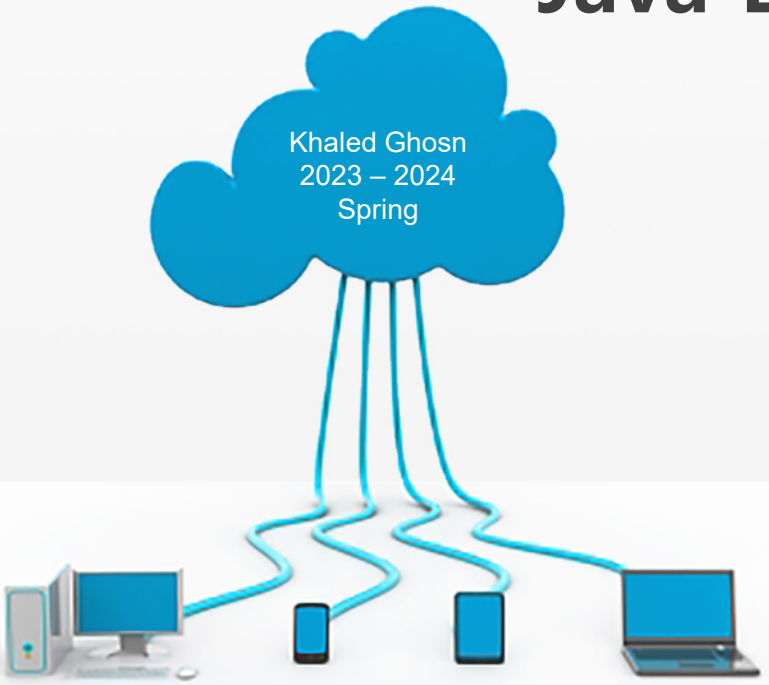
ARTS, SCIENCES & TECHNOLOGY
UNIVERSITY IN LEBANON
AUL

Java Database Programming



Khaled Ghosn
2023 – 2024
Spring

Java Database Connectivity (JDBC)

What is a Database System?

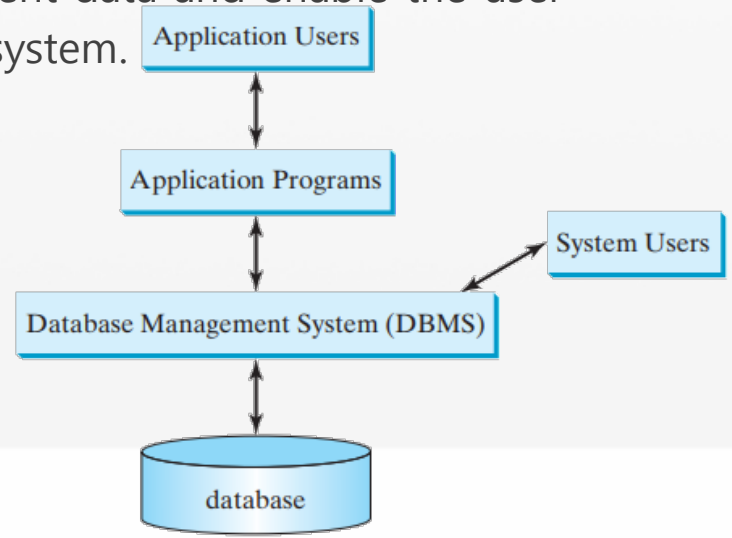
A **database** is a repository of data that form information.

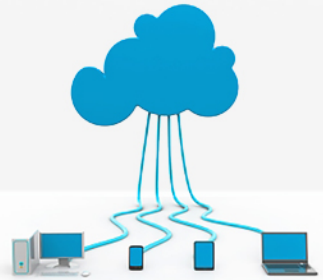
A **database system** consists of a:

a) *Database*

b) *Software* that stores and manages data in the database

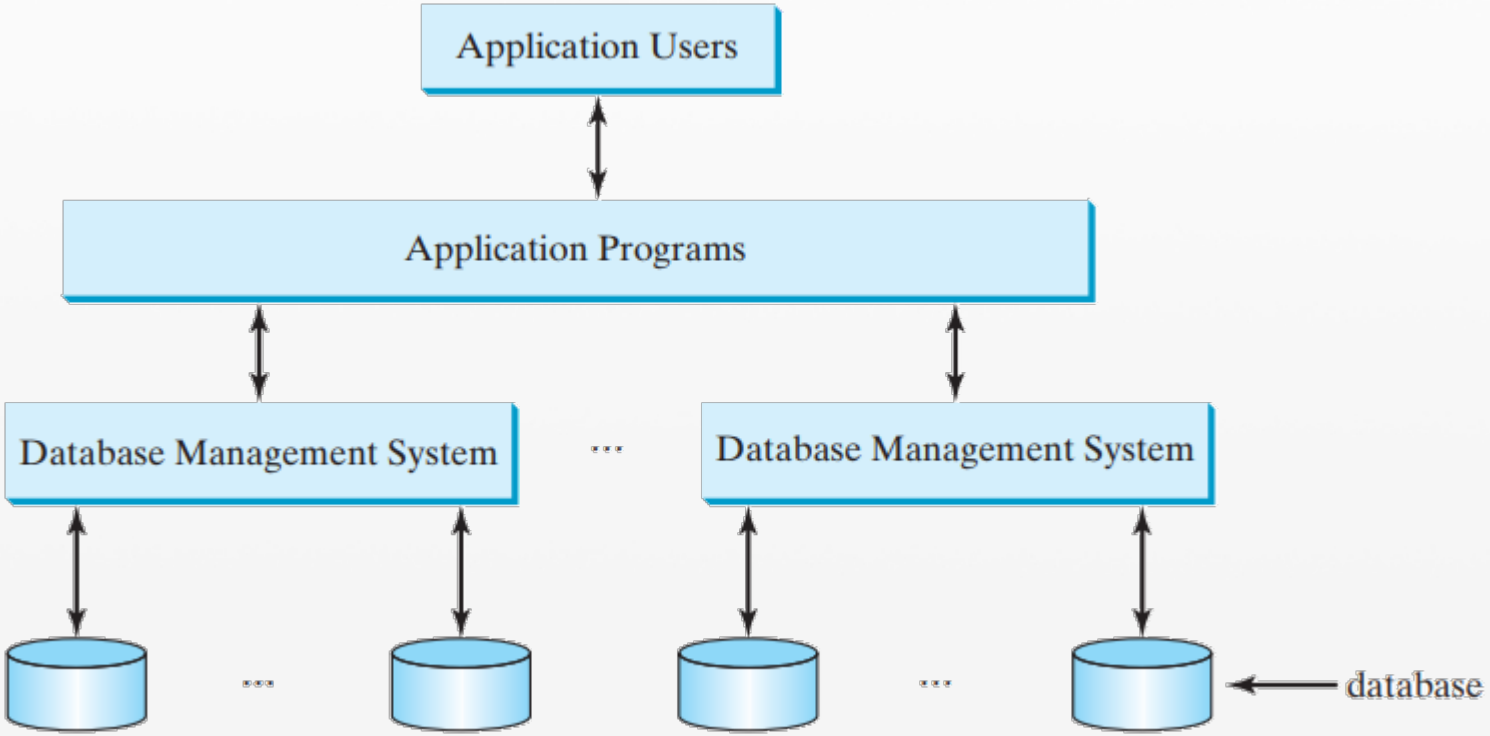
c) *Application* programs that present data and enable the user to interact with the database system.

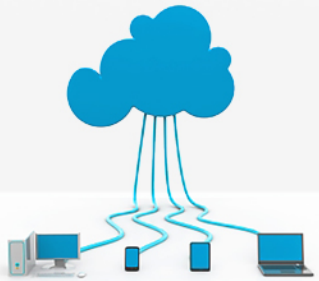




Database Application Systems

An application program can access multiple database systems.





Relational Database

Most of today's database systems are **relational database systems**.

They are based on the relational data model, which has three key components:

i. **Structure**: defines the representation of the data.
ii. **Integrity**: imposes constraints on the data.
iii. **Language**: provides the means for accessing and manipulating data.

Relational Structures

A relational database consists of a set of relations.

A relation is actually a **table** that consists of non-duplicate rows.

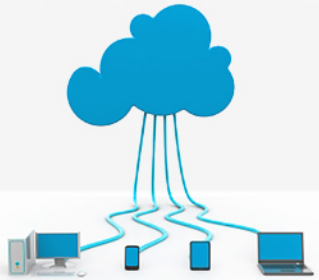
Tables describe the relationship among data.

A table has a table name, column names, and rows.

A **row** (*tuple*) of a table represents a record

- each row in a table represents a record of related data.

A **column** (*attribute*) of a table represents the value of a single attribute of the record.



Relational Structures

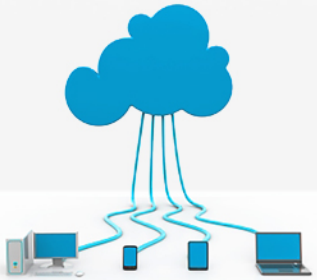
Relation/Table Name

Columns/Attributes

Course Table

Tuples/
Rows

courseId	subjectId	courseNumber	title	numOfCredits
11111	CSCI	1301	Introduction to Java I	4
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
11114	CSCI	4750	Rapid Java Application	3
11115	MATH	2750	Calculus I	5
11116	MATH	3750	Calculus II	5
11117	EDUC	1111	Reading	3
11118	ITEC	1344	Database Administration	3

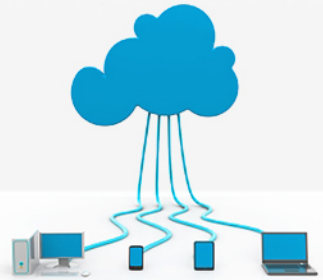


Integrity Constraints

An *integrity constraint* imposes a condition that all the legal values in a table must satisfy.

In general, there are three types of constraints:

1. **Domain** constraints:
2. **Primary key** constraints
3. **Foreign key** constraints.



Domain Constraints

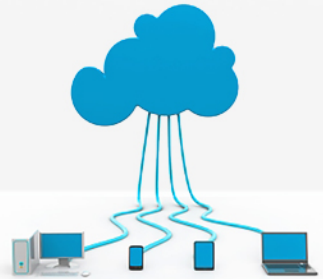
Domain Constraints specify the permissible values for an attribute

Domains can be specified using standard data types

(integers, floating-point numbers, fixed-length strings, and variant-length strings …)

Additional constraints can be specified to narrow the ranges

such as: Not Null, Check, Unique, Default



Domain Constraints

Enrollment Table

ssn	courseId	dateRegistered	grade
444111110	11111	2004-03-19	A
444111110	11112	2004-03-19	B
444111110	11113	2004-03-19	C
...			

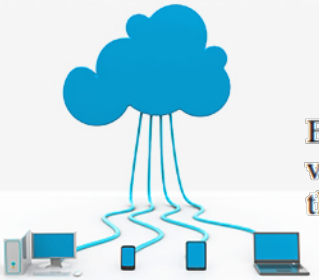
Each value in `courseId` in the Enrollment table must match a value in `courseId` in the Course table

Course Table

courseId	subjectId	courseNumber	title	numOfCredits
11111	CSCI	1301	Introduction to Java I	4
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
...				

Each row must have a value for `courseId`, and the value must be unique

Each value in the `numOfCredits` column must be greater than 0 and less than 5

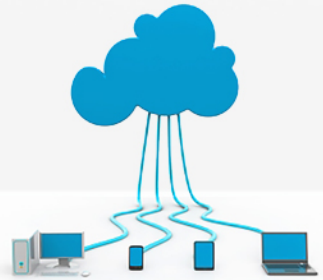


Primary Key Constraints

The primary key is often used to identify tuples in a relation.

The primary key constraint specifies that:

- the primary key value of a tuple cannot be null (Not Null)
- and no two tuples in the relation can have the same value (Unique) on the primary key.



Primary Key Constraints

Enrollment Table

ssn	courseId	dateRegistered	grade
444111110	11111	2004-03-19	A
444111110	11112	2004-03-19	B
444111110	11113	2004-03-19	C
...			

Each value in `courseId` in the Enrollment table must match a value in `courseId` in the Course table

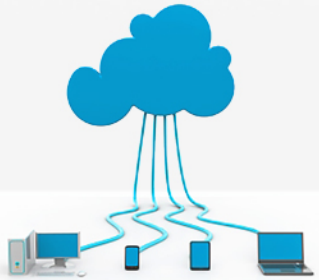
Course Table

courseId	subjectId	courseNumber	title	numOfCredits
11111	CSCI	1301	Introduction to Java I	4
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
...				

Primary key constraint

Each row must have a value for `courseId`, and the value must be unique

Each value in the `numOfCredits` column must be greater than 0 and less than 5



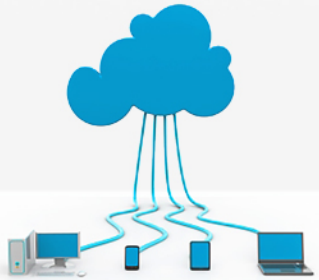
Foreign Key Constraints

In a relational database, data are related.

Tuples in a relation are related and tuples in different relations are related through their common attributes.

The common attributes are foreign keys.

The foreign key constraints define the relationships among relations.



Foreign Key Constraints

Enrollment Table

ssn	courseId	dateRegistered	grade
444111110	11111	2004-03-19	A
444111110	11112	2004-03-19	B
444111110	11113	2004-03-19	C
...			

Foreign key constraint

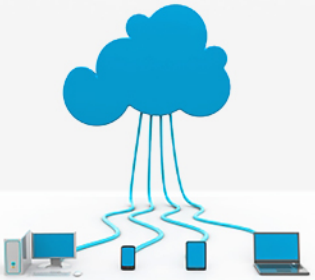
Each value in `courseId` in the Enrollment table must match a value in `courseId` in the Course table

Course Table

courseId	subjectId	courseNumber	title	numOfCredits
11111	CSCI	1301	Introduction to Java I	4
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
...				

Each row must have a value for `courseId`, and the value must be unique

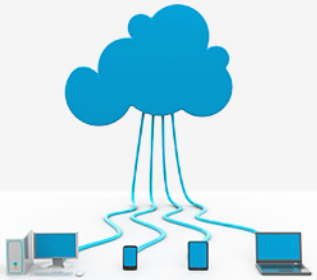
Each value in the `numOfCredits` column must be greater than 0 and less than 5



Structured Query Language (SQL)

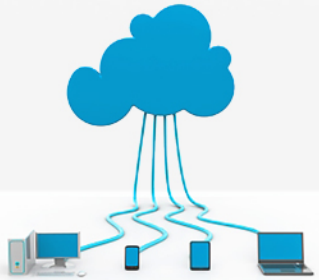
is the language for defining tables and integrity constraints, and for accessing and manipulating data.

- DCL (Data Control Language)
  - Creating a User Account
  - …
- DDL (Data Definition Language)
  - Creating a Database
  - Creating and Dropping Tables
  - …
- DML (Data Manipulation Language)
  - Insert data
  - Update data
  - Delete data
  - Retrieve Data …



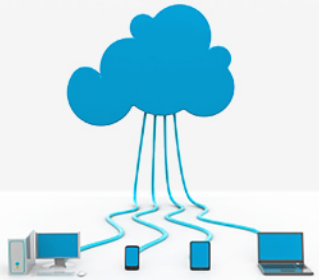
JDBC

- Most organizations have the bulk of their data structured into **relational databases** , which often need to be accessed from more than one site
- **JDBC** is the Java *API* for accessing relational database
- How to cope with the variation in internal format of databases (Oracle, MySQL, Access …)?
- A **driver** is a mediating software that will allow JDBC to communicate with the vendor specific API (of a database)

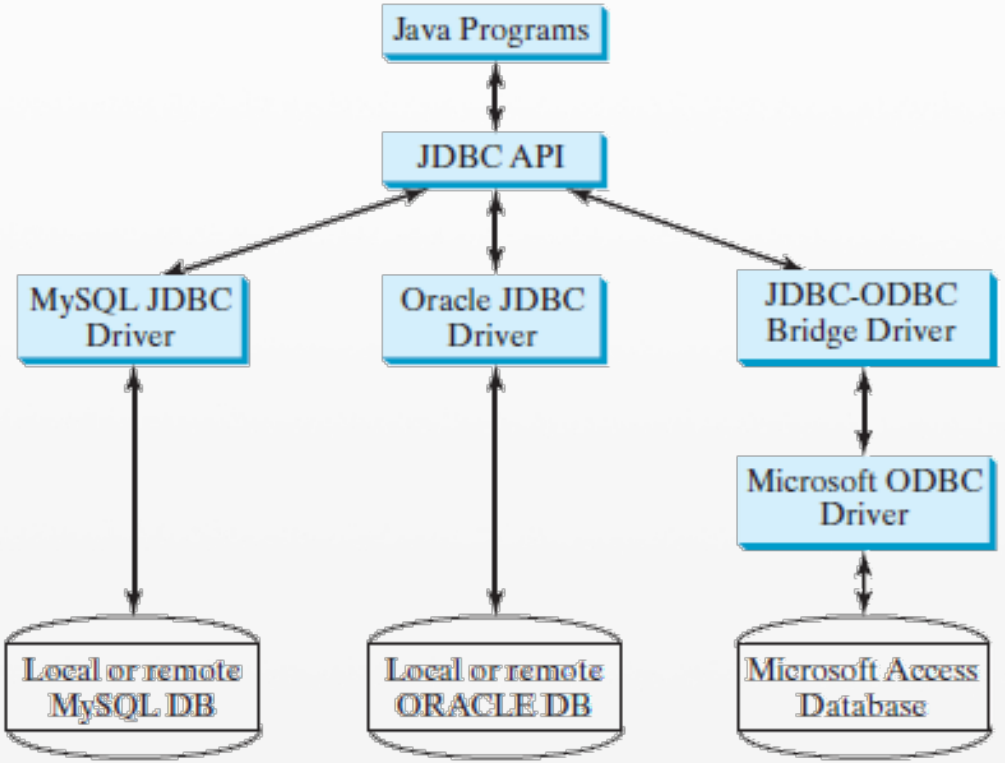


JDBC

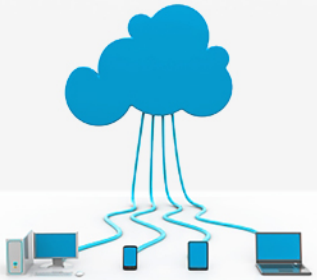
- Drivers are usually supplied either by the database vendors or by third parties
- Open Database Connectivity (**ODBC**), by Microsoft, used to access databases with different internal formats
- **JDBC-ODBC bridge driver** converts the JDBC protocol into the corresponding ODBC one and allows Java programmers to access databases for which there are ODBC drivers



JDBC



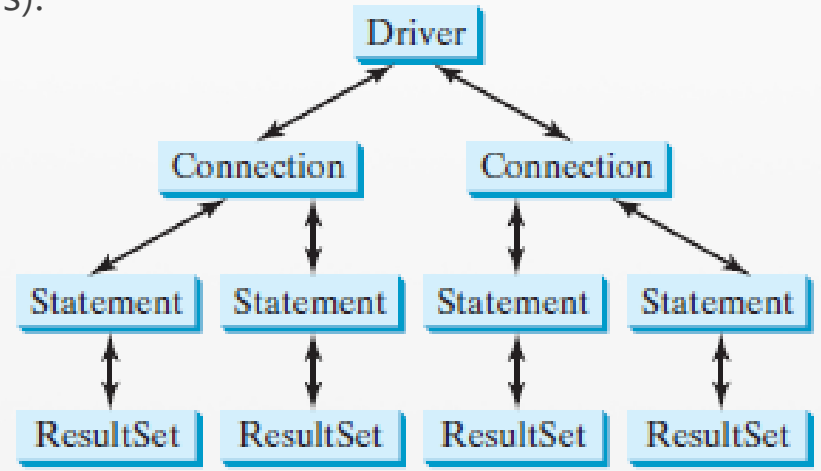
Java programs access and manipulate databases through JDBC drivers



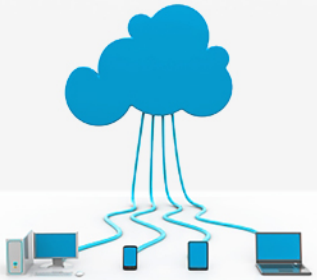
JDBC

Steps to access a database

- Each JDBC driver must implement these three interfaces (and the implementation classes):
  - *Connection*
  - *Statement*
  - *ResultSet*



JDBC classes enable Java programs to connect to the database, send SQL statements, and process results

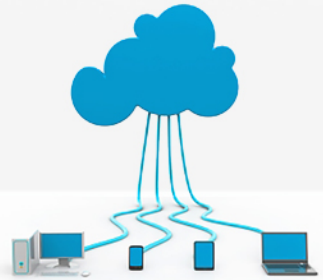


JDBC

Steps to access a database

To access a database, several steps are required:

1. Load the database driver
2. Establish a *connection* to the database
3. Create a *Statement* object (using the *connection*) and store a reference to it
4. Run a specific query or update statement (using statement) and accept the result(s)
5. Process result-set(s): For a query, manipulate and display the results. For an update, check/show number of database rows affected
6. Repeat steps 4 and 5 as many times as required for further queries / updates
7. Close the connection

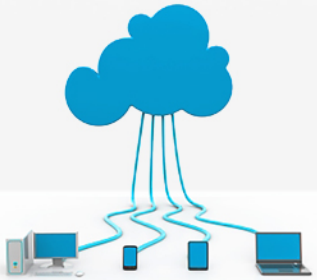


JDBC

1. Load the database driver

- An appropriate driver must be loaded before connecting to a database
- Class.forName (**"JDBCDriverClass"**);
- A driver is a concrete class that implements the **java.sql.Driver** interface
- JDBC Drivers:

Database	*Driver Class*	*Source*
Access	sun.jdbc.odbc.JdbcOdbcDriver	Already in JDK
MySQL	com.mysql.jdbc.Driver	mysql-connector-java-5.1.26.jar
Oracle	oracle.jdbc.driver.OracleDriver	ojdbc6.jar



JDBC

2. Establish a *connection* to the database

use the static method **getConnection (databaseURL)** in the **Driver Manager** class, as follows:

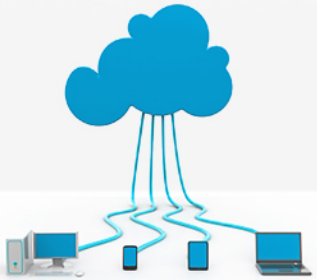
```
Connection connection = DriverManager.getConnection ( databaseURL );
```

Method *getConnection* takes three *String* arguments:

- a URL-style address for the database;
- a user name;
- a password

Database	*URL Pattern*
Access	jdbc:odbc:dataSource
MySQL	jdbc:mysql://hostname/dbname
Oracle	jdbc:oracle:thin:@hostname:port#:oracleDBSID

JDBC URLs



JDBC

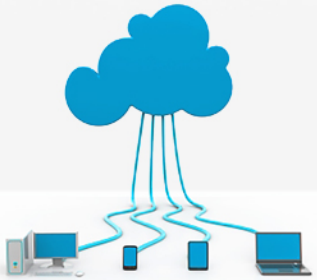
2. Establish a *connection* to the database

Connecting local DB:

```
Connection connection =
    DriverManager.getConnection ( "jdbc:odbc:DBNAME", "", "" );
```

Connecting remote DB:

```
Connection connection =
    DriverManager.getConnection
        ("jdbc:odbc://SERVER.SomethingElse.com/DBNAME", "", "" );
```



JDBC

3. Create a *Statement* object and store its reference

A *Statement* object is created by calling the ***createStatement*** method of our *Connection* object

```
Statement statement = connection.createStatement ( );
```

JDBC

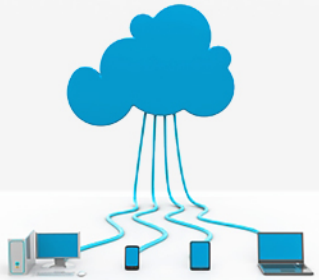
4. Run a Query or Update and Accept the Result(s)

Data Manipulation Language (DML) statements:

- Retrieve data (**SELECT** statements)
  - ➔ *executeQuery* : returns a *ResultSet* object

- Change contents in DB (**INSERT**, **UPDATE**, **DELETE** statements)
  - ➔ *executeUpdate* : returns the number of rows that have been affected by the updating operation

- It is common practice to store the SQL query in a *String* variable and then invoke *executeQuery*



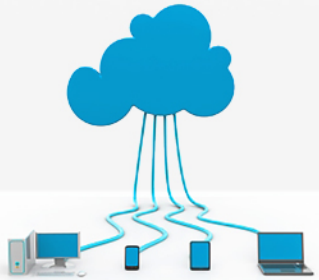
JDBC

4. Run a Query or Update and Accept the Result(s)

i.
```
String selectAll = "SELECT * FROM Accounts";
ResultSet results = statement.executeQuery (selectAll);
```

ii.
```
String selectFields = "SELECT acctNum, balance FROM Accounts";
ResultSet results = statement.executeQuery (selectFields);
```

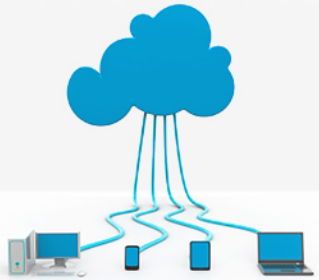
iii.
```
String selectRange = "SELECT * FROM Accounts WHERE
    balance >= 0 AND balance <= 1000 ORDER BY balance DESC";
ResultSet results = statement.executeQuery (selectRange);
```



JDBC

5. Processing Resultset

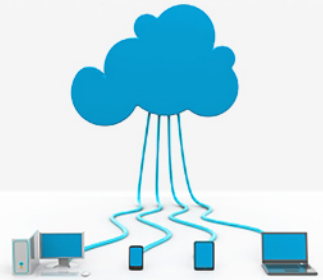
- Manipulate / Display / Check Result(s)
- The *ResultSet* object contains the database rows that satisfy the query
- ResultSet contains a very large number of methods for manipulating these rows
- A ***next*** method moves the *ResultSet* cursor / pointer to the next row
- A **get**XXX method can have a columnIndex or columnName as an argument



JDBC

5. Processing Resultset

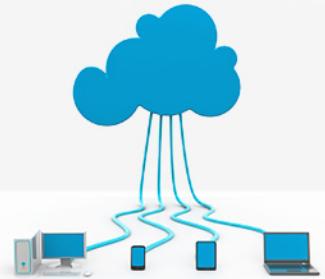
```
String select = "SELECT * FROM Accounts";
ResultSet results = statement.executeQuery(select);
while ( results.next ( ) ) {
      System.out.println ("Account no.“ + results.getInt (1) );      // or
  // System.out.println("Account no.“ + results.getInt ( "acctNum“ );

      System.out.println("Account holder: “ + results.getString (3) + " "
                                                + results.getString (2) );
      System.out.println ("Balance: “ + results.getFloat (4) );
      System.out.println ( );
}
```



JDBC

6. Repeat 4, 5 as required

- A *Statement* reference can be reused to execute other queries or updates
- For each operation, simply repeat step 4 and 5
- Repeat as many times as required by the applications



JDBC

7. Close the connection

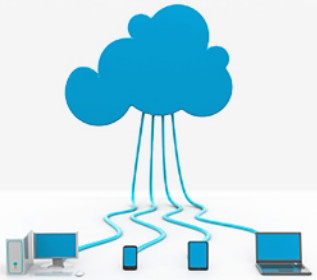
- Close the Connection object

  connection.close ( );

- Close the Statement object

  statement.close ( );

- *SQLException* must be caught for any SQL statement
- *ClassNotFoundException* is generated when the database driver couldn't be found



JDBC

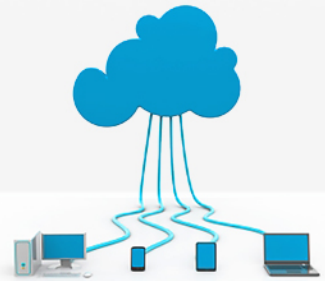
Prepared Statement

- String *queryString* = "select * from Products where
  Color = ? and Price = ?";

- **PreparedStatement** ps =
  connection.prepareStatement (*queryString*);

- ps.**setString** (1, “Red”);
- ps.**setInt** (2, 100);

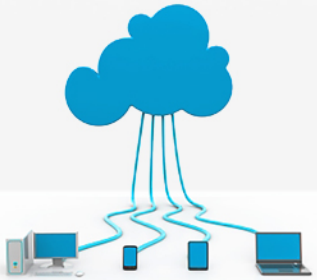
- ResultSet results = ps.**executeQuery** ( );



JDBC

Transactions

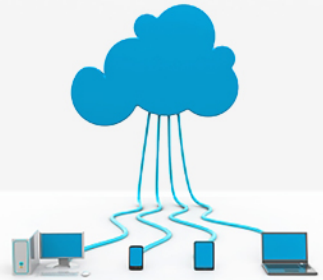
- A transaction is one or more SQL statements that may be grouped together as a single processing entity
- Transaction execution must satisfy the ALL-or-NONE property
- **COMMIT** is used at the end of a transaction to commit / finalize the database changes
- **ROLLBACK** is used (in an error situation) to restore the database to the state it was in prior to the current transaction
- By default, JDBC automatically commits individual SQL statements applied to a database
- To switch off auto-commit: *setAutoCommit* (*false*)



JDBC

Transactions

```
// Assumes existence of 3 SQL update strings called
// update1 , update2 and update3
connection.setAutoCommit (false);
Try {
        statement.executeUpdate (update1);
        statement.executeUpdate (update2);
        statement.executeUpdate (update3);
        connection.commit ( );
}
Catch (SQLException sqlEx) {
        connection.rollback ( );
        System.out.println ("* SQL error! Changes aborted... *");
}
```



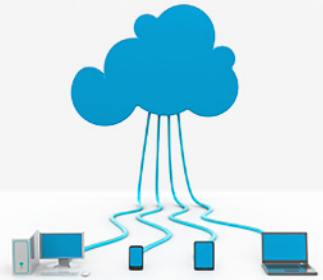
JDBC

Meta Data

- Meta data is 'data about data'

- Two categories of meta data available through the JDBC API:
  1. data about *ResultSet* objects
     (i.e, data about the rows and columns returned by a query)

  2. data about the database as a whole



JDBC

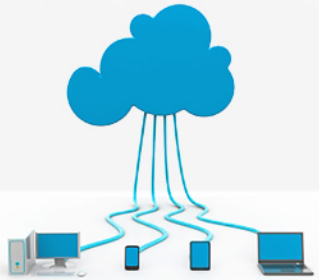
Meta Data

Data about **DB** as a whole:

- is provided by interface *DatabaseMetaData*
- is an object is returned by the *Connection* method *getMetaData*

Data about *ResultSet* objects:

- is provided by interface *ResultSetMetaData*
- is an object is returned by the *ResultSet* method *getMetaData*
- includes:
  - ✓ the number of fields / columns in a *ResultSet* object
  - ✓ the name of a specified field
  - ✓ the data type of a field
  - ✓ the maximum width of a field
  - ✓ the table to which a field belongs



JDBC

Meta Data

Merhods of ResultSetMetaData to access properties of data fields:

- int **getColumnCount** ( )
- String **getColumnName** (*column number*)
- int **getColumnType** (*column number*)
  returns an int const of SQL types
- String **getColumnTypeName** (*column number*)
  returns the DB-specific type name in String
- String **getTableName** (*column number*)