Accesso alle Basi di Dati

- I Sistemi Informativi hanno il compito di gestire in modo efficiente ed affidabile i dati, mantenuti in una forma strutturata e possibilmente normale.

- La struttura dei dati viene definita attraverso il **modello Entity/Relationship**

- L'accesso e la manipolazione dei dati è standardizzata a livello di linguaggio attraverso lo standard di fatto che è **SQL (Simple Query Language)**

- Ogni base di dati può essere gestita quindi in modo standard, senza conoscere quali sono i dettagli del software e dell'hardware che si occupa della gestione del dato fisico

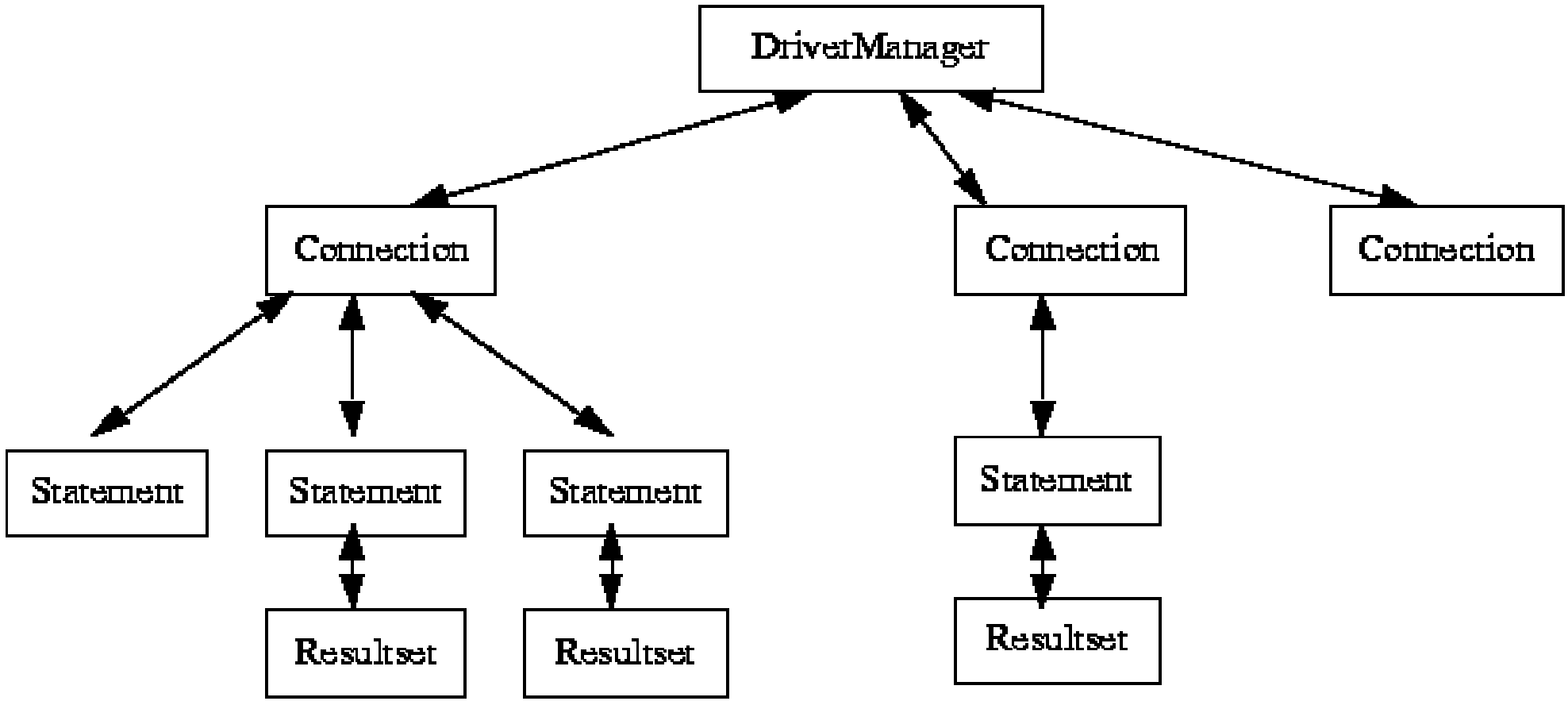
JDBC
Java DataBase Connection

- Nasce come strumento standard per gestire le connessioni alle basi di dati DBMS

- È costituito da un insieme di package standard:
  1. java.sql
  2. javax.sql

- I package standard definiscono una interfaccia al quale deve sottostare il package DataBase dependent detto "*driver*"

JDBC
Struttura delle API



JDBC
Struttura delle API

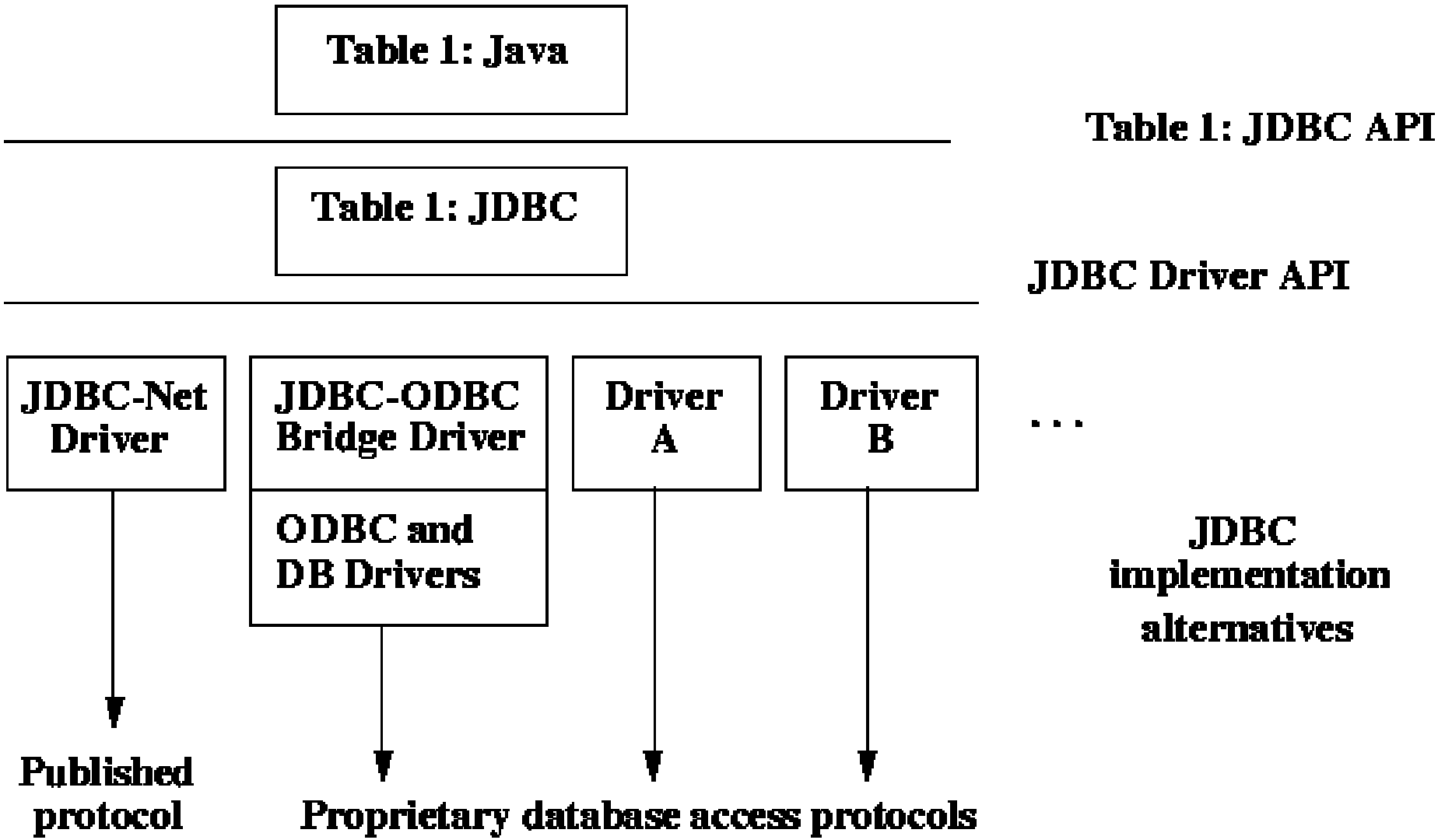
- **Driver Manager:** (java.sql.DriverManager) si occupa della gestione dei Driver per accedere ai diversi sistemi informativi, ha il compito di gestire l'apertura e la chiusura delle connessioni verso questi sistemi

- **Connection:** (java.sql.Connection) rappresenta una connessione diretta alla base di dati. Mantiene lo stato della interazione tra applicazione e databse.

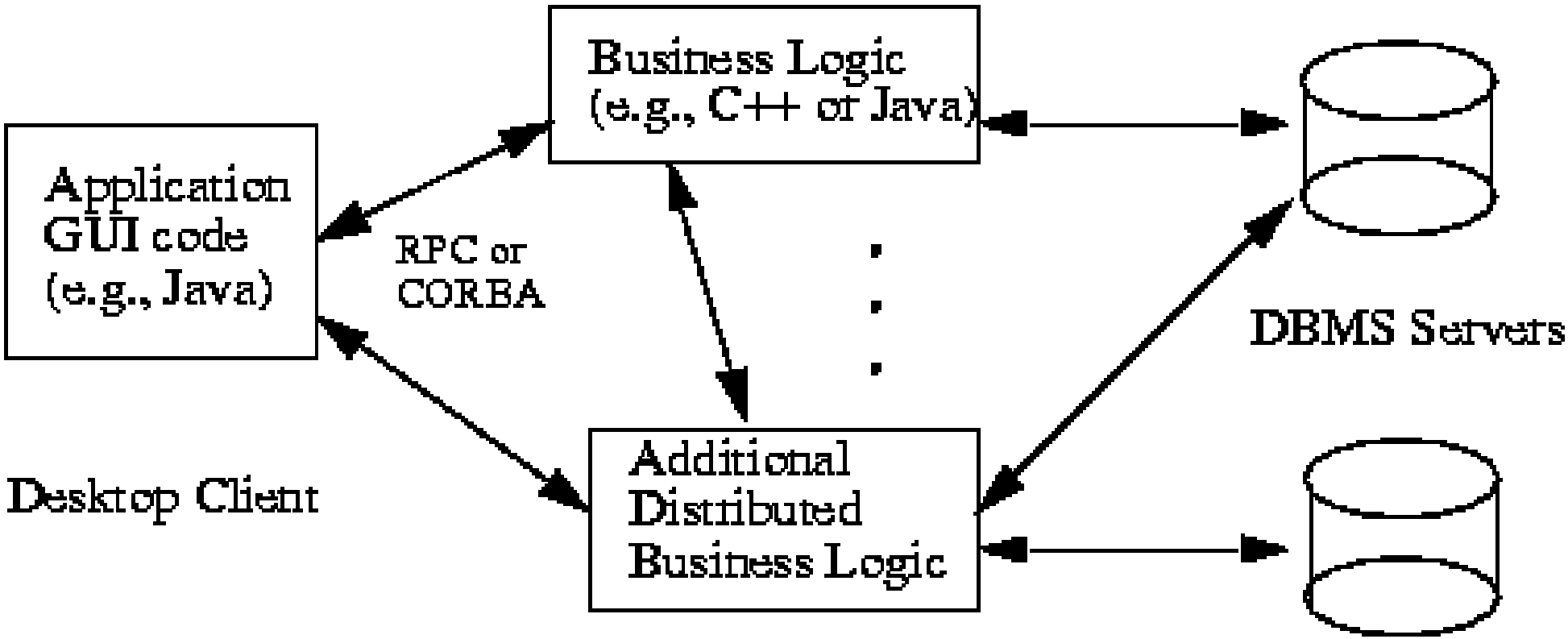
- **Statement:** (java.sql.Statement) rappresenta l'accesso (via SQL) ad un insieme di dati attraverso una specifica connessione. Lo statement rappresenta quindi una richiesta SQL

- **ResultSet:** (java.sql.ResultSet) contiene il risultato di una specifica richiesta SQL; contiene in generale l'insieme delle righe (record) ottenuti dalla richiesta.

JDBC Driver Interface



JDBC
Modello di esecuzione



JDBC
Apertura di una Connessione

```
try {

    // Create a URL specifying an ODBC data source name.
    String url = "jdbc:odbc:NomeDB";
    String user=“scott”;
    String pwd=“tiger”

    // Connect to the database at that URL.
   Connection con = DriverManager.getConnection(url, user,pwd);

} catch (java.lang.SQLException ex) {
… … … …
}
```

JDBC
Esecuzione di una Query SQL

```
try {

    String sSQL="SELECT a, b, c, d, key FROM Table1“;

    // Create a Statement
    Statement stmt = con.createStatement();

    // Execute a SELECT statement
    ResultSet rs = stmt.executeQuery(sSQL);

} catch (java.lang.SQLException ex) {
… … … …
}
```

JDBC
Lettura di un ResultSet

```
try {

    while (rs.next()) {
        // get the values from the current row:
        int a = rs.getInt("a");
        BigDecimal b = rs.getBigDecimal("b");
        char c[] = rs.getString("c").tocharArray();
        boolean d = rs.getBoolean("d");
        String key = rs.getString("key");
        … … … … …
    }
    stmt.close();
    con.close();
} catch (java.lang.SQLException ex) {
… … … …
}
```

JDBC
Compatibilità dei dati Java-SQL

SQL → Java

SQL type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Java → SQL

Java Type	SQL type
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

		SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	**X**	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	**X**	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	**X**	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	**X**	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	**X**	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	**X**	**X**	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	x	**X**	**X**	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	x	x	**X**	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	**X**	**X**	x	x	x	x	x	x	x
getBytes														**X**	**X**	x			
getDate											x	x	x				**X**		x
getTime											x	x	x					**X**	x
getTimestamp											x	x	x				x		**X**
getAsciiStream											x	x	**X**	x	x	x			
getUnicodeStream											x	x	**X**	x	x	x			
getBinaryStream														x	x	**X**			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 1: Use of ResultSet.getXXX methods to retrieve common SQL data types
An "x" means that the given getXXX method can be used to retrieve the given SQL type.
An "**X**" means that the given getXXX method is recommended for retrieving the given SQL type

JDBC
Transazioni

- Associata ad ogni connessione esiste sempre una transazione
- Di default, ogni connessione è configurata in "*autocommit*" , automaticamente, al completamento di ogni query la tranzazione viene "*chiusa con successo*" (*commit*) e ne viene riaperta una nuova
- È possibile configurare la connessione in *manual commit:*
  - **Connection.setAutoCoomit(false);**
- Quando il commit è manuale, per chiudere le transazioni è necessario specificare se la chiusura è con successo o con rollback:
  - **Connection.commit();**
  - **Connection.roolback();**