JDBC Programming

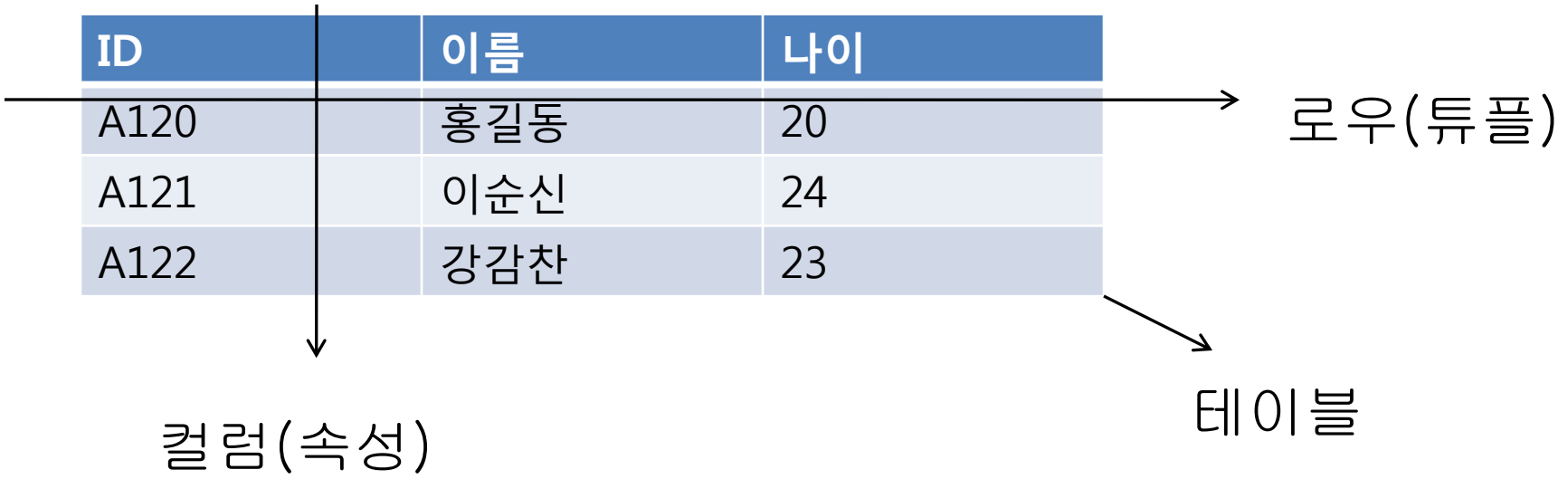
기본 SQL과 JDBC

관계형 데이터베이스

- 데이터 베이스 : 지속적으로 유지, 관리되어야 하는 데이터들의 집합
- 데이터베이스관리시스템 : Database Management System으로 데이터베이스를 관리해 주는 시스템을 말한다. – 오라클 DBMS, MS-SQL, DB2 등.
- 관계형 데이터베이스 : 데이터들을 여러항목(컬럼)의 집합으로 표현하는 테이블 형태로 데이터를 관리하는 방식

테이블의 형태

- 테이블 데이타베이스의 기본적인 데이타 저장 단위
- 테이블은 시스템내에서 독립적으로 사용되길 원하는 엔티티를 표현한다.
- 컬럼(속성)과 로우(튜플, 레코드)로 구성되어 있다.
- 테이블은 다른 테이블과의 관계를 표현할 수 있다.
  -외래키 사용



SQL

- Structured Query Language
- 데이터베이스에 데이터를 질의, 수정, 삭제 등을 요청하는 표준언어.
  - DQL – SELECT : 데이터조회
  - DML – INSERT, UPDATE, DELETE : Data 변경
  - DDL – CREATE, ALTER, DROP : 객체 생성 및 변경

테이블 생성

```
CREATE TABLE [schema.]table_name
 ( column datatype
   [, column datatype  . . .]
   [제약조건]
 )
```

- 예)CREATE TABLE DEPT(
  DEPTNO NUMBER CONSTRAINT dept_pk_deptno PRIMARY KEY,
  DNAME VARCHAR2(20),
  LOC VARCHAR2(100)
  );
- 구조만 복사하기

```
CREATE TABLE [schema.]table_name
  AS SELECT * FROM target_table name
     WHERE 1=0;
```

- 관리할 수 있는 테이블 조회(오라클)
  SELECT * FROM TAB;
- 테이블의 상세사항 보기
  DESC 테이블명

테이블 생성

- 제약조건

제약조건	설명
PRIMARY KEY(PK)	유일하게 테이블의 각행을 식별(NOT NULL과 UNIQUE조건을 만족)
FOREIGN KEY(FK)	열과 참조된 열 사이의 외래키 관계를 적용하고 설정
UNIQUE KEY(UK)	테이블의 모든 행을 유일하게 하는 값을 가진 열(NULL을 허용)
NOT NULL(NN)	열은 NULL값을 포함할 수 없다
CHECK(CK)	참이어야 하는 조건을 지정함(대부분 업무 규칙을 설정)

테이블 생성

- 데이터 타입

DATA TYPE	설 명
VARCHAR2(n)	가변 길이 문자 데이터(1~4000byte)
CHAR(n)	고정 길이 문자 데이터(1~2000byte)
NUMBER(p,s)	전체 p자리 중 소수점 이하 s자리(p:1~38, s:-84~127)
DATE	7Byte(BC 4712년 1월 1일부터 AD 9999년 12월 31일)
LONG	가변 길이 문자 데이터(1~2Gbyte)
CLOB	가변 길이 문자 데이터(1~4Gbyte)
RAW(n)	원시 이진 데이터(1~2000)
LONG RAW	가변 길이 원시 이진 데이터(1~2Gbyte)
BLOB	가변 길이 이진 데이터(1~4Gbyte)
BFILE	가변 길이 외부 파일에 저장된 이진 데이터(1~4Gbyte)

테이블 삭제 및 데이터삭제

- 테이블 삭제

```
DROP TABLE table_name
```

- 예)
  DROP TABLE MEMBER;

- 테이블의 모든 ROW 제거

```
TRUNCATE TABLE table_name
```

  - TRUNCATE 실행 후 바로 COMMIT 되어 ROLLBACK을 할 수 없다.

테이블 수정

- 컬럼 및 제약조건 추가

```
ALTER TABLE table_name
  ADD (COLUMN_NAME DATA_TYPE [제약조건])
```

```
ALTER TABLE table_name
  ADD CONSTRAINT 제약조건명 제약조건
```

  - 예) ALTER TABLE emp ADD (address VARCHAR2(100));
- 컬럼 변경

```
ALTER TABLE table_name
  MODIFY (COLUMN_NAME DATA_TYPE [제약조건])
```

  - 데이터가 존재하는 경우 변경할 수 없다. 단 VARCHAR2, CHAR의 경우 변경하려는 크기가 더 큰 경우는 가능하다.
  - 예) ALTER TABLE emp MODIFY (address VARCHAR2(200));
- 컬럼 삭제

```
ALTER TABLE table_name
DROP COLUMN   COLUMN_NAME
```

  - 예) ALTER TABLE emp DROP COLUMN address

테이블 수정

- 제약조건 삭제

```
ALTER TABLE table_name
DROP 제약조건
```

  - 예) ALTER TABLE emp DROP PRIMARY KEY;

INSERT (데이터 삽입)

- 기본구문

```
INSERT INTO table_name(column1, column2,...)
VALUES (데이터, '데이터',...) ;
```

  - 예) **INSERT INTO** DEPT (DEPTNO, DNAME, LOC) **VALUES** (100, '홍길동', '서울');
  - 문자열의 경우 삽입할 값을 ' '로 감싸준다.
  - 테이블의 모든 컬럼에 데이터를 넣을 경우 컬럼명은 생략할 수 있다.

- SELECT 문을 이용한 데이터 삽입

```
INSERT INTO table_name(column1, column2,...)
SELECT column1, column2,...
FROM table_name
WHERE 조건 ;
```

  - 예)
    INSERT INTO DEPT_EX(DEPTNO, DNAME, LOC)
    SELECT DEPTNO, DNAME, LOC FROM DEPT

UPDATE (수정), DELETE (삭제)

- 기본구문

```
UPDATE  table_name
SET column1 = 값(고칠내용), column2 = 값, ...
WHERE 조건
```

  - 예) **UPDATE** DEPT **SET** LOC = '부산' **WHERE** DEPTNO = 100 ;
    UPDATE EMP **SET** SAL = SAL * 1.1 **WHERE** EMPNO = 120 ;
    UPDATE EMP **SET** HIREDATE = SYSDATE ;

- 기본구문

```
DELETE  FROM  table_name WHERE 조건
```

  - 예) **DELETE FROM** DEPT **WHERE** DEPTNO = 100;
    DELETE FROM EMP **WHERE** SAL < (SELECT AVG(SAL) FROM EMP);
    DELETE FROM DEPT;

SELECT (조회)

- 기본구문

```
SELECT      [DISTINCT]  {*, column [alias], . . .}
FROM          table_name
[WHERE        condition]
[ORDER BY {column, expression} [ASC | DESC]];
```

- 항목
  - DISTINCT : 중복되는 행을 제거하는 옵션입니다.
  - * : 테이블의 모든 column을 출력 합니다.
  - alias : 해당 column에 대해서 다른 이름을 부여할 때 사용합니다.
  - table_name : 질의 대상 테이블명
  - WHERE : 조건을 만족하는 행들만 검색
  - condition : column, 표현식, 상수 및 비교 연산자
  - ORDER BY : 질의 결과 정렬을 위한 옵션(ASC:오름차순(Default),DESC내림차순)
- 예제

SELECT * **FROM** DEPT **WHERE** LOC = '서울' **ORDER BY** DEPTNO **DESC**

SELECT – 제약조건 연산자

연산자	설 명
BETWEEN a AND b	a와b사이의 데이터를 출력 (a, b값 포함)
IN (list)	list의 값 중 어느 하나와 일치하는 데이터를 출력
LIKE	문자 형태로 일치하는 데이터를 출력 (%, _사용)
IS NULL	NULL값을 가진 데이터를 출
NOT BETWEEN a AND b	a와b사이에 있지 않은 데이터를 출력(a, b값 포함하지 않음)
NOT IN (list)	list의 값과 일치하지 않는 데이터를 출력
NOT LIKE	문자 형태와 일치하지 않는 데이터를 출력
IS NOT NULL	NULL값을 갖지 않는 데이터를 출력

조인

- 조인이란
  - 둘이상의 테이블을 연결하여 데이터를 검색하는 방법.
  - 두 개 이상의 테이블을 SELECT문장 안에서 조인하려면 하나 이상의 컬럼이 공유 되어야한다.
- Cartesian Product(카티션 곱)
  - 검색하고자 했던 데이터뿐 아니라 조인에 사용된 테이블들의 모든 데이터가 Return되는 현상
  - 발생이유
    1. 조인 조건을 정의하지 않았을 경우
    2. 조인 조건이 잘못된 경우
    3. 첫 번째 테이블의 모든 행들이 두 번째 테이블의 모든 행과 조인이 되는 경우

    테이블의 개수가 N이라면 Cartesian Product를 피하기 위해서는 적어도 N-1 개의 조인 조건을 SELECT 문안에 포함시켜야 한다.

조인

- Equi Join
  - Equality Condition(=)에 의한 조인.
    - 대상의 되는 두 테이블에서 공통적으로 존재하는 컬럼 값이 일치되는 행을 연결하여 결과 생성
  - Equi join의 성능을 높이기 위해서는 Index 기능을 사용하는 것이 좋습니다.
  - 예

```
SELECT    e.ename, d.dname
FROM      emp e , dept d
WHERE     e.deptno = d.deptno;
```

- Non-Equi Join
  - '=' 이외의 연산을 이용한 조인
    - 조인 조건이 범위인 경우 주로 사용
  - 예

```
SELECT      e.name, e.sal, s.grade
FROM        emp e, salgrade s
WHERE       e.sal >= s.low_sal
    AND     e.sal <= s.hi_sal
```

조인

- Self Join
  - Equi Join과 같으나 하나의 테이블에서 조인이 일어나는 것이 다름니다.
  - 같은 테이블에 대해 두 개의 alias를 작성함으로 FROM절에 두 개의 테이블을 사용 하는 것과 같이 합니다.
  - 예)

```
SELECT a.ename, b.sal
 FROM  emp a, emp b
WHERE a.empno = b.empno
```

조인

- Out(외부) Join
  - Outer Join은 정상적으로 조인 조건을 만족하지 못하는 행들을 보기 위한 조인
  - Equijoin 제약점 : 조인을 생성하려 하는 두 개의 테이블의 두 개 컬럼에서 공통된 값이 없는 경우 데이터를 하나도 Return하지 않는다.
  - 구문 : 연산자 "(+)"를 이용
  - 조인시킬 값이 없는 조인측에 "**(+)**"를 위치시킨다. – 조건을 만족하지 못하는 행을 가지지 않은 쪽에 (+)을 붙인다.
  - Outer join 연산자는 표현식의 한 편에만 올 수 있다.
  - Outer join은 IN 연산자를 사용할 수 없고 OR 연산자에 의해 다른 하나의 조건에 연결될 수 없습니다

예제1) **일반 조인의 경우**

```
SQL> SELECT DISTINCT(a.deptno), b.deptno
     FROM emp a, dept b
     WHERE  a.deptno = b.deptno

DEPTNO     DEPTNO
---------- ----------
        10         10
        20         20
        30         30
```

예제2) **out join을 했을 경우**

```
SQL>  SELECT DISTINCT(a.deptno), b.deptno
      FROM emp a, dept b
      WHERE  a.deptno(+) = b.deptno

 DEPTNO     DEPTNO
-------     ----------
     10         10
     20         20
     30         30
                40
```

서브 쿼리

- 개요 : 하나의 테이블에서 검색한 결과를 다른 테이블에 전달하여 검색하도록 하는 기능
  - FROM절에서 사용 또는 WHERE 절에서 사용
  - 메인 쿼리 : 실제 조회하고자 하는 쿼리
  - 서브 쿼리 : 메인 쿼리에서 사용할 데이터를 조회하기 위한 쿼리
  - 종류 : 단일 행 서브쿼리, 다중 행 서브 쿼리, 인라인뷰
- 단일 행 서브쿼리
  - 서브 쿼리의 조회결과가 0개 또는 1개인 쿼리
- 예)

```
SELECT      dname
FROM        dept
WHERE       dpetno = (SELECT deptno
                      FROM    emp
                      WHERE  ename="홍길동")
```

서브쿼리

- 다중 행 서브쿼리
  - 서브 쿼리의 조회결과 행이 0개 이상의 경우의 쿼리
  - IN, ALL, ANY 연산자등과 사용된다.

예)

```
SELECT      e.ename
 FROM       emp e
WHERE       e.salary IN (
                        SELECE salary
                        FROM   emp
                        WHERE deptno=30)
```

서브쿼리

- 인라인 뷰(Inline view)
  - FROM절 상에 오는 서브쿼리
  - 예

```
SELECT e.ename, d.dname
 FROM  emp e,
       (SELECT deptno, dname
        FROM   dept
        WHERE deptno=10
        ) d
WHERE       e.deptno = d.deptno;
```

JDBC란?

- **Java Database Connectivity**
- 자바 데이터베이스 프로그래밍 API
  - 자바 프로그램이 데이터베이스와 연결되어 데이터를 주고 받을 수 있게 해 주는 프로그래밍 인터페이스이다.

JDBC API

세가지 입장에서의 JDBC API

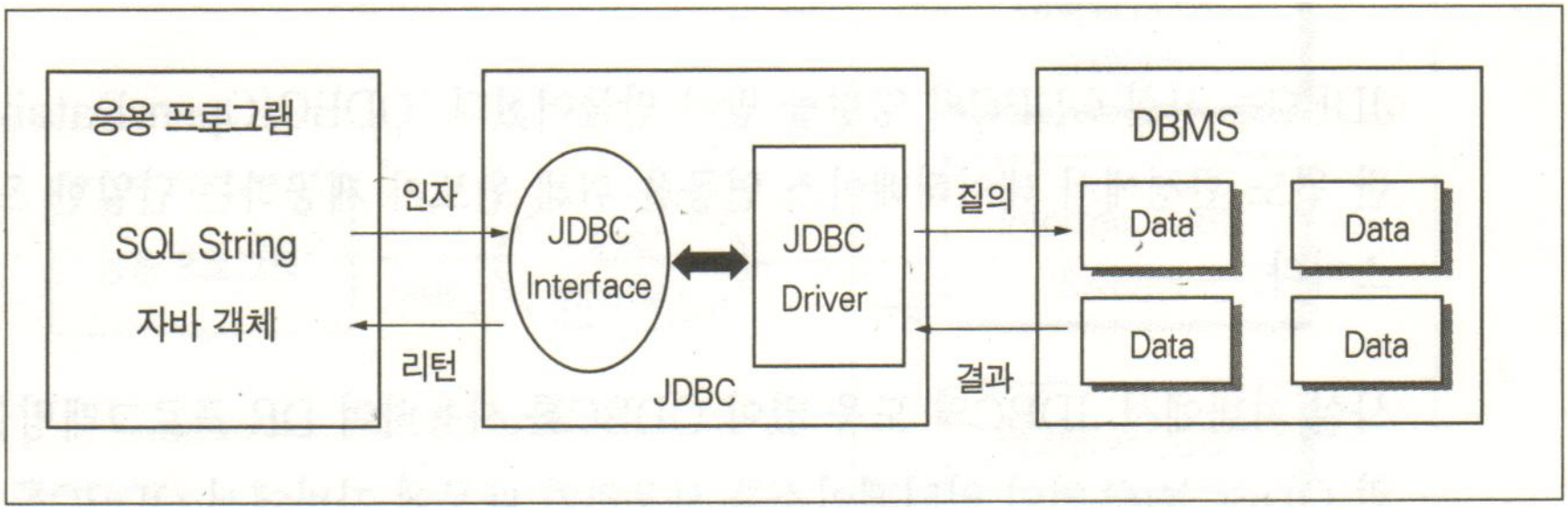
1. Java 표준 제안자
2. Java 응용 프로그램 개발자(일반 Programmer)
3. DBMS Vendor
   (Oracle, MS SQL, DB2..)

JDBC API

1. Java 표준 제안자(Sun MicroSystems)
   - JDBC 명세(기준)를 제공.
2. DB Vendor
   - JDBC 명세를 기준으로 JDBC API를 개발하여 배포
3. Java 응용 프로그램 개발자
   - DB Vendor에서 배포한 JDBC API를 받아 JDBC 명세를 보고 프로그램을 작성한다.

JDBC의 역할

통역자의 역할 – 응용프로그램과 DBMS간의 통신을 중간에서 번역해주는 역할을 한다.



JDBC Driver Class

Driver : DB와의 연결을 관리하는 클래스
URL : DBMS에 연결을 위한 URL – DBMS마다 다르다.

JDBC driver	Driver Class	**URL**
JDK 기본 driver	sun.jdbc.odbc.JdbcOdbcDriver	"jdbc:odbc:<dbname>"
오라클 **8i thin** driver	oracle.jdbc.driver.OracleDriver	"jdbc:oracle:thin:@<server-ip>:1521:SID"
오라클 **OCI** driver	oracle.jdbc.driver.OracleDriver	"jdbc:oracle:oci8:@<dbname>"
IBM DB2	**com.ibm.db2.jcc.DB2Driver**	"jdbc:db2://<server-ip>:<port-no>/<dbname>"
인포믹스 driver **2.0 JC1**	om.informix.jdbc.IfxDriver	"jdbc:informix-sql://<ip>:<port>/<dbname>:INFORMIXSERVER"
mSQL Imaginary JDBC driver	com.imaginary.sql.msql.MsqlDriver	"jdbc:msql://<ip>:<port>/<dbname>"
MySQL driver	com.mysql.jdbc.Driver	"jdbc:mysql://<ip>:3306/<dbname>"

java.sql Package

JDBC관련 중요 interface들

- java.sql.Driver
  - 모든 DBMS를 만드는 Vender측에서 DB와 연결하는 Driver class를 만들 때 반드시 implements 해야 하는 interface로 JDBC 드라이버의 중심이 되는 Interface.
- java.sql.Connection
  - 특정 데이타베이스와 연결정보를 가지는 interface. DriverManager로 부터 Connection객체를 가져 온다.
- java.sql.Statement
  - SQL query문을 DB에 전송하는 방법을 정의한 Interface. Connection을 통해 가져 온다.
- java.sql.ResultSet
  - SELECT 구문 실행 결과를 조회할 수 있는 방법을 정의한 Interface.

java.sql Package

- java.sql.PreparedStatement
  - Statement 의 하위 Interface. SQL문을 미리 컴파일 하여 실행 속도를 높임.
- java.sql.CallableStatement
  - PreparedStatement의 하위 Interface. DBMS의 Stored procedure를 호출.

JDBC Programming Pattern

1. Driver loading
2. Connection(연결)
3. Statement/PreparedStatement
4. ResultSet(Select의 경우)
5. close(Connection, Statement, ResultSet)

JDBC Programming Pattern

1.Driver Loading

Driver : DB와 P/G의 연결을 관리

```
String driver = "oracle.jdbc.driver.OracleDriver"
    Class.forName(driver);
```

```
new oracle.jdbc.driver.OracleDriver();
```

```
DriverManager.resisterDriver
(oracle.jdbc.driver.OracleDriver());
```

```
Prompt> java -D jdbc.drivers=
oracle.jdbc.driver.OracleDriver SimpleJDBC
```

JDBC Programming Pattern

2. DB와 연결

DB와 연결을 위해 URL과 계정정보 필요

DB url : jdbc:subprotocol:subname

URL은 DBMS vender마다 다름.

- Oracle url - "jdbc:oracle:thin:@server-ip:1521:SID“

연결메소드 : DriverManager.getConnection(url, id, pwd) : Connection

```
String url = " jdbc:oracle:thin:@ 127.0.0.1:1521:ORLC”

Connection con =
        DriverManager.getConnection(url, “user”, “pass”);
```

JDBC Programming Pattern

3. Create a Statement

```
Statement stmt = con.createStatement();
```

Statement : query의 내용이 run-time에 결정되어진다.

(dynamic, 실행은 늦지만 융통성이 좋다.)

참고:PreparedStatement ; query의 내용이 compile-time에 결정되어진다. (static, 융통성은 떨어지지만 실행은 빠르다)

JDBC Programming Pattern

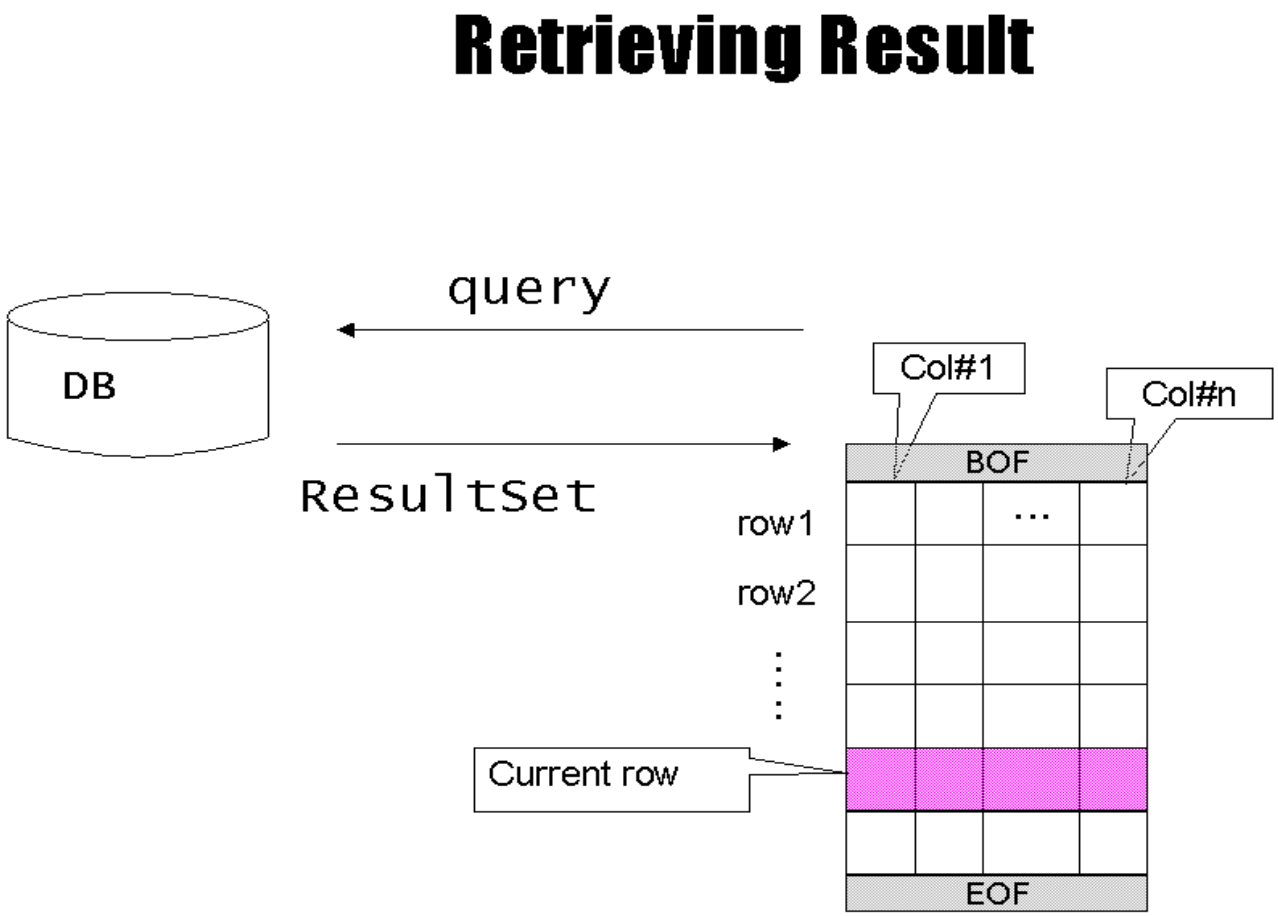
4. 쿼리 전송

```
String query = “sql문”;
int count = stmt.executeUpdate(query); //DML
ResultSet rset = stmt.executeQuery(query); //DQL
```

**** SQL Query가 INSERT, DELETE, UPDATE인 경우에는 executeUpdate() 를 사용한다. Query 실행 후 반영된 결과 record의 개수가 반환된다.***

**** SQL Query가 SELECT인 경우에는 executeQuery() 를 사용한다. Query 실행후 SELECT한 결과가 ResultSet type으로 반환된다.***

- **ResultSet의 형태**



JDBC Programming Pattern

6. Retrieving Result (throws SQLException)

```
rset : ResultSet
1.  while(rset.next()) {                    //Cursor를 한행씩 내린다.
      String str = rset.getString(1);     //그 행의 특정 Column의 값을 가져온다.
      int i = rset.getInt("field_name"); //Column의 값을 가져온다.
    }
2.  if(rset.next()) {

     String str = rs.getString(1);

    }
```

- ***1) SELECT 결과가 여러 개일 경우***
- ***2) SELECT 결과가 하나일 경우***

JDBC Programming Pattern

7. Close Resource (throws SQLException)

```
rset.close();
stmt.close();
con.close();
```

PreparedStatement

- **생성 –** SQL 구문을 정의하고 변경 될 값은 치환문자(?)를 이용해 쿼리 전송 전에 값을 setting 한다.

  String sql = "INSERT INTO MEMBER (ID, NAME, ADDRESS)
  VALUES(?,?,?)";

  PreparedStatement pstmt = conn.prepareStatement(sql);

- **값 세팅 –** 쿼리 전송전에 Column의 데이터 타입에 맞는 set 메소드를 호출하여 ? 에 값을 setting한다.

  setXXX(int ?index, value);

- **쿼리 전송**

  executeQuery() : ResultSet
  executeUpdate() : int

Transaction 처리

- Transaction : Database update작업 중 도중 실패했을 경우 작업이전 상태로 되돌리는 것.
  - Begin : 작업의 시작
  - Commit : 정상 종료
  - rollback : 비 정상 종료
- JDBC Connection은 기본적으로 쿼리 전송시 자동 커밋이 일어남
- Transaction 처리는 Connection 객체를 통해 처리
  - begin : conn.setAutoCommit(false);
  - commit : conn.commit();
  - rollback : conn.rollback();

Connection Pool과 DataSource

- DBMS와 연결 : 속도가 느리다.
- 해결책 : Connection Pool
- 내용
  - Connection Pool이란 Connection을 관리하는 객체 Pool
  - Connection들을 미리 생성 하여 Pool에 저장한 뒤 필요 시마다 빌려 쓰는 개념
  - 사용 후에는 다시 Connection Pool로 반납한다.
- Connection Pool은 작성하거나 API로 제공 되는 것을 사용한다.
  - Apache의 DBCP api

- DataSource : DriverManager의 upgrade version으로 Connection Factory 이다.
  - JDBC 2.0에서 지원
  - DataSource 객체는 연결할 DB에 대한 정보(driver, url, 계정) 를 가지고 있다.
  - 내부적으로 Connection Pool을 지원할 수 있다.
  - 구현 방식 : Vendor 마다 다름.
    - 요청 시 Connection을 생성하여 제공
    - Connection Pool을 이용해 Connection을 미리 생성후 요청 시 제공