 여러 개의 결과셋들을 다루는 코드 예제이다.

```

CallableStatement sCallStmt = connection().prepareCall("{call p1()}");
sCallStmt.execute();
ResultSet sRs = null;
ResultSetMetaData sRsMd = null;

do{
    sRs = sCallStmt.getResultSet();
    sRsMd = sRs.getMetaData();

    if(sRsMd != null)
    {
        while(sRs.next())
        {
            // do something
            for(int i=1; i <= sRsMd.getColumnCount(); i++)
            {
                System.out.println(sRs.getString(i));
            }
        }
    }
}while(sCallStmt.getMoreResults());
sCallStmt.close();

```

JDBC와 Failover

이 절은 Altibase JDBC 애플리케이션에서 Failover 기능을 사용하는 방법을 설명한다.

Failover란

Failover란 데이터베이스 서버에 장애가 발생하여 연결이 끊어졌을 때, 애플리케이션이 즉시 다른 서버로 연결을 생성하여 기존에 수행하던 작업을 계속하는 기능을 말한다.

Failover는 아래 두 가지 방식으로 동작할 수 있다.

- CTF(Connection Time Failover)
데이터베이스로의 접속 시도를 실패한 경우, 다른 서버로 접속을 재시도하는 동작 방식이다. CTF는 Connection 객체의 connect 메소드를 호출할 때 발생할 수 있다.
- STF(Session Time Failover)
SQL문을 수행하여 서버로부터 결과를 받기 전에 연결 오류가 발생한 경우, 다른 서버로 접속하여 사용자가 지정한 작업을 계속하는 동작 방식이다. STF는 connect를 제외한 서버와 통신을 하는 모든 메소드 수행 시에 발생할 수 있다.

Failover에 대한 자세한 내용은 *Replication Manual*의 “Failover”장을 참고하도록 한다.

사용 방법

여기에서는 JDBC 애플리케이션에서 CTF 및 STF 기능을 사용하는 방법을 설명한다.

CTF

Properties 객체에 아래의 속성을 추가해서 CTF 기능을 사용할 수 있다.

```

Properties sProps = new Properties();
sProps.put("alternateservers", "database1:20300, database2:20300");
sProps.put("connectionretrycount", "5");
sProps.put("connectionretrydelay", "2");
sProps.put("sessionfailover", "off");

```

각각의 속성에 대한 자세한 설명은 1장의 "연결 속성" 절을 참고하라.

STF

CTF 기능을 설정하는 속성에 추가로 "SessionFailover=on"을 설정해서 STF 기능을 사용할 수 있다.

데이터베이스 서버로 접속을 시도하는 것을 제외한 통신 상황에서, 클라이언트가 서버의 장애를 감지하면 먼저 CTF 과정을 수행하여 접속을 복원한다. 그 다음, 클라이언트는 사용자가 등록한 콜백 함수를 수행한 후에, 사용자가 Failover 발생을 인지할 수 있도록 Failover Success Exception을 발생시킨다. 이 때 모든 서버로 Failover가 실패하면 드라이버는 원래 발생했던 Exception을 던진다.

아래는 사용자가 작성해야 할 Failover 콜백 함수를 위한 인터페이스이다.

```
public interface AltibaseFailoverCallback
{
    public final static class Event
    {
        public static final int BEGIN      = 0;
        public static final int COMPLETED = 1;
        public static final int ABORT      = 2;
    }
    public final static class Result
    {
        public static final int GO    = 3;
        public static final int QUIT = 4;
    }
    int failoverCallback(Connection aConnection,
                        Object      aAppContext,
                        int         aFailoverEvent);
};
```

다음은 사용자가 Failover 콜백 함수를 등록하고 해제하는 과정을 보여주는 코드 예제이다.

```
public class UserDefinedFailoverCallback implements AltibaseFailoverCallback
{
    ...

    public int failoverCallback(Connection aConnection,
                                Object      aAppContext,
                                int         aFailoverEvent)
    {
        // User Defined Code
        // Result.GO나 Result.QUIT 중 한 가지 값을 반환해야 함.
    }

    ...
}
```

위의 AltibaseFailoverCallback 인터페이스에 포함된 Event 상수는 사용자가 작성한 Failover 콜백 함수가 JDBC 드라이버에 의해 호출될 때, 콜백 함수의 세 번째 인자인 aFailoverEvent로 전달된다. 각 Event 상수의 의미는 다음과 같다.

- Event.BEGIN: Session Failover가 시작됨
- Event.COMPLETED: Session Failover가 성공하였음
- Event.ABORT: Session Failover가 실패하였음

AltibaseFailoverCallback 인터페이스에 포함된 Result 상수는 사용자가 작성하는 콜백 함수에서 반환할 수 있는 값들이다. 콜백 함수에서 Result 상수 이외의 값을 반환하면 Failover가 정상적으로 동작하지 않는다.

- Result.GO: 콜백 함수에서 이 상수값이 반환되면, JDBC 드라이버는 STF의 다음 과정을 계속해서 진행한다.
- Result.QUIT: 콜백 함수에서 이 상수값이 반환되면, JDBC 드라이버는 STF 과정을 종료한다.

다음은 사용자가 작성하는 Failover 콜백 함수의 두 번째 인자로 사용할 수 있는 객체의 코드 예제이다.

```
public class UserDefinedAppContext
{
    // User Defined Code
}
```

사용자가 구현한 애플리케이션의 정보를 STF 과정에서 사용할 필요가 있는 경우, Failover 콜백 함수를 등록하면서 콜백 함수에 전달될 객체를 지정할 수 있다. 콜백 함수를 등록하는 registerFailoverCallback 메소드의 두 번째 인자로 이 객체를 지정하면, 실제로 콜백 함수가 호출될 때 이 객체가 전달된다. 다음은 이런 과정을 코드로 나타낸 예제이다.

```
// 사용자 정의 콜백 함수 객체 생성
UserDefinedFailoverCallback sCallback = new UserDefinedFailoverCallback();
// 사용자 정의 애플리케이션 정보 객체 생성
UserDefinedAppContext sAppContext = new UserDefinedAppContext();

...

Connection sCon = DriverManager.getConnection(sURL, sProp);

// 사용자 정의 애플리케이션 객체와 함께 콜백 함수 등록
((AltibaseConnection)sCon).registerFailoverCallback(sCallback, sAppContext);

...

// 콜백 함수 해제
((AltibaseConnection)sCon).deregisterFailoverCallback();
```

코드 예제

STF를 위한 콜백 함수를 구현하는 코드 예제이다.

아래의 예제는 여러 가지 경우를 무시한 단순 코드이므로, 사용자 애플리케이션에서 그대로 사용할 수 없음을 주의해야 한다.

```

public class MyFailoverCallback implements AltibaseFailoverCallback
{
    public int failoverCallback(Connection aConnection, Object aAppContext,int aFailoverEvent)
    {
        Statement sStmt = null;
        ResultSet sRes = null;

        switch (aFailoverEvent)
        {
            // 사용자 어플리케이션의 로직상 Failover 시작 전에 필요한 작업을 진행할 수 있다.
            case Event.BEGIN:
                System.out.println("Failover Started .... ");
                break;
            // 사용자 어플리케이션의 로직상 Failover 완료 후에 필요한 작업을 진행할 수 있다.
            case Event.COMPLETED:
                try
                {
                    sStmt = aConnection.createStatement();
                }
                catch( SQLException ex1 )
                {
                    try
                    {
                        sStmt.close();
                    }
                    catch( SQLException ex3 )
                    {
                    }
                    return Result.QUIT;
                }

                try
                {
                    sRes = sStmt.executeQuery("select 1 from dual");
                    while(sRes.next())
                    {
                        if(sRes.getInt(1) == 1 )
                        {
                            break;
                        }
                    }
                }
                catch ( SQLException ex2 )
                {
                    try
                    {
                        sStmt.close();
                    }
                    catch( SQLException ex3 )
                    {
                    }
                    // Failover 과정을 종료한다.
                    return Result.QUIT;
                }
                break;
            }
            // Failover 과정을 계속한다.
            return Result.GO;
        }
    }
}

```

다음은 STF 성공 여부를 확인하는 코드 예제이다. STF가 성공했는지 실패했는지를 판단하는 방법은 SQLException의 ErrorCode가 Validation.FAILOVER_SUCCESS와 일치하는지 확인하는 것이다. while loop 내에 Failover 검증 코드를 삽입한 이유는 Failover가 성공하더라도 이전에 수행중이던 작업은 다시 수행해야 하기 때문이다.

```
// 반드시 수행하려던 작업을 재수행할 수 있도록 구현해야 한다.
// 이 경우에는 while loop를 사용하였다.
while (true)
{
    try
    {
        sStmt = sConn.createStatement();
        sRes = sStmt.executeQuery("SELECT C1 FROM T1");
        while (sRes.next())
        {
            System.out.println("VALUE : " + sRes.getString(1));
        }
    }
    catch (SQLException e)
    {
        // Failover 성공 여부 확인
        if (e.getErrorCode() == AltibaseFailoverCallback.FailoverValidation.FAILOVER_SUCCESS)
        {
            // Failover가 성공했으므로 Exception을 무시하고 계속 진행한다.
            continue;
        }
        System.out.println("EXCEPTION : " + e.getMessage());
    }
    break;
}
```

JDBC Escapes

JDBC 스펙은 데이터베이스 제품들에 대해 벤더 특유의 SQL 문법을 JDBC 애플리케이션이 인식할 수 있도록 escape 문법을 제공한다. 즉, escape 문법이 포함된 SQL문에 대해서는 JDBC 드라이버가 자신의 데이터베이스에 맞는 SQL문으로 변환한다.

아래는 JDBC 스펙에서 지원하는 escape가 포함된 SQL문과 Altibase JDBC 드라이버가 이것을 Altibase용으로 변환한 SQL문을 정리한 표이다.

종류	JDBC 스펙에서 지원하는 SQL문	Altibase용으로 변환된 SQL문
ESCAPE	SELECT cVARCHAR FROM t1 WHERE cVARCHAR LIKE '%a %b%' {escape ' '}	SELECT cVARCHAR FROM t1 WHERE cVARCHAR LIKE '%a %b%' escape ' '
FN	SELECT {fn concat('concat', 'test')} FROM dual	SELECT concat('concat', 'test') FROM dual
DTS	UPDATE t1 SET cDATE = {d '1234-12-30'}	UPDATE t1 SET cDATE = to_date('1234-12-30', 'yyyy-MM-dd')
	UPDATE t1 SET cDATE = {t '12:34:56'}	UPDATE t1 SET cDATE = to_date('12:34:56', 'hh24:mi:ss')
	UPDATE t1 SET cDATE = {ts '2010-01-23 12:23:45'}	UPDATE t1 SET cDATE = to_date('2010-01-23 12:23:45', 'yyyy-MM-dd hh24:mi:ss')
	UPDATE t1 SET cDATE = {ts '2010-11-29 23:01:23.971589'}	UPDATE t1 SET cDATE = to_date('2010-11-29 23:01:23.971589', 'yyyy-MM-dd hh24:mi:ss.ff6')
CALL	{call p1()}	execute p1()
	{? = call p2(?)}	execute ? := p2(?)
OJ	SELECT * FROM {oj t1 LEFT OUTER JOIN t2 ON t1.cINT = t2.cINT}	SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.cINT = t2.cINT

ResultSet 사용하기

이 절은 Altibase JDBC 드라이버가 지원하는 ResultSet의 유형과 그 사용법을 설명한다.

ResultSet 생성

결과셋은 데이터베이스에 대해 쿼리문을 실행할 때 생성되며, JDBC의 ResultSet 객체에 대응한다.

다음은 JDBC에서 ResultSet 객체를 생성하는 메소드들이다.

```
public Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException;

public Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException;

public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency) throws SQLException;

public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException;

public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency) throws SQLException

public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLExc
```

ResultSet의 유형

JDBC의 ResultSet 객체는 결과셋 내에서 현재 행을 가리키는 커서를 관리하고 유지한다. 기본적인 ResultSet 객체의 커서는 업데이트가 불가능하고 순방향으로만 이동하는 커서이지만, 옵션을 사용해서 스크롤 가능하고 업데이트 가능한 ResultSet 객체를 생성할 수 있다.

다음은 사용자가 지정 가능한 ResultSet 객체의 유형이다.

- TYPE_FORWARD_ONLY
스크롤이 불가능하며, 커서를 순방향으로만 이동할 수 있다. 데이터베이스 서버에서 커서가 열리는 시점에 결과셋의 데이터가 결정된다.
- TYPE_SCROLL_INSENSITIVE
스크롤이 가능하므로, 커서를 순방향, 역방향, 또는 위치를 지정해서 이동할 수 있다. 데이터베이스 서버에서 커서가 열리는 시점에 결과셋의 데이터가 결정된다. 서버에서 가져온 결과셋을 클라이언트에 누적해서 캐시하므로 메모리 사용량이 증가할 수 있다.
- TYPE_SCROLL_SENSITIVE
스크롤이 가능하므로, 커서를 순방향, 역방향, 또는 위치를 지정해서 이동할 수 있다. 데이터베이스 서버에서 커서가 열리는 시점에 결과셋이 결정되지만, 결과셋 내의 데이터는 클라이언트가 가져오거나 갱신하는 시점에 결정된다. 서버에서 가져온 결과셋을 클라이언트에 누적해서 캐시하므로 메모리 사용량이 증가할 수 있다.

Concurrency

ResultSet 객체를 통한 업데이트 허용 여부를 결정하는 옵션이다. 아래의 두 가지 상수 중 하나를 사용할 수 있다.

- CONCUR_READ_ONLY
업데이트를 허용하지 않는다. 기본값이다.
- CONCUR_UPDATABLE
ResultSet 객체를 이용한 업데이트를 허용한다.

Holdability

트랜잭션 커밋 후에도 ResultSet 객체를 유지할 것인지를 결정하는 옵션이다. 아래 두 가지 상수 중 하나를 사용하면 된다.

- CLOSE_CURSORS_AT_COMMIT
트랜잭션이 커밋 될 때, 커서가 닫힌다.
- HOLD_CURSORS_OVER_COMMIT
트랜잭션을 커밋하더라도 커서는 유지된다. 커서가 열린 후 한 번이라도

트랜잭션이 커밋되었다면, 그 커서는 이 후의 commit, rollback 수행에도 계속 유지된다. 하지만 커서가 열린 후 한 번도 커밋을 하지 않았다면, 트랜잭션이 rollback될 때 그 커서는 닫힌다.

주의사항

- ResultSet 객체를 위해 JDBC 드라이버가 클라이언트에 FetchSize에 설정된 개수만큼 행을 캐시하고 있으므로, 커서가 닫히더라도 캐시에 남아있는 데이터는 애플리케이션에서 가져갈 수 있다. 만약 커서가 닫힌 걸 애플리케이션에서 바로 감지하고 싶다면, FetchSize를 1로 설정하면 된다.
- Altibase JDBC 드라이버의 Holdability 기본값은 CLOSE_CURSORS_AT_COMMIT으로, JDBC 스펙의 기본값인 HOLD_CURSORS_OVER_COMMIT와 다르다. Holdability가 HOLD_CURSORS_OVER_COMMIT인 세션에서는 setAutoCommit() 메소드로 자동커밋 모드를 변경하기 전에 열려 있는 ResultSet 객체를 반드시 닫아야 한다. 아래는 오류가 발생하는 예제 코드이다.

```
sCon = getConnection();
sStmt = sCon.createStatement();
byte[] br;
byte[] bb = new byte[48];
for(byte i = 0; i < bb.length;i++) bb[i] = i;

sCon.setAutoCommit(false);

sStmt.executeUpdate("insert into Varbinary_Tab values(null)");
sCon.commit();

sPreStmt = sCon.prepareStatement("update Varbinary_Tab set VARBINARY_VAL=?");
sPreStmt.setObject(1, bb, java.sql.Types.VARBINARY);
sPreStmt.executeUpdate();

sRS = sStmt.executeQuery("Select VARBINARY_VAL from Varbinary_Tab");
sRS.next();
br = sRS.getBytes(1);

sCon.commit();
sCon.setAutoCommit(true); -> (1)
```

(1)에서 다음과 같은 exception이 발생한다.

```
java.sql.SQLException: Several statements still open
    at Altibase.jdbc.driver.ex.Error.processServerError(Error.java:320)
    at Altibase.jdbc.driver.AltibaseConnection.setAutoCommit(AltibaseConnection.java:988)
    at HodabilityTest.testHoldability(HodabilityTest.java:46)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:616)
```

exception이 발생하지 않게 하려면 의 sCon.setAutoCommit(true)에 앞서 sRs.close()를 호출해야 한다.

- ? : Holdability 유형이 HOLD_CURSORS_OVER_COMMIT인 ResultSet 객체를 사용하기 위해서는 클라이언트 세션이 Non-Autocommit 모드이거나 clientside_auto_commit 연결 속성이 on으로 설정되어야 한다. clientside_auto_commit 연결 속성을 on으로 설정하면, Holdability 유형이 자동으로 HOLD_CURSORS_OVER_COMMIT으로 변경된다.

예제


```
Statement sUpdStmt = sConn.prepareStatement("UPDATE t1 SET val = ? WHERE id = ?");
Statement sSelStmt = sConn.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY, ResultSet.HOLD_CURSORS_OVER_COMMIT);
ResultSet sRS = sSelStmt.executeQuery("SELECT * FROM t1");
while (sRS.next())
{
    // TODO : set parameters

    sUpdStmt.execute();
    sConn.commit();
}
sRS.close();
```

제약 사항

Updatable ResultSet 또는 Scrollable ResultSet을 사용하기 위해서는 결과셋을 가져오는 SELECT 쿼리문에 다음의 제약 사항이 있다.

업데이트가 가능한 결과셋을 사용하기 위해서는,

- FROM 절에 한 개의 테이블만 지정할 수 있다.
- SELECT 리스트에 순수 칼럼만 지정할 수 있다. 수식 또는 함수가 포함될 수 없다. 그리고, NOT NULL 제약조건이 있으면서 DEFAULT값이 없는 칼럼은 SELECT 리스트에 반드시 포함되어야 한다.

Scrollable-Sensitive 결과셋을 사용하기 위해서는,

- FROM 절에 한 개의 테이블만 지정할 수 있다.

PSM을 수행하는 경우에는 기본 유형의 ResultSet 객체만 사용할 수 있다. 만약 사용자가 기본 유형이 아닌 옵션을 지정하면, 그 옵션은 무시된다.

CONCUR_UPDATABLE하고 TYPE_SCROLL_SENSITIVE한 ResultSet 객체는 JDBC 드라이버 내부적으로 한 개의 Statement를 더 사용하기 때문에, Statement 개수 제약에 더 일찍 도달할 수 있다. 따라서 이러한 유형의 결과셋을 많이 사용하는 경우에는 Statement의 최대 개수를 설정해야 한다.

업데이트가 가능하고 스크롤이 가능한 결과셋은 많은 데이터를 포함하고 있으므로, 일반적으로 순방향 전용의 결과셋에 비해 메모리 사용량이 높다. 따라서 결과셋이 큰 경우에는 메모리가 부족할 수 있으므로, 이러한 유형을 사용하지 않기를 권장한다.

ResultSet 객체의 특성은 위에서 설명한 ResultSet 유형, concurrency 유형, 및 holdability 유형으로 결정된다. 사용자는 이들 세 값을 임의의 조합으로 지정할 수 있지만, 결과셋을 만드는 쿼리문에 따라 사용자가 지정한 조합이 허용되지 않을 수도 있다. 이 경우 드라이버는 예외를 발생하지 않고 가능한 조합으로 변환한다. 즉, 아래와 같이 왼쪽 유형이 불가능한 경우 오른쪽 유형으로 자동 변환한다.

- TYPE_SCROLL_SENSITIVE → TYPE_SCROLL_INSENSITIVE
- CONCUR_UPDATABLE → CONCUR_READ_ONLY
- HOLD_CURSORS_OVER_COMMIT → CLOSE_CURSORS_AT_COMMIT

이렇게 내부적으로 변환이 발생하면, 경고를 통해 변환 발생 여부를 확인할 수 있다.

ResultSet 객체의 유형이 TYPE_SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE인 경우 메모리 사용량 증가로 인해 ResultSet의 결과가 349,502건으로 제한되어 있다. 이 값을 초과할 경우 Dynamic array cursor overflow 에러가 발생할 수 있다.

Hole 감지

TYPE_SCROLL_SENSITIVE 유형의 ResultSet 객체는 fetch할 때 서버로부터 최신 데이터를 가져온다. 따라서 커서가 열리는 순간에는 보였던 처음의 행이 스크롤 되면서 안 보일 수 있다. 예를 들어, ResultSet 객체에 있던 행이 다른 Statement를 통해 지워진다면, 그 행은 ResultSet 객체에서 더 이상 볼 수 없게 된다. 이렇게 볼 수 없게 된 행을 Hole이라고 한다.

아래는 JDBC에서 Hole을 검출하는 코드 예제이다.

```

while (sRS.next())
{
    if (sRS.rowDeleted())
    {
        // HOLE DETECTED!!!
    }
    else
    {
        // do something ...
    }
}

```

Hole에서는 유효한 데이터를 얻을 수 없으며, Hole에 해당하는 **ResultSet**의 반환값은 다음 중 하나이다:

- SQL 데이터형의 NULL
- 참조형으로는 null
- 값으로는 0

Fetch Size

Altibase JDBC 드라이버는 성능 향상을 위해, **ResultSet** 객체를 위한 데이터를 서버로부터 가져올 때 한 행씩 가져오는 대신에 여러 행을 한번에 가져와서 클라이언트에 캐시한다. 이것을 prefetch라고 하며, **Statement**객체의 **setFetchSize()** 메소드를 이용해서 한번에 가져오는 행의 개수를 설정할 수 있다.

```

public void setFetchSize(int aRows) throws SQLException;

```

Altibase JDBC 드라이버에서는 0에서 2147483647까지의 값으로 설정할 수 있다. JDBC 스펙에는 이 범위를 벗어난 값을 지정할 때 **Exception**을 발생하도록 되어 있지만, Altibase JDBC 드라이버는 편의상 예외를 발생하지 않고 무시한다.

0을 설정하면, Altibase 서버가 클라이언트로 한번에 반환할 크기를 스스로 결정한다. 이 경우, 한 행의 크기에 따라 반환되는 행의 개수가 달라질 것이다.

FetchSize 값은 **Scroll-Sensitive** 결과셋에서 특히 중요하다. 사용자가 **Scroll-Sensitive** 결과셋으로부터 데이터를 가져갈 때, 드라이버는 prefetch한 것을 우선으로 반환한다. 그러므로, 데이터베이스의 데이터가 갱신되었더라도, 그 행이 prefetch한 캐시에 존재하는 한 캐시의 데이터가 사용자에게 반환된다. 사용자가 데이터베이스의 최신 데이터를 보기 원한다면, **FetchSize**를 1로 하면 된다. 그러나 이런 설정은 서버로부터 데이터를 가져오는 빈도수를 높여 성능을 떨어뜨릴 수 있다.

Refreshing Rows

ResultSet 객체의 **refreshRow()** 메소드를 사용하면, **SELECT**문을 실행하지 않고서도 서버로부터 이미 가져온 데이터를 다시 가져올 수 있다. **refreshRow()** 메소드는 현재 행을 기준으로 **FetchSize**에 설정된 행 개수만큼 가져온다. 그러므로 이 메소드를 사용하기 위해서는, 커서가 결과셋에서 어떤 행이라도 가리키고 있는 상태이어야 한다.

이 메소드는 **ResultSet** 객체의 유형이 다음과 같을 때 동작한다.

- **TYPE_SCROLL_SENSITIVE & CONCUR_UPDATABLE**
- **TYPE_SCROLL_SENSITIVE & CONCUR_READ_ONLY**

TYPE_FORWARD_ONLY일 경우에는 이 메소드를 호출하면 예외가 발생하고, **TYPE_SCROLL_INSENSITIVE**일 경우에는 아무런 동작도 일어나지 않는다.

Atomic Batch

알티베이스 JDBC 드라이버는 **Atomic Batch** 기능을 제공하여, 일괄처리(Batch)의 원자성을 보장할 뿐 아니라 대용량의 데이터 삽입을 빠르게 처리할 수 있도록 지원한다.

이 절에서는 알티베이스 JDBC 드라이버가 지원하는 Atomic Batch 사용법에 대하여 설명한다.

사용법

Atomic Batch 기능을 사용하기 위해서 우선 자바 프로그래밍에서 PreparedStatement 객체를 생성하여 AltibasePreparedStatement 클래스 타입으로 캐스팅한다.

다음은 Atomic Batch 기능을 사용하기 위해 호출하는 setAtomicBatch() 메소드이다.

```
public void setAtomicBatch(boolean aValue) throws SQLException
```

Atomic Batch가 PreparedStatement 객체에 설정되었는지 여부를 확인하려면 getAtomicBatch() 메소드를 호출한다.

```
public boolean getAtomicBatch()
```

제약 사항

Altibase에서 Atomic Batch 기능을 사용할 때 아래의 제약 사항이 있다.

- 단순 INSERT 문에 대해서만 지원한다. 복합 INSERT 문이나 UPDATE, DELETE 등의 DML 문에 대한 정합성을 보장하지 못한다.
- 트리거가 동작할 때 수행 단위가 Each Statement일 경우 트리거는 한 번만 동작한다.
- SYSDATE는 1번만 동작한다.

예제

```
.....
Connection con = sConn = DriverManager.getConnection(aConnectionStr, mProps);
Statement stmt = con.createStatement();

try
{
    stmt.execute("Drop table " + TABLE_NAME); } catch (SQLException e) { }
    stmt.execute("create table " + TABLE_NAME + "(c1 VARCHAR (1000))");

    PreparedStatement sPrepareStmt = con.prepareStatement("insert into " + TABLE_NAME + " values(?)");
    ((AltibasePreparedStatement)sPrepareStmt).setAtomicBatch(true);

    for(int i = 1; i <= MAX_RECORD_CNT; i++)
    {
        sPrepareStmt.setString(1, String.valueOf(i % 50));
        sPrepareStmt.addBatch();

        if(i%BATCH_SIZE == 0)
        {
            sPrepareStmt.executeBatch();
            con.commit();
        }
    }
    con.commit();
}
catch (SQLException e)
{
    System.out.println(e.getMessage());
}
.....
```

Date, Time, Timestamp

이 절은 날짜형 데이터 타입인 Date, Time, 및 Timestamp 각각의 의미와, Altibase JDBC 드라이버에서 지원하는 데이터 변환 범위를 설명한다.

의미

- Date: 날짜만 표현

- Time: 시각을 표현 (날짜가 포함될 수도 있음)
- Timestamp: 날짜, 시각, 초 및 그 이하의 시각까지 표현

변환 표

아래의 표는 setObject 메소드에 전달되는 객체의 타입에 따라 Altibase JDBC 드라이버가 처리하는 형식을 보여준다.

전달객체	String	Date	Time	Timestamp
setObject (DATE)	2134-12-23 00:00:00.0 사용자가 시분초 부분을 입력하면 오류 발생. 드라이버가 0으로 설정함.	2134-12-23 00:00:00.0 시분초는 입력해도 드라이버가 무시함.	SQLException: UNSUPPORTED_TYPE_CONVERSION	2134-12-23 12:34:56.123456
setObject (TIME)	1970-01-01 12:34:56.0 사용자가 년월일 또는 nano초 부분을 입력하면 오류 발생. 드라이버가 기준 년월일로 설정함.	2134-12-23 12:34:56.0	2134-12-23 12:34:56.0	2134-12-23 12:34:56.0
setObject (TIMESTAMP)	2134-12-23 12:34:56.123456	2134-12-23 00:00:00.0 시분초는 입력해도 드라이버가 무시함.	SQLException: UNSUPPORTED_TYPE_CONVERSION	2134-12-23 12:34:56.123456
setString()	DATE_FORMAT 속성에 설정한 형식으로 입력해야 함.	-	-	-
setDate()	-	2134-12-23 00:00:00.0 시분초는 입력해도 드라이버가 무시함.	-	-
setTime()	-	-	2134-12-23 12:34:56.0	-
setTimestamp()	-	-	-	2134-12-23 12:34:56.123456

아래는 데이터베이스에 저장되어 있는 DATE 타입의 값(1234-01-23 12:23:34.567123)을 getDate(), getTime(), 및 getTimestamp() 메소드를 사용해서 가져오는 값을 보여준다.

함수	반환값
getDate()	1234-01-23 00:00:00.0
getTime()	1234-01-23 12:23:34.0
getTimestamp()	1234-01-23 12:23:34.567123

GEOMETRY

이 절은 Altibase가 제공하는 GEOMETRY 타입의 데이터를 JDBC 애플리케이션에서 조작하는 방법을 설명한다.

사용 방법

Altibase JDBC 애플리케이션에서는 GEOMETRY 타입의 데이터에 대해 byte 배열을 사용해서 조작할 수 있다.

PreparedStatement의 IN 파라미터를 사용해서 데이터베이스의 GEOMETRY 타입 칼럼에 데이터(NULL 포함)를 삽입하는 경우, AltibaseTypes.GEOMETRY 상수를 사용해서 데이터 타입을 반드시 명시해야 한다.

쿼리문에 직접 GEOMETRY 타입의 데이터를 표기하는 방법은 *Spatial SQL Reference*를 참고하도록 한다.

예제

다음은 JDBC 애플리케이션에서 GEOMETRY 타입의 칼럼에 데이터를 삽입하는 코드 예제이다.

```
int sSize = ... ;
byte[] sGeometryData = new byte[sSize];

Connection sConn = ... ;
PreparedStatement sPstmt = sConn.prepareStatement("INSERT INTO TEST_TABLE VALUES (?)");
sPstmt.setObject(1, sGeometryData, AltibaseTypes.GEOMETRY);
sPstmt.executeQuery();

...
```

LOB

이 절은 Altibase가 제공하는 LOB 타입의 데이터를 JDBC 애플리케이션에서 조작하는 방법을 설명한다.

전제 조건

- Altibase가 지원하는 LOB 데이터 타입은 BLOB 및 CLOB이 있으며, 각각 4GB-1byte의 최대 크기를 가질 수 있다. 단, JDK 1.6 이상에서만 가능하다.

LOB 데이터를 정상적으로 다루기 위해서는 세션의 autocommit 모드가 아래의 조건 중 하나를 만족해야 한다.

- Connection 객체의 setAutoCommit(false)을 사용해서 세션의 자동커밋을 해제(disable)하고 사용자가 수동으로 트랜잭션을 제어해야 한다.
- Clientside_auto_commit을 on으로 지정하여 JDBC 드라이버가 트랜잭션의 자동커밋을 제어하도록 한다.

BLOB 사용하기

아래는 JDBC 애플리케이션에서 BLOB 데이터를 조작하는 방법을 코드 예제로 보여준다.

PreparedStatement 객체를 통해서 BLOB 데이터 쓰기

다음은 예제에서 사용되는 테이블을 생성하는 구문이다.

```
CREATE TABLE TEST_TABLE ( C1 BLOB );
```

setBinaryStream 메소드와 InputStream객체 사용

```
InputStream sInputStream = ...
long sLength = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE VALUES (?)");
...
sPstmt.setBinaryStream(1, sInputStream, sLength);
...
sPstmt.execute();
...
```

JDK 1.5에서는 sPstmt를 AltibasePreparedStatement 타입으로 캐스팅 하면 long 타입의 길이 변수로 정의된 setBinaryStream() 메소드를 호출할 수 있다.

```
import Altibase.jdbc.driver.AltibasePreparedStatement;
...
((AltibasePreparedStatement)sPstmt).setBinaryStream(1, sInputStream, sLength);
...
```

setBinaryStream 메소드와 OutputStream 객체 사용

```
byte[] sBuf = ...
...
PreparedStatement sPstmt = connection().prepareStatement("SELECT * FROM TEST_TABLE FOR UPDATE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    Blob sBlob = sPstmt.getBlob(1);
    OutputStream sOutputStream = sBlob.setBinaryStream(1);
    sOutputStream.write(sBuf);
    sOutputStream.close();
    ...
}
...
sPstmt.execute();
...
```

setBlob 메소드와 Blob 객체 사용

```
java.sql.Blob sBlob = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setBlob(1, sBlob);
...
sPstmt.execute();
...
```

setObject 메소드와 Blob 객체 사용

```
java.sql.Blob sBlob = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setObject(1, sBlob);
...
sPstmt.execute();
...
```

setObject 메소드에 SQL 타입 지정

```
java.sql.Blob sBlob = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setObject(1, sBlob);
...
sPstmt.execute();
...
```

ResultSet 객체를 통해서 BLOB 데이터 쓰기

다음은 예제에서 사용되는 테이블을 생성하는 구문이다.

```
CREATE TABLE BLOB_TABLE ( BLOB_COLUMN BLOB );
```

updateBinaryStream 메소드와 InputStream 객체 사용

```

InputStream sInputStream = ...
long sLength = ...
...
PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN FROM BLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateBinaryStream(1, sInputStream, sLength);
    sRs.updateRow();
    ...
}
...

```

updateBlob 메소드와 Blob 객체 사용

```

java.sql.Blob sBlob = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN FROM BLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateBlob(1, sBlob), ;
    sRs.updateRow();
    ...
}

...

```

updateObject 메소드와 Blob 객체 사용

```

java.sql.Blob sBlob = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN FROM BLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateObject(1, sBlob);
    sRs.updateRow();
    ...
}

...

```

updateObject 메소드에 SQL 타입 지정

```

java.sql.Blob sBlob = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN FROM BLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateObject(1, sBlob, AltibaseTypes.BLOB);
    sRs.updateRow();
    ...
}

...

```

SELECT ... FOR UPDATE 구문으로 BLOB 데이터 갱신

```

byte[] sBytes = new byte[sLength];
...

PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN FROM BLOB_TABLE FOR UPDATE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    Blob sBlob = sRs.getBlob(1);
    sBlob.setBytes(0, sBytes);
    ...
}
...

```

BLOB 데이터 읽기

getBinaryStream 메소드와 InputStream 객체 사용

```

...
PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN
FROM BLOB_TABLE");
ResultSet sRs = sPstmt.executeQuery();
while(sRs.next())
{
    ...
    InputStream sInputStream = sRs.getBinaryStream(1);
    ...
}
...

```

getBlob 메소드와 InputStream 객체 사용

```

...
PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN
FROM BLOB_TABLE");
ResultSet sRs = sPstmt.executeQuery();
while(sRs.next())
{
    ...
    Blob sBlob = sRs.getBlob(1);
    InputStream sInputStream = sBlob.getBinaryStream();
    ...
}
...

```

getBlob 메소드와 byte 배열 사용

```

...
final int sReadLength = 100;

PreparedStatement sPstmt = connection().prepareStatement("SELECT BLOB_COLUMN FROM BLOB_TABLE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    Blob sBlob = sRs.getBlob(1);
    long sRemains = sBlob.length();
    long sOffset = 0;
    while(sRemains > 0)
    {
        byte[] sReadBytes = sBlob.getBytes(sOffset, sReadLength);
        sRemains -= sReadBytes.length;
        sOffset += sReadBytes.length;
        ...
    }
    ...
}
...

```


BLOB 데이터 변경하기

Truncation

```
Statement sStmt = ...

ResultSet sRs = sStmt.executeQuery("SELECT * FROM t1 FOR UPDATE");

while(sRs.next())
{
    ...
    int sLength = ... ;
    Blob sBlob = sRs.getBlob(2);

    // After executing this method
    // sBlob.length() == sLength
    sBlob.truncate(sLength);
}

...
```

CLOB 사용하기

아래는 JDBC 애플리케이션에서 CLOB 데이터를 조작하는 방법을 코드 예제로 보여준다.

PreparedStatement를 통해서 CLOB 데이터 쓰기

다음은 예제에서 사용되는 테이블을 생성하는 구문이다.

```
CREATE TABLE TEST_TABLE ( C1 BLOB );
```

setCharacterStream 메소드와 Reader 객체 사용

```
Reader sReader = ...
long sLength = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setCharacterStream(1, sReader, sLength);
...
sPstmt.execute();
...
```

JDK 1.5에서는 sPstmt를 AltibasePreparedStatement 타입으로 캐스팅 하면 long 타입의 길이 변수로 정의된 setCharacterStream() 메소드를 호출할 수 있다.

```
import Altibase.jdbc.driver.AltibasePreparedStatement;
...
((AltibasePreparedStatement)sPstmt).setCharacterStream(1, sReader, sLength);
...
```

setCharacterStream 메소드와 Writer 객체 사용

```

char[] sBuf = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT * FROM TEST_TABLE FOR UPDATE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    Clob sClob = sPstmt.getClob(1);
    Writer sWriter = sClob.setCharacterStream(1);
    sWriter.write(sBuf);
    sWriter.close();
    ...
}

...

sPstmt.execute();

...

```

setClob 메소드와 Clob 객체 사용

```

java.sql.Clob sClob = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setClob(1, sClob);
...
sPstmt.execute();
...

```

setObject 메소드와 Clob 객체 사용

```

java.sql.Clob sClob = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setObject(1, sClob);
...
sPstmt.execute();
...

```

setObject 메소드에 SQL 타입 지정

```

java.sql.Clob sClob = ...
...
PreparedStatement sPstmt = connection().prepareStatement("INSERT INTO TEST_TABLE
VALUES (?)");
...
sPstmt.setObject(1, sClob, AltibaseTypes.Clob);
...
sPstmt.execute();
...

```

ResultSet 객체를 사용해서 CLOB 데이터 쓰기

다음은 예제에서 사용되는 테이블을 생성하는 구문이다.

```

CREATE TABLE CLOB_TABLE ( CLOB_COLUMN CLOB );

```

updateCharacterStream 메소드와 Reader 객체 사용

```

Reader sReader = ...
long sLength = ... // The length of source from which Reader is linked

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateCharacterStream(1, sReader, sLength);
    sRs.updateRow();
    ...
}

...

```

updateClob 메소드와 Clob 객체 사용

```

java.sql.Clob sClob = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateClob(1, sClob);
    sRs.updateRow();
    ...
}

...

```

updateObject 메소드와 Clob 객체 사용

```

java.sql.Clob sClob = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateObject(1, sClob);
    sRs.updateRow();
    ...
}

...

```

updateObject 메소드에 SQL 타입 지정

```

java.sql.Clob sClob = ...

...

PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE", ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    sRs.updateObject(1, sClob, AltibaseTypes.CLOB);
    sRs.updateRow();
    ...
}

...

```

SELECT ... FOR UPDATE 구문으로 CLOB 데이터 삽입

```

...

String sStr = ... ;
PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE FOR UPDATE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    Clob sClob = sRs.getClob(1);
    sClob.setString(0, sStr);
    ...
}

...

```

CLOB 데이터 읽기

getCharacterStream 메소드와 Reader 객체 사용

```

...
PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    Reader sReader = sRs.getCharacterStream(1);
    ...
}

...

```

getClob 메소드와 Reader 객체 사용

```

...
PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    Clob sClob = sRs.getClob(1);
    Reader sReader = sClob.getCharacterStream();
    ...
}

...

```

getClob 메소드와 String 객체 사용

```

...
final int sReadLength = 100;

PreparedStatement sPstmt = connection().prepareStatement("SELECT CLOB_COLUMN FROM CLOB_TABLE");

ResultSet sRs = sPstmt.executeQuery();

while(sRs.next())
{
    ...
    Clob sClob = sRs.getClob(1);
    long sRemains = sClob.length();
    long sOffset = 0;
    while(sRemains > 0)
    {
        String sStr = sClob.getSubString(sOffset, sReadLength);
        sRemains -= sStr.length();
        sOffset += sStr.length();
        ...
    }
    ...
}

...

```

CLOB 데이터 변경하기

Truncation

```

Statement sStmt = ...

ResultSet sRs = sStmt.executeQuery("SELECT * FROM t1 FOR UPDATE");

while(sRs.next())
{
    ...
    int sLength = ... ;
    Clob sClob = sRs.getClob(2);

    // After executing this method
    // sClob.length() == sLength
    sClob.truncate(sLength);
}

...

```

자원 해제하기

많은 수의 LOB 객체를 사용해서 데이터를 획득하는 JDBC 애플리케이션의 경우, 반드시 획득한 LOB 객체를 해제하여야 한다. 또한 트랜잭션을 커밋하는 것과 상관없이 LOB 객체는 명시적으로 해제되어야 한다.

아래는 Blob 객체를 해제하는 코드 예제이다.

```

...
Blob sBlob = sRs.getBlob(1);
// Freeing Lob Locator
((Altibase.jdbc.driver.AltibaseLob)sBlob).free();
...

```

Blob 객체를 free 메소드로 해제하면, 대응하는 Lob Locator가 서버에서 해제되므로 그 객체에 대해서는 더이상 연산 수행이 불가능하다.

아래는 Clob 객체를 해제하는 코드 예제이다.

```

...
Clob sClob = sRs.getClob(1);
// Freeing Lob Locator
((Altibase.jdbc.driver.AltibaseLob)sClob).free();
...

```

Clob 객체도 Blob과 마찬가지로 free 메소드로 해제하면, 대응하는 Lob Locator가 서버에서 해제되므로 그 객체에 대해서는 더 이상 연산 수행이 불가능하다.

아래는 BlobInputStream 객체와 BlobOutputStream 객체를 해제하는 코드 예제이다.

```
InputStream sInputStream = sRs.getBinaryStream(1);

// Freeing Lob Locator
((Altibase.jdbc.driver.BlobInputStream)sInputStream).freeLocator();

CallableStatement sCallStmt = aConn.prepareCall("INSERT INTO TEST_TABLE VALUES (?)");
sCallStmt.registerOutParameter(1, Types.BLOB);
sCallStmt.execute();

Blob sBlob = sCallStmt.getBlob(1);
OutputStream sOutputStream = sBlob.setBinaryStream(1);

// Freeing Lob Locator
((Altibase.jdbc.driver.BlobOutputStream)sOutputStream).freeLocator();
```

BlobInputStream 또는 BlobOutputStream 객체를 freeLocator 메소드로 해제하면, 대응하는 Lob Locator가 서버에서 해제되므로 그 객체에 대해서는 더이상 연산 수행이 불가능하다.

아래는 ClobReader 객체와 ClobWriter 객체를 해제하는 코드 예제이다.

```
Reader sClobReader = sRs.getCharacterStream(1);

// Freeing Lob Locator
((Altibase.jdbc.driver.ClobReader)sClobReader).freeLocator();

CallableStatement sCallStmt = aConn.prepareCall("INSERT INTO TEST_TABLE VALUES (?)");
sCallStmt.registerOutParameter(1, Types.CLOB);
sCallStmt.execute();

Clob sClob = sCallStmt.getClob(1);
Writer sClobWriter = sClob.setCharacterStream(1);

// Freeing Lob Locator
((Altibase.jdbc.driver.ClobWriter)sClobWriter).freeLocator();
```

ClobReader 또는 ClobWriter 객체를 freeLocator 메소드로 해제하면, 대응하는 Lob Locator가 서버에서 해제되므로 그 객체에 대해서는 더이상 연산 수행이 불가능하다.

제약 사항

clientside_auto_commit을 on으로 지정하여 JDBC 드라이버가 트랜잭션의 자동커밋을 제어하도록 하여도 LOB 데이터를 다루는 것에는 아래와 같은 제약이 여전히 존재한다.

ResultSet 객체(커서)를 통해 가져온 LOB 데이터를 커서가 닫히기 전에 다른 Statement의 executeUpdate() 메소드에 사용하면 Lob locator가 사라지기 때문에 그 커서로부터 더 이상의 fetch가 불가능해진다. 아래는 이러한 오류 상황이 발생할 수 있는 예제 코드이다.

```

PreparedStatement sPreStmt =
    sCon.prepareStatement( "INSERT INTO TEST_TEXT " +
        "VALUES ( ?, ?, ?, ? )" );
Statement sStmt = sCon.createStatement();
ResultSet sRS = sStmt.executeQuery( "SELECT ID, TEXT " +
    " FROM TEST_SAMPLE_TEXT " );

while ( sRS.next() ) -> (2)
{
    sID    = sRS.getInt( 1 );
    sClob  = sRS.getClob( 2 );
    switch ( sID )
    {
        case 1 :
            sPreStmt.setInt( 1, 1 );
            sPreStmt.setString( 2, "Altibase Greetings" );
            sPreStmt.setClob( 3, sClob );
            sPreStmt.setInt( 4, (int)sClob.length() );
            break;
        case 2 :
            sPreStmt.setInt( 1, 2 );
            sPreStmt.setString( 2, "Main Memory DBMS" );
            sPreStmt.setClob( 3, sClob );
            sPreStmt.setInt( 4, (int)sClob.length() );
            break;
        default :
            break;
    }
    sPreStmt.executeUpdate(); -> (1)
}

```

(1): ResultSet sRS가 열려 있는 상태에서 sPreStmt.executeUpdate()를 호출하면, JDBC 드라이버가 트랜잭션을 자동으로 commit하면서 sClob의 Lob locator가 사라진다.

(2): Lob locator가 사라졌으므로 sRs.next()에서 exception이 발생할 수 있다.

따라서, 이러한 로직에서 LOB 데이터를 다룰 때는 먼저 setAutoCommit(false)를 호출하여 세션의 자동커밋을 해제해야 한다.

Autocommit 제어

Altibase JDBC 애플리케이션에서는 auto_commit 연결 속성 또는 JDBC Connection 객체의 setAutoCommit 메소드를 사용해서 세션의 자동커밋(autocommit) 모드를 정할 수 있다. 사용자가 auto_commit=true 또는 setAutoCommit(true) 메소드를 사용해서 자동커밋을 설정(enable)하면, Altibase 서버가 트랜잭션을 자동으로 커밋한다.

연결 속성 clientside_auto_commit을 이용해서 자동커밋 모드를 설정할 수도 있다. 연결 속성 clientside_auto_commit을 on으로 지정하면, Altibase 서버 대신에 JDBC 드라이버가 트랜잭션을 자동으로 커밋한다.

clientside_auto_commit을 off로 설정한 경우에는 세션의 자동커밋 모드가 setAutoCommit 메소드에 의해 결정된다.

세션의 자동커밋을 해제(disable)하려면 setAutoCommit(false)을 호출하면 된다.

clientside_auto_commit=on인 세션에서도 setAutoCommit(false)을 호출하면 자동커밋은 해제되며, JDBC 드라이버의 자동 커밋 모드로 전환하려면 setAutoCommit(true)를 호출하면 된다.

자동커밋이 해제되면, 사용자가 commit() 또는 rollback() 메소드를 사용해서 트랜잭션을 수동으로 커밋하거나 롤백해야 한다.

자동커밋 모드를 설정하고 해제하는 방법을 정리하면 아래의 표와 같다.

Autocommit 모드	설정 방법
서버에서 트랜잭션 자동커밋	auto_commit=true(또는 미지정) or setAutoCommit(true)
JDBC 드라이버에서 트랜잭션 자동커밋	auto_commit=true(또는 미지정) and clientside_auto_commit=on

Autocommit 모드	설정 방법
자동커밋 해제	auto_commit=false or setAutoCommit(false)

BIT, VARBIT

이 절은 BIT, VARBIT 타입의 데이터를 JDBC 애플리케이션에서 조작하는 방법과 주의사항을 설명한다.

사용 방법

Altibase JDBC 애플리케이션에서는 Java BitSet 클래스를 사용하여 BIT, VARBIT 타입의 데이터를 조작할 수 있다.

PreparedStatement의 IN 파라미터를 사용할 때는 Types.BIT로 타입을 지정하거나, 지정하지 않을 때에는 BitSet 또는 문자열로 값을 지정할 수 있다.

주의사항

비트가 '0'으로 끝나는 BIT, VARBIT 값을 만들려면, 특정 길이의 비트 값으로 구성할 수 있는 AltibaseBitSet 클래스 또는 문자열의 값을 사용해야 한다. Java BitSet은 어떤 비트를 set()했는지 기억하도록 구현된 것으로, 0으로 구성된 특정 길이의 BIT 값은 만들 수 없기 때문이다.

예제

다음은 JDBC 애플리케이션에서 BIT, VARBIT 타입의 칼럼에 데이터를 삽입하는 코드 예제이다.

```

...
BitSet sBitSet1 = new BitSet();
sBitSet1.set(1);
BitSet sBitSet2 = new AltibaseBitSet( 5 );
sBitSet1.set(2);

PreparedStatement sPstmt = sConn.prepareStatement("INSERT INTO TEST_TABLE VALUES (?)");
sPstmt.setObject(1, sBitSet1, Types.BIT);
sPstmt.executeUpdate();
sPstmt.setObject(1, sBitSet2);
sPstmt.executeUpdate();
sPstmt.setObject(1, "0110100", Types.BIT);
sPstmt.executeUpdate();
...

```

JDBC 로깅

JDBC 로깅(JDBC Logging)은 Altibase JDBC 드라이버에서 발생하는 각종 로그를 기록하는 것을 의미하며, java.util.logging 패키지를 이용하여 관련된 로그를 기록할 수 있다. 이 절에서는 JDBC 로깅을 하기 위한 설정 방법 및 사용법을 설명한다.

JDBC 로깅 설정 방법

JDBC 드라이버로부터 로그를 남기려면 로깅 기능이 추가된 JDBC jar 파일을 사용해야 한다. 그리고 ALTIBASE_JDBC_TRACE 환경 변수에서 로깅 기능을 활성화한 후 사용하면 된다.

JRE버전

JRE 1.5 이상이 설치되어야 JDBC 로깅을 사용할 수 있으며, 이외의 라이브러리는 필요하지 않다.

CLASSPATH 설정

로깅을 사용하려면 CLASSPATH 환경변수에 Altibase_t.jar 파일을 추가해야 한다.

ex) 유닉스 환경에서 bash 셸을 사용하는 경우


```
export CLASSPATH=$ALTIBASE_HOME/lib/Altibase_t.jar:.$CLASSPATH
```

로깅 활성화

JVM 파라미터를 이용하여 ALTIBASE_JDBC_TRACE 환경 변수를 아래와 같이 TRUE로 설정하면, 클라이언트의 프로그램 수정 없이 글로벌하게 로깅 기능이 활성화된다. 그러나 ALTIBASE_JDBC_TRACE의 변경 값을 적용하려면, 클라이언트 프로그램을 재시작해야 한다.

```
java -DALTIMBASE_JDBC_TRACE=true ...
```

JDBC 로깅 사용법

java.util.logging 파일 설정

java.util.logging 설정은 \$JRE_HOME/lib/logging.properties 파일에서 설정하거나, 아래와 같이 java.util.logging.config.file를 이용하여 별도로 설정할 수 있다.

```
java -Djava.util.logging.config.file=$ALTIBASE_HOME/sample/JDBC/Logging/logging.properties -DALTIMBASE_JDBC_TRACE=true ...
```

알티베이스는 \$ALTIBASE_HOME/sample/JDBC/Logging 디렉토리에 logging.properties 샘플 파일을 제공하고 있으며, 이를 참조하여 사용하거나, 직접 설정 파일을 생성하여 Djava.util.logging.config.file 프로퍼티를 통해 사용하면 된다.

Logger의 종류

로거는 트리 구조로 구성되어 있으며 부분별로 로그의 양이나 셋팅을 조절할 때 사용한다. 알티베이스 JDBC 드라이버에서 지원하는 Logger의 종류는 아래와 같다.

Logger	설명
altibase.jdbc	알티베이스 JDBC 메시지 (Connection, Statement, PreparedStatement 등의 JDBC API call)
altibase.jdbc.pool	커넥션 풀 관련 메시지
altibase.jdbc.rowset	ResultSet 관련 메시지
altibase.jdbc.xa	xa 관련 메시지
altibase.jdbc.failover	failover 관련 메시지
altibase.jdbc.cm	CmChannel 네트워크 패킷 메시지

Logger 레벨

logger에 레벨을 설정하면 보다 상세하게 로그의 양을 조절할 수 있다. 알티베이스 JDBC 드라이버에서 제공하는 레벨은 아래와 같으며, SEVERE에서 FINEST으로 갈수록 상세한 로그를 남긴다. CONFIG레벨을 설정하면 SEVERE, WARNING, INFO, CONFIG 레벨의 로그들이 남는다.

Logger 레벨	설명
OFF	로그를 남기지 않는다.
SEVERE	SQLException이나 내부 에러가 발생했을 때 해당 에러만 로그에 남긴다.
WARNING	SQLWarning을 로그에 남긴다.
INFO	JDBC 드라이버 내부적으로 특정 객체의 상태를 모니터링하여 로그에 남긴다.
CONFIG	JDBC 드라이버 내부적으로 어떤 SQL문이 실행되는지 확인할 때 주로 사용한다. PreparedStatement같은 경우 prepare할 때 sql이 표시되며, Statement는 execute 할 때 sql이 표시된다. 이때 sql이 실행되는데 걸린 시간이 milli sec 단위로 같이 표시된다.
FINE	표준 JDBC API에 진입하고 리턴될 때 해당 인자 값과 반환 값을 로그에 남긴다. API에 진입할 때마다 로그가 남기 때문에 로그의 양이 많아질 수 있으며, Connection이나 Statement가 close되는데 걸린 시간이 추가로 표시된다.

Logger 레벨	설명
FINEST	JDBC 드라이버와 알티베이스 서버가 주고 받는 패킷 정보를 로그로 남긴다. 로그의 양이 가장 많다.

logging.properties

다음은 로그 레벨을 CONFIG로 하고, 네트워크 패킷 로그를 남기는 샘플 logging.properties이며, \$ALTIBASE_HOME/sample/JDBC/Logging/logging.properties 에서도 내용을 참조할 수 있다.

```
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler --> 기본적인 handler로 FileHandler와 ConsoleHandler를 추가한다.

.level = CONFIG --> 루트 logger 레벨을 CONFIG로 셋팅한다.

# default file output is in same directory.
java.util.logging.FileHandler.level = CONFIG
java.util.logging.FileHandler.pattern = ./jdbc_trace.log
java.util.logging.FileHandler.limit = 10000000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.append = false
java.util.logging.FileHandler.formatter = Altibase.jdbc.driver.logging.SingleLineFormatter
--> java.util.logging.FileHandler의 기본셋팅을 설정하는 부분이다. sql정보만 보면 되기때문에 FileHandler의 레벨을 CONFIG으로 설정했다.

java.util.logging.ConsoleHandler.level = CONFIG
java.util.logging.ConsoleHandler.formatter = Altibase.jdbc.driver.logging.SingleLineFormatter
--> java.util.logging.ConsoleHandler를 설정하는 부분이다. 로그를 한 줄로 출력하기 위해 SingleLineFormatter를 사용했다.

altibase.jdbc.level = CONFIG
altibase.jdbc.rowset.level = SEVERE
altibase.jdbc.cm.level = FINEST
altibase.jdbc.cm.handlers = Altibase.jdbc.driver.logging.MultipleFileHandler
#altibase.jdbc.cm.handlers = java.util.logging.FileHandler
--> altibase jdbc logger를 설정하는 부분으로 레코드 셋의 정보는 제외해야 하기 때문에 rowset의 레벨을 SEVERE로 설정했다. 네트워크 패킷 정보는 로그로 남겨야 하기 때문에 cm의 레벨을 F1

Altibase.jdbc.driver.logging.MultipleFileHandler.level = FINEST
Altibase.jdbc.driver.logging.MultipleFileHandler.pattern = ./jdbc_net_%.log
Altibase.jdbc.driver.logging.MultipleFileHandler.limit = 10000000
Altibase.jdbc.driver.logging.MultipleFileHandler.count = 1
Altibase.jdbc.driver.logging.MultipleFileHandler.formatter = java.util.logging.XMLFormatter
--> MultipleFileHandler를 설정하는 부분으로 pattern에 jdbc_net_%.log를 사용해 세션의 아이디별로 파일이 생성되도록 설정했다. 또한 formatter로 XMLFormatter를 사용해 XML형태의 파일여
```

Hibernate

Altibase 는 비표준 SQL 을 제공하며, Hibernate 는 이러한 기능을 수행할 수 있도록 Dialect 클래스를 지원한다. Hibernate 에서 Altibase 를 연동하려면 Altibase 의 JDBC Driver 를 설정하고, Hibernate 의 configuration 에 AltibaseDialect.class 를 지정해야 한다.

AltibaseDialect

Hibernate 가 공식적으로 제공하는 라이브러리는 AltibaseDialect.class 를 포함하지 않기 때문에 AltibaseDialect.java 파일 (필요에 따라 AltibaseLimitHandler.java 포함)을 컴파일하고 Hibernate 가 제공하는 파일에 포팅해야 사용할 수 있다. AltibaseDialect.java 파일과 AltibaseLimitHandler.java 파일은 Altibase Github 사이트에서 제공한다. 상세한 사용 방법은 AltibaseDialect 포팅 방법 (https://github.com/ALTIBASE/hibernate-orm/blob/master/ALTIBASE_DIALECT_PORTING.md) 을 참고한다.

Lob 관련 속성

Lob 컬럼 값이 null 일때 Hibernate는 JDBC 스펙에 따라 ResultSet.getBlob(), ResultSet.getClob()이 null을 리턴할 것을 가정하고 기능이 동작한다. 하지만 해당 인터페이스는 기존에 값이 null 이더라도 Lob 관련 객체가 리턴되었기 때문에 Hibernate에서 Lob 관련 기능을 사용하려면 아래 JDBC 연결 속성을 off로 설정하는 것이 권장된다.

lob_null_select

기본값	on
값의 범위	[on off]

기본값	on
필수 여부	No
설정 범위	세션
설명	lob 컬럼값이 null 일때 ResultSet.getBlob(), ResultSet.getClob()이 객체를 리턴하는지 여부

예제

lob_null_select 값이 off 인 경우 다음과 같이 getBlob(), getClob()을 한 후 null 처리를 해줘야 한다.

```

Blob sBlob = sRs.getBlob();
if (sBlob != null) // sBlob이 null인 경우 NullPointerException이 발생할 수 있다.
{
    long sLength = sBlob.length();
    System.out.println("blob length==>" + sLength);
}
...
Clob sClob = sRs.getClob();
if (sClob != null) // sClob이 null인 경우 NullPointerException이 발생할 수 있다.
{
    long sLength = sClob.length();
    System.out.println("clob length==>" + sLength);
}

```

Sharding

Properties

jdbc sharding 기능을 위해 다음 속성들이 추가되었다.

shard_transaction_level

기본값	1
값의 범위	[0 1]
필수 여부	No
설정 범위	세션
설명	샤드트랜잭션 레벨을 설정한다. 0 : single node transaction 1 : multiple node transaction

shard_conn_type

기본값	TCP
값의 범위	[TCP SSL IB] [1 6 8]
필수 여부	No
설정 범위	세션
설명	클라이언트와 데이터 노드의 네트워크 연결 방식을 결정한다. TCP(1) : tcp SSL(6) : ssl IB(8) : infiniband(JNI라이브러리 필요)

shard_lazy_connect

기본값	true
값의 범위	[true false]
필수 여부	No

기본값	true
설정 범위	세션
설명	<p>데이터노드와의 연결을 수립할 때 지연된 연결을 사용할지 여부를 결정한다.</p> <p>false : meta접속 후 바로 데이터노드들로 접속하고 prepare시에도 각 노드들로 prepare요청을 한꺼번에 보낸다.</p> <p>true : execute시 필요한 노드에 대해 연결하고 prepare요청을 보낸다.</p>

버전 확인

Altibase.jar 파일 하나에 sharding 기능이 통합되어 있으며, 다음과 같이 java -jar 를 해보면 sharding을 지원하는지 확인할 수 있다.

```
$ java -jar Altibase.jar
Altibase 7.2.0.0.0 with CMP 7.1.7 for JDBC 3.0 compiled with JDK 5(sharding included)
```

연결 설정

DriverManager

기존 jdbc url에 sharding prefix를 추가한다.

```
String sUrl = "jdbc:sharding:Altibase://127.0.0.1:20300/mydb"
Connection sCon = DriverManager.getConnection( sUrl, sProps);
...
```

DataSource

javax.sql.DataSource를 구현하고 있는 AltibaseShardingDataSource를 사용하면 된다.

```
String sUrl = "jdbc:sharding:Altibase://127.0.0.1:20300/mydb";
AltibaseShardingDataSource sDataSource = new AltibaseShardingDataSource();
sDataSource.setURL(sUrl);
sDataSource.setUser("sys");
sDataSource.setPassword("manager");
Connection sCon = sDataSource.getConnection();
...
```

Connection Pool 설정

Meta서버와의 접속에 커넥션풀링 기능을 사용할 수 있다. 커넥션풀로는 DBCP와 같은 오픈소스 커넥션풀이나 AltibaseShardingConnectionPoolDataSource를 사용할 수 있으며 데이터 노드와의 접속은 드라이버 자체적으로 캐싱을 하고 있기 때문에 pooling이 되지 않는다.

DBCP

dbcp는 다음과 같이 DriverClassName을 이용해 connection pool을 생성할 수 있다.

Simple JDBC

```
public DataSource createDataSourceNode(String aUrl)
{
    BasicDataSource sResult = new BasicDataSource(); // Apache DBCP Connection Pool DataSource
    sResult.setDriverClassName(Altibase.jdbc.driver.AltibaseDriver.class.getName());
    sResult.setUrl(aUrl); // with shard prefix (ex : jdbc:sharding:Altibase://127.0.0.1:20300/mydb)
    sResult.setUsername("sys");
    sResult.setPassword("manager");
    return sResult;
}
```

Spring

```
<bean id="shardDataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="Altibase.jdbc.driver.AltibaseDriver" />
    <property name="url" value="jdbc:sharding:Altibase://127.0.0.1:20300/mydb" />
    <property name="username" value="sys" />
    <property name="password" value="manager" />
    <property name="initialSize" value="3" />
    <property name="maxActive" value="10" />
</bean>
```

ConnectionPoolDataSource

shardjdbc는 javax.sql.ConnectionPoolDataSource를 구현하고 있는 AltibaseShardingConnectionPoolDataSource를 제공한다. 사용법은 기존 AltibaseConnectionPoolDataSource와 같으며 오픈소스 커넥션풀을 사용할 수 없는 환경에서 이용할 수 있다.

```
private Connection getConnection() throws SQLException
{
    String sURL = "jdbc:sharding:Altibase://127.0.0.1/mydb";
    AltibaseShardingConnectionPoolDataSource sDataSource = new AltibaseShardingConnectionPoolDataSource();
    sDataSource.setURL(sURL);
    PooledConnection sPooledConn = sDataSource.getPooledConnection("SYS", "MANAGER");
    return sPooledConn.getConnection();
}
```

환경변수

병렬로 shard sql을 실행할때 사용할 쓰레드풀의 설정을 환경변수로부터 읽어들이 수 있다.

Name	Description	Default
SHARD_JDBC_POOL_CORE_SIZE	pool에 유지할 쓰레드의 갯수(idle포함)	CPU코어수
SHARD_JDBC_POOL_MAX_SIZE	pool에서 허용 할 수 있는 최대 스레드 수	128
SHARD_JDBC_IDLE_TIMEOUT	현재 풀에 core size 수보다 많은 thread가 있는 경우, 초과한 만큼의 thread는 IDLE 상태가 되어 있는 시간이 idle_timeout 를 넘으면 종료한다.	10(분)

로깅

로깅과 관련된 셋팅도 환경변수로 가능하다. 기본적으로 SHARD_JDBC_TRCLOG_LEVEL 환경변수를 INFO나 SEVERE로 설정하면 로그를 남길 수 있다.

Name	Description	Default
SHARD_JDBC_TRCLOG_LEVEL	shardjdbc 로그 레벨	OFF
SHARD_JDBC_TRCLOG_PRINT_STDERR	부모로거에 넘길지 여부. TRUE로 설정하면 보통 콘솔에도 로그가 남는다.	FALSE

로그 레벨

원래 JDK Logging은 SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST 순으로 레벨셋팅이 가능하지만 shardjdbc는 이중 SEVERE와 INFO레벨 둘만 사용한다

Name	Description
INFO	sharding과 관련된 객체들의 정보가 로깅에 포함된다.
SEVERE	정상적인 상황에서는 로그가 남지않고 exception이 올라왔을때 해당 exception의 정보가 로그로 남는다.

로그 파일 경로

환경변수여부	경로
환경변수에 ALTIBASE_HOME이 있을 때	\${ALTIBASE_HOME}/trc/shardjdbc.trc
환경변수에 ALTIBASE_HOME이 없을 때	클라이언트 실행 경로

로그 파일 Rotate

기본 로그파일 하나당 사이즈는 약 15메가 이고 크기를 넘어가면 5개까지 rotate된다.

Failover

응용프로그램 가이드

Altibase Sharding 환경에서는 여러 샤드 노드에서 수행중인 트랜잭션 및 커넥션이 있으며, 이들은 최적화 과정을 거쳐서 샤드 라이브러리 혹은 서버에서 내부적으로

처리된다. 그러므로 특정 노드의 장애나 접속 에러 시에도 일부 커넥션이 남아 있거나 트랜잭션이 완전히 철회되지 않을 수 있다.

이러한 분산 환경에서 응용 프로그램이 트랜잭션 처리를 일관되게 하기 위해서는 NON-AUTOCOMMIT을 사용하여 다음의 가이드에 따라 작성되어야 Fail-Over가 정상적으로 처리될 수 있다.

다만, 응용 프로그램이 다수의 샤드 노드를 접근하지 않도록 설계된 경우에는 AUTOCOMMIT 모드를 사용할 수 있으나 이 경우에도 AUTOCOMMIT에 대한 가이드에 따라 처리되어야 Fail-Over 이후에 서비스가 정상적으로 처리될 수 있다.

CTF(Connection Time Failover)

CTF의 경우에는 데이터 베이스 연결이 되는지에 따라 성공 여부를 바로 알 수 있다.

단, ShardJDBC같은 경우 lazy 방식이 기본이기 때문에 트랜잭션이 이미 시작된 경우 execute 시점에 노드의 에러로 장애가 발생하더라도 CTF가 발생하지 않고 STF가 올라오게 된다. AUTOCOMMIT 상황이라면 이런 경우 CTF가 발생하게 되며, shard_lazy_connect 속성이 false일 때는 ShardCLI와 동일하게 동작하게 된다.

따라서 최초 사용자 커넥션 생성이 실패했을 때 전체 연결을 끊을 필요는 없으며 사용자 커넥션 접속만 재시도하면 된다.

STF(Service Time Failover)

ShardJDBC 경우는 prepare, execute, fetch등에서 ShardFailOverSuccessException 예외가 발생하면 STF가 성공한 것으로 판단할 수 있다.

- NON-AUTOCOMMIT 트랜잭션

ShardJDBC에서 예외가 발생하였을 때 다음의 순서로 에러로직을 처리한다.

1. STF가 성공한 경우(ShardFailOverSuccessException) Rollback을 수행하며 Rollback이 성공하면 트랜잭션 재시작 위치로 되돌아 가서 응용 프로그램 로직을 수행한다.
 1. 트랜잭션 재시작 위치는 prepare를 사용하는 경우 최초 prepare 이전, execute시는 execute 이전으로 하면 된다. 또한 이때 Bind는 다시 하지 않아도 된다.
 2. direct execute를 사용하는 경우에는 direct execute 이전으로 돌아가면 된다.
 3. STF 성공 후 Rollback을 하는 중에 다시 Fail-Over가 발생할 수 있으므로 이 경우에는 Rollback을 한번 더 수행한다.
2. STF가 실패하고 더 이상 서비스 가능한 가용 노드가 없는 경우(ShardFailoverIsNotAvailableException) 전체 노드에 대한 연결을 명시적으로 끊고 최초 연결부터 재시도 한다.
 1. 샤딩 환경에서는 다수의 노드에 접속이 이뤄져 있으므로 명시적으로 Connection.close()를 호출해야 모든 노드에 연결이 끊긴다
3. 그 외의 에러에 대해서는 응용 프로그램 에러 처리 로직을 수행한다.

- AUTOCOMMIT 트랜잭션

ShardJDBC에서 예외가 발생하였을 때 다음의 순서로 에러 로직을 처리한다.

1. STF가 성공한 경우(ShardFailOverSuccessException) 트랜잭션 재시작 위치로 되돌아 가서 응용 프로그램 로직을 수행한다.
 1. 트랜잭션 재시작 위치는 prepare를 사용하는 경우 최초 prepare 이전, execute시는 execute 이전으로 하면 된다. 또한 이때 Bind는 다시 하지 않아도 된다.
 2. direct execute를 사용하는 경우에는 direct execute 이전으로 돌아가면 된다.
2. STF가 실패하고 더 이상 서비스 가능한 가용 노드가 없는 경우 (ShardFailoverIsNotAvailableException) 전체 노드에 대한 연결을 명시적으로 끊고 최초 연결부터 재시도 한다.
 1. 샤딩 환경에서는 다수의 노드에 접속이 이뤄져 있으므로 명시적으로 Connection.close()를 호출해야 모든 노드에 연결이 끊긴다.

ShardJDBC Failover Sample Code

Altibase Sharding의 failover를 포함하는 ShardJDBC sample 코드는 \$ALTIBASE_HOME/sample/SHARD/Fail-Over/FailoverSample.java에 있으며, 해당 프로그램은 ShardJDBC를 이용하여 작성한 fail-over를 고려한 응용 프로그램 예제이다.

FailoverSample.java의 코드는 “CREATE TABLE T1 (I1 VARCHAR(20), I2 INTEGER);”의 구문으로 T1 테이블을 생성한 후 T1 테이블을 샤드 테이블로 등록하였다고 가정한다.

해당 프로그램은 최초 접속할 샤드 노드의 port와 alternate port를 순차적으로 입력받아 연결하고 응용 프로그램 로직을 수행하여 Direct-Execute 방식으로 데이터를 한 건 입력하고 Prepare-Execute 방식으로 질의를 수행한 후 검색된 데이터를 출력하는 프로그램이다.

예제 프로그램을 수행중에 특정 노드에 장애가 있는 경우 최초 접속시에는 에러가 발생하지 않지만 실행 중에는 STF를 통해 fail-over 된다.

주의할 점은, 접속을 재시도 하기 위해서는 남아 있을 수 있는 커넥션을 종료하기 위해서 Connection.close()를 명시적으로 호출해 주어야 하며, 에러가 발생했을 때에는 다수의 노드에서 발생했을 수 있는 에러를 확인하기 위해서 SQLException.getNextException()통해 모든 노드의 에러를 점검해야 한다.

에러 점검을 통해서 Service Time Fail-over가 되면 연결이 종료되지 않은 샤드 노드에 남아 있는 트랜잭션을 정리하기 위해서 Connection.rollback()을 호출해 준 후 다시 prepare 혹은 execute 로직으로 돌아가서 수행 한다.

자세한 코드 내용은 \$ALTIBASE_HOME/sample/SHARD/Fail-Over/FailoverSample.java를 참고한다.

제약사항

일반 Altibase jdbc 드라이버는 지원하는데 sharding jdbc 드라이버에서 지원하지 않는 기능은 다음과 같다.

Savepoint

- Savepoint 관련 기능은 지원하지 않는다.
 - java.sql.Connection
 - rollback(Savepoint aSavepoint)
 - setSavepoint()
 - setSavepoint(String aName)
 - releaseSavepoint(Savepoint aSavepoint)

Scrollable Statement

- Sharding의 특성상 ResultSetType은 FORWARD_ONLY만 지원한다.
 - java.sql.Connection
 - createStatement(int aResultSetType, int aResultSetConcurrency, int aResultSetHoldability)
 - prepareStatement(String aSql, int aResultSetType, int aResultSetConcurrency)

Lob

- Multiple node lob 데이터 처리
 - java.sql.PreparedStatement
 - 다수의 노드에 대해 lob데이터를 insert 또는 update 하는 기능
 - setXXX 호출 후 execute할때 NOT SUPPORTED 에러가 발생한다.
 - setCharacterStream(int aParameterIndex, Reader aReader, int aLength)
 - setBinaryStream(int aParameterIndex, InputStream aValue, int aLength)
 - setAsciiStream(int aParameterIndex, InputStream aValue, int aLength)
 - setBlob(int aIndex, Blob aValue)
 - setClob(int aIndex, Clob aValue)
- 서버사이드 lob 데이터 처리
 - 서버에서 lob을 지원하지 않는다.

Statement Batch

- Statement가 실행될때마다 노드를 결정해야 하기때문에 batch기능은 PreparedStatement 에서만 지원
 - java.sql.Statement
 - addBatch(String aSql)
 - clearBatch()
 - executeBatch()

XADataSource

- XA관련 인터페이스는 지원하지 않는다.
 - javax.sql.XADataSource
 - getConnection()
 - getConnection(String user, String password)

SQL Plan

SQL 실행 계획을 문자열로 가져오는 기능을 비표준 API로 제공한다. 실행 계획은 Altibase가 명령문을 실행하기 위해 수행하는 작업의 순서를 나타낸다. Option에는 ON, OFF, 또는 ONLY가 올 수 있으며 기본 설정값은 OFF이다.

사용법

실행 계획을 가져오기 위해서는 SQL 문을 수행하기 전에 AltibaseConnection 객체의 setExplainPlan(byte aExplainPlanMode) 메소드를 호출해, 어떤 내용의 실행 계획을 가져올지 지정해야 한다. 지정 가능한 aExplainPlanMode 옵션은 아래 표에 기술되어 있다. AltibaseStatement 객체에 SQL 문을 입력 후, getExplainPlan() 메서드를 호출하여 문자열 형태의 실행 계획을 반환 받을 수 있다.

인자

속성	속성값	내용
AltibaseConnection.EXPLAIN_PLAN_OFF	0	SELECT 문 실행 후 Plan Tree 정보는 보여주지 않고 결과 레코드만 보여준다.
AltibaseConnection.EXPLAIN_PLAN_ON	1	SELECT 문 실행 후 결과 레코드와 함께 Plan Tree의 정보를 보여준다. Plan tree에는 레코드 접근 횟수 및 튜플이 점유한 메모리 양, 비용 등이 출력된다.
AltibaseConnection.EXPLAIN_PLAN_ONLY	2	SELECT 문 실행 후 결과 레코드와 함께 Plan Tree의 정보를 보여준다. EXPLAN PLAN = ONLY인 경우 질의 실행 없이 실행 계획만 생성하므로, ACCESS 항목과 같이 실제 실행 후 그 값이 결정되는 항목들은 물음표("??")로 표시된다.

코드 예제

```
AltibaseConnection sConn = (AltibaseConnection)DriverManager.getConnection(sURL, sProps);
sConn.setExplainPlan(AltibaseConnection.EXPLAIN_PLAN_ONLY);
AltibaseStatement sStmt = (AltibaseStatement)sConn.prepareStatement("SELECT sysdate FROM dual");
System.out.println(sStmt.getExplainPlan());
```

코드 결과

```
-----
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 8, COST: 0.01 )
SCAN ( TABLE: DUAL, FULL SCAN, ACCESS: ??, COST: 0.01 )
-----
```

4.Tips & Recommendation

이 장은 Altibase JDBC 드라이버를 효율적으로 사용하기 위한 방법을 제시한다.

성능을 위한 팁

아래는 JDBC 애플리케이션의 성능을 향상시키기 위해 염두에 두어야 할 몇 가지 사항이다.

- JDBC 애플리케이션에서 LOB 데이터를 조작할 때에는 Stream 또는 Writer 객체를 사용할 것을 권장한다. 만약 사용할 LOB 데이터의 크기가 8192바이트 이하일 경우, Lob_Cache_Threshold 연결 속성을 적절한 값으로 설정하도록 한다.
- 하나의 Connection 객체에 한 가지 작업을 수행할 것을 권장한다. 예를 들어, 한 Connection 객체에 여러 개의 Statement 객체를 생성하여 작업을 수행하면 성능 저하가 발생할 수도 있다.

- Connection 객체의 생성, 소멸이 빈번할 경우, 미들웨어(WAS)에서 제공하는 Connection Pool을 사용할 것을 권장한다. Connection을 맺고 끊는 작업은 다른 연산에 비해 비교적 비용이 크기 때문이다.

5.에러 메시지

이 장은 Altibase JDBC 드라이버를 사용하면서 발생할 수 있는 오류의 SQL State를 기술한다.

SQL States

SQLSTATE에 반환되는 문자열 값은 클래스를 나타내는 처음 2개의 문자와 그 뒤에 서브클래스를 나타내는 3개의 문자로 이루어진다. 클래스는 상태를 나타내고 서브클래스는 세부 상태를 나타낸다.

아래는 Altibase JDBC 드라이버에서 발생할 수 있는 SQLState의 종류와 그 의미를 간략하게 정리한 표이다.

Condition	Class	Subcondition	Subclass
connection exception	08		
		Communication link failure	S01
		Invalid packet header version	P01
		Fail-Over completed	F01
		Invalid format for alternate servers	F02
		Invalid packet next header type	P02
		Invalid packet sequence number	P03
		Invalid packet serial number	P04
		Invalid packet module ID	P05
		Invalid packet module version	P06
		Invalid operation protocol	P07
		Invalid property id: %s	P08
		Invalid connection URL	U01
		Unknown host	H01
		There are no available data source configurations	D01
		connection failure	006
		SQL-client unable to establish SQL-connection	001
		Unsupported Algorithm	K01
		Could not create keystore instance	K02
		Could not load keystore	K03
		Invalid keystore url	K04
		Could not open keystore file	K05
		Key management exception occurred	K06
		Could not retrieve key from keystore	K07
		Default algorithm definition invalid	K08

Condition	Class	Subcondition	Subclass
		Mandatory properties that are supported for the client version are not supported for the server version.	M01
dynamic SQL error	07	This statement returns result set(s)	R01
		Invalid query string	Q01
		Statement has not been executed yet	S01
no data	02		
		The sql statement does not produce a result set	001
warning	01		
		cursor operation conflict	001
		Invalid connection string attribute	S00
		Batch update exception occurred: %s	B00
		There are no batch jobs	B01
		There are existing some batch jobs	B02
		The query cannot be executed while batch jobs are executing	B03
		Binding cannot be permitted during executing batch jobs	B04
		Fetch operation cannot be executed during batch update	B05
		There are too many added batch jobs	B31
		Statement has already been closed	C01
		The result set has already been closed	C02
		The stream has already been closed	C03
		additional result sets returned	00D
		This result set doesn't retain data	R01
		Attempt to return too many rows in only one fetch operation	R02
		Option value changed	S02
		Invalid value for bitset	V01
feature not supported	0A		000
		Cannot change the name of the database	C01
		The read only mode in transaction cannot be supported	C02
		Not supported operation on forward only mode	T01
		Not supported operation on read only mode	T02
		violate the JDBC specification	V01
syntax error or access rule violation	42	Invalid type conversion	001
		Column not found	S22
JDBC internal error	JI		000
		Overflow occurred on dynamic array which is defined by JDBC	D01
		Underflow occurred on dynamic array which is defined by JDBC	D02
		This result set was created by JDBC driver's internal statement	D03

Condition	Class	Subcondition	Subclass
		Connection thread is interrupted	D04
		Remaining data exceeds the max size of the primitive type	D05
		Packet Operation has been twisted	P01
		Invalid method invocation	I01
cardinality violation	21	Insert value list does not match column list	S01
data exception	22		000
		null value not allowed	004
		invalid parameter value	023
		Insufficient number of parameters	P01
		IN type parameter needed	P02
		OUT type parameter needed	P03
		There is no column which needs to bind parameter.	P04
		Statement ID mismatch	V01
		Error occured from InputStream	S01
		The length between actual lob data and written lob data into the communication buffer is different.	L01
invalid transaction state	25		
		branch transaction already active	002
savepoint exception	3B		
		Cannot set savepoint at auto-commit mode	S01
		Invalid savepoint name	V01
		Invalid savepoint	V02
		Not supported operation on named savepoint	N01
		Not supported operation on un-named savepoint	N02
invalid schema name	3F		000
		Explain Plan Error	EP
		EXPLAIN PLAN is set to OFF	S01
General Error	HY		
		There are too many allocated statements	000
		Associated statement is not prepared	007
		Attribute cannot be set now	011
		Invalid string or buffer length	090
		Invalid cursor position	109
		Empty ResultSet	R01
		Timeout expired	T00
XA error	XA		
		XA open failed	F01
		XA close failed	F02

Condition	Class	Subcondition	Subclass
		XA recover failed	F03

A.부록: 데이터 타입 매핑

이 부록은 Altibase의 데이터 타입과 JDBC 표준 데이터 타입, Java 데이터 타입간에 호환 여부를 기술한다.

데이터 타입 매핑

아래의 표는 JDBC 데이터 타입, Altibase JDBC의 데이터 타입, 및 Java 언어의 타입 간에 기본적으로 매핑되는 관계를 보여준다.

JDBC 타입	Altibase 타입	Java 타입
CHAR	CHAR	String
VARCHAR	VARCHAR	String
LONGVARCHAR	VARCHAR	String
NUMERIC	NUMERIC	BigDecimal
DECIMAL	NUMERIC	BigDecimal
BIT	VARBIT	BitSet
BOOLEAN	-	-
TINYINT	SMALLINT	Short
SMALLINT	SMALLINT	Short
INTEGER	INTEGER	Integer
BIGINT	BIGINT	Long
REAL	REAL	Float
FLOAT	FLOAT	BigDecimal
DOUBLE	DOUBLE	Double
BINARY	BYTE	byte[]
VARBINARY	BLOB	Blob
LONGVARBINARY	BLOB	Blob
DATE	DATE	Timestamp
TIME	DATE	Timestamp
TIMESTAMP	DATE	Timestamp
CLOB	CLOB	Clob
BLOB	BLOB	Blob
ARRAY	-	-
DISTINCT	-	-
STRUCT	-	-
REF	-	-
DATALINK	-	-
JAVA_OBJECT	-	-

JDBC 타입	Altibase 타입	Java 타입
NULL	-	null
-	GEOMETRY	byte[]

Java 데이터형을 데이터베이스 데이터형으로 변환하기

아래의 표는 setObject 메소드를 사용해서 파라미터에 객체를 설정할 경우, 각 객체 별로 어떠한 데이터베이스 데이터입으로 변환이 가능한지를 보여준다.

	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL/NUMERIC	BIT	CHAR	VARCHAR/
Array										
Blob										
Boolean	○	○	○	○	○	○	○	○	○	○
byte[]									○	○
char[]	○	○	○	○	○	○	○	○	○	○
Clob										
Double	○	○	○	○	○	○	○	○	○	○
Float	○	○	○	○	○	○	○	○	○	○
Integer	○	○	○	○	○	○	○	○	○	○
Java class										
BigDecimal	○	○	○	○	○	○	○	○	○	○
java.net.URL										
java.sql.Date									○	○
java.sql.Time									○	○
java.sql.Timestamp									○	○
java.util.BitSet								○		
Long	○	○	○	○	○	○	○	○	○	○
Ref										
Short		○	○	○	○	○	○	○	○	○
String	○	○	○	○	○	○	○	○	○	○
Struct										
InputStream										
Reader										

데이터베이스 데이터형을 Java 데이터형으로 변환하기

아래의 표는 데이터베이스의 각 데이터형에 대해 getXXX 메소드를 사용해서 변환이 가능한지를 보여준다.

	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL/NUMERIC	BIT	CHAR/VARCHAR
getArray									
getAsciiStream	○	○	○	○	○	○	○	○	○
getBigDecimal	○	○	○	○	○	○	○		○
getBinaryStream	○		○	○	○	○	○	○	

