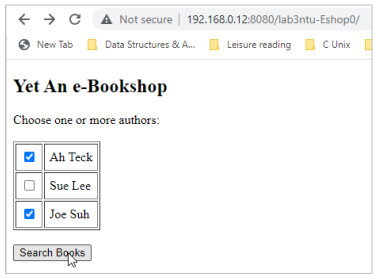
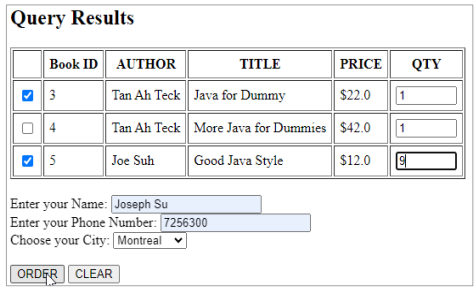
Main topics (tentative)

- Web App Architecture. Preliminary knowledge/Review
  - Client side: HTML CSS JavaScript
  - Java, UML, design patterns, relational databases & SQL
- Client-Server, low level: socket programming, RMI
- Web applications (server side)
  - LAMP/CGI
  - Java Servlet
  - JSP, JavaBean, MVC pattern
  - Database access: JDBC
  - More: listener, filter, Ajax, JSON
- Web (RESTful) services, micro services
- Advanced topics (TBD): Deployments: Docker container, Node JS, React, Angular, SpringBoot
- Other advanced topics (TBD)






```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

      String authors[] = request.getParameterValues("author");
      Table t = new Table();
      ArrayList<Book> resu = t.search(authors);

       /* for (Book e : resu) {
         printWritter…..(e.id);
          printWritter…..(e.author );
          …
      }  */

      request.setAttribute("searchedBooks", resu);
      String target = "bookResultView.jsp";
      request.getRequestDispatcher(target).forward(request, response);
```

Relational database and SQL (brief review)

4

4

Relational database and SQL (brief review)

- Another way to maintain data persistency -- database
- SQL: Structured Query Language
  - The standard for relational database management systems (RDBMS)
  - RDBMS: A database management system that manages data as a collection of tables in which all relationships are represented by common values in related tables

5

5

Relational database and SQL (brief review)

- **Relational database**
  - Logical representation of data, not necessarily the way the data is stored
  - Table
    - Rows (entities), columns (attributes)
  - Primary key (column or group of columns)
    - Unique value for each row
    - Not every table has a primary key

- **SQL statement**

  Query (which data to select from table or tables)

6

6

SQL Environment

- Catalog

  A set of schemas that constitute the description of a database
- Schema

  The structure that contains descriptions of objects created by a user (base tables, views, constraints)

- Data Definition Language (DDL)

  Commands that define a database, including creating, altering, and dropping tables and establishing constraints

- Data Manipulation Language (DML)

  Commands that maintain and query a database

- Data Control Language (DCL)

  Commands that control a database, including administering privileges and committing data

7

7

DDL - Data Definition Language

Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

Command	Description
SELECT	Retrieves certain records from one or more tables.
INSERT	Creates a record.
UPDATE	Modifies records.
DELETE	Deletes records.

DCL - Data Control Language

Command	Description
GRANT	Gives a privilege to user.
REVOKE	Takes back privileges granted from user.

8

8

Another view: SELECT as DQL

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

9

9

Tables in SQL

Table name

Attribute names

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Tuples or rows

10

SQL DataTypes

TABLE 6-2 Sample SQL Data Types

String	CHARACTER (CHAR)	Stores string values containing any characters in a character set. CHAR is defined to be a fixed length.
	CHARACTER VARYING (VARCHAR or VARCHAR2)	Stores string values containing any characters in a character set but of definable variable length.
	BINARY LARGE OBJECT (BLOB)	Stores binary string values in hexadecimal format. BLOB is defined to be a variable length. (Oracle also has CLOB and NCLOB, as well as BFILE for storing unstructured data outside the database.)
Number	NUMERIC	Stores exact numbers with a defined precision and scale.
	INTEGER (INT)	Stores exact numbers with a predefined precision and scale of zero.
Temporal	TIMESTAMP TIMESTAMP WITH LOCAL TIME ZONE	Stores a moment an event occurs, using a definable fraction-of-a-second precision.Value adjusted to the user's session time zone (available in Oracle and MySQL)
Boolean	BOOLEAN	Stores truth values: TRUE, FALSE, or UNKNOWN.

12

UNIVERSITÉ UNIVERSITY

12

DDL: CREATE, ALTER, DROP

```
CREATE TABLE Customer_T
        (CustomerID              NUMBER(11,0)      NOT NULL,
         CustomerName            VARCHAR2(25)      NOT NULL,
         CustomerAddress         VARCHAR2(30),
         CustomerCity            VARCHAR2(20),
         CustomerState           CHAR(2),
         CustomerPostalCode      VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

```
CREATE TABLE Order_T
        (OrderID                 NUMBER(11,0)      NOT NULL,
         OrderDate               DATE DEFAULT SYSDATE,
         CustomerID              NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

```
CREATE TABLE Product_T
        (ProductID               NUMBER(11,0)      NOT NULL,
         ProductDescription      VARCHAR2(50),
         ProductFinish           VARCHAR2(20)
                    CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                 'Red Oak', 'Natural Oak', 'Walnut')),
         ProductStandardPrice    DECIMAL(6,2),
         ProductLineID           INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

```
CREATE TABLE OrderLine_T
        (OrderID                 NUMBER(11,0)      NOT NULL,
         ProductID               INTEGER           NOT NULL,
         OrderedQuantity         NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

13

13

DDL: CREATE, ALTER, DROP

ALTER TABLE

ALTER TABLE table_name alter_table_action;

ADD [COLUMN] column_definition
ALTER [COLUMN] column_name **SET DEFAULT** default-value
ALTER [COLUMN] column_name **DROP DEFAULT**
DROP [COLUMN] column_name **[RESTRICT] [CASCADE]**
ADD table_constraint

e.g.

```
ALTER TABLE CUSTOMER_T
ADD COLUMN CustomerType VARCHAR2 (2) DEFAULT "Commercial";
```

DROP TABLE `DROP TABLE table_name;`

14 `DROP TABLE CUSTOMER_T;`

14

Data Manipulation Language (DML)

Allows data retrieval and modification:

- Data retrieval
  - Select
- Data modification
  - Insert
  - Delete
  - Update

CURD: **CREATE**, **UPDATE**, **READ** and **DELETE**. These terms describe the four essential operations for creating and managing persistent data elements, mainly in relational and NoSQL databases.
CREATE: Insert READ: Select
UPDATE: Update DELETE: delete

15

SQL Query --- SELECT

Used for queries on single or multiple tables

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

16

Simple SQL Query

```
SELECT  *
FROM  Product
```

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

"selection"

17

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Manufacturer
FROM    Product
WHERE   Price > 100
```

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

18

More on SELECT Statement

Clauses of the SELECT statement:

- **SELECT**
  - List the columns (and expressions) to be returned from the query
- **FROM**
  - Indicate the table(s) or view(s) from which data will be obtained
- **WHERE**
  - Indicate the conditions under which a row will be included in the result
- **ORDER BY**
  - Sorts the result according to specified criteria
- **GROUP BY**
  - Indicate categorization of results
- **HAVING**
  - Indicate the conditions under which a category (group) will be included

21

SQL statement processing order (based on van der Lans, 2006 p.100)

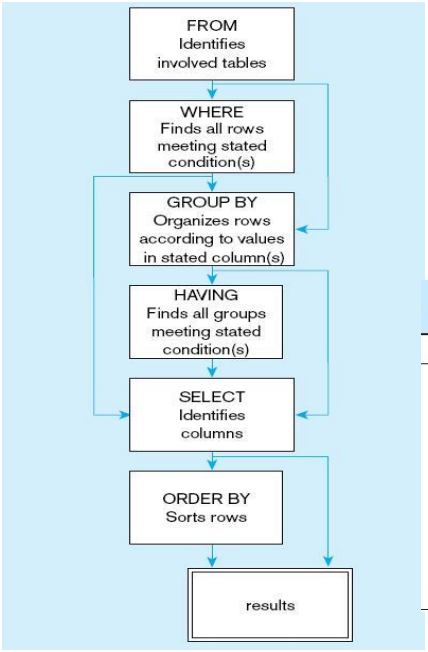


TABLE 6-3 Comparison Operators in SQL

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
!=	Not equal to

22

22

Eliminating Duplicates

```
SELECT   DISTINCT category
FROM     Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT   category
FROM     Product
```

Category
Gadgets
Gadgets
Photography
Household

23

Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='gizmo' AND price > 50
ORDER BY price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

24

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

⇨ ?

```
SELECT   Category
FROM     Product
ORDER BY  PName
```

⇨ ?

```
SELECT   DISTINCT category
FROM     Product
ORDER BY PName
```

⇨ ?

25

The **LIKE** operator

```
SELECT   *
FROM     Products
WHERE    PName LIKE '%gizmo%'
```

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

26

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Foreign key

27

Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

Join between Product and Company

28

Joins

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

PName	Price
SingleTouch	$149.99

29

Aggregation

```
SELECT  avg(price)
FROM    Product
WHERE   maker="Toyota"
```

```
SELECT  count(*)
FROM    Product
WHERE   year > 1995
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

30

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(category)
FROM    Product
WHERE   year > 1995
```

same as Count(*)

We probably want:

```
SELECT  Count(DISTINCT category)
FROM    Product
WHERE   year > 1995
```

31

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```

→ 50 (= 20+30)

33

DDL: Insertions

General form:

```
INSERT INTO R(A1,…., An) VALUES (v1,…., vn)
```

Example: Insert a new purchase to the database:

```
INSERT INTO Purchase(buyer, seller, product, store)
        VALUES ('Joe', 'Fred', 'wakeup-clock-espresso-machine',
                'The Sharper Image')
```

Missing attribute → NULL.
May drop attribute names if give them in order.

34

Insertions

```
INSERT INTO PRODUCT(name)

   SELECT DISTINCT Purchase.product
   FROM    Purchase
   WHERE   Purchase.date > "10/26/01"
```

The query replaces the VALUES keyword.
Here we insert *many* tuples into PRODUCT

35

Insertion: an Example

Product(name, listPrice, category)
Purchase(prodName, buyerName, price)

prodName is foreign key in Product.name

Suppose database got corrupted and we need to fix it:

Product

name	listPrice	category
gizmo	100	gadgets

Purchase

prodName	buyerName	price
camera	John	200
gizmo	Smith	80
camera	Smith	225

Task: insert in Product all prodNames from Purchase

36

DDL: Deletions

Example:

```
DELETE   FROM   PURCHASE

WHERE    seller = 'Joe'   AND
         product = 'Brooklyn Bridge'
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

39

DDL: Updates

Example:

```
UPDATE   PRODUCT
SET    price = price/2
WHERE  Product.name  IN
                (SELECT product
                  FROM    Purchase
                  WHERE  Date ='Oct, 25, 1999');
```

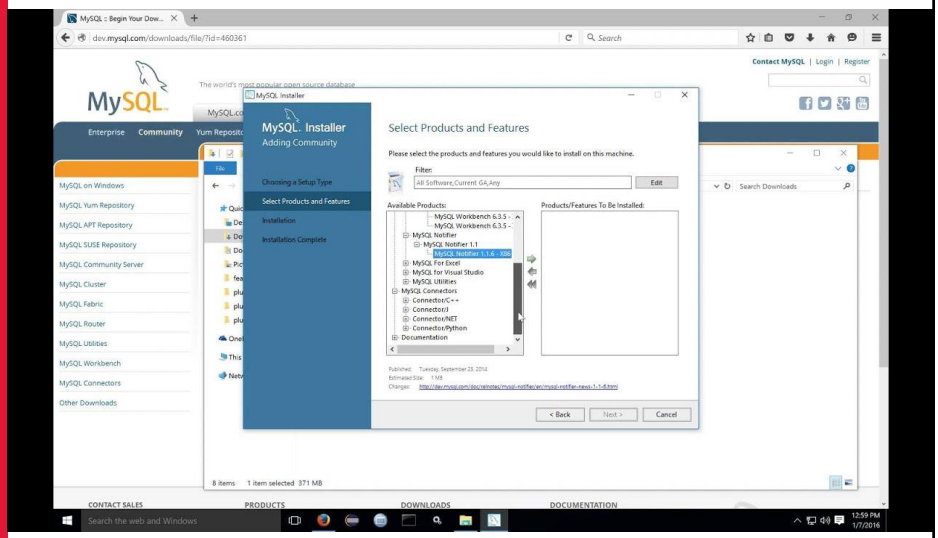
40

41

MySQL server

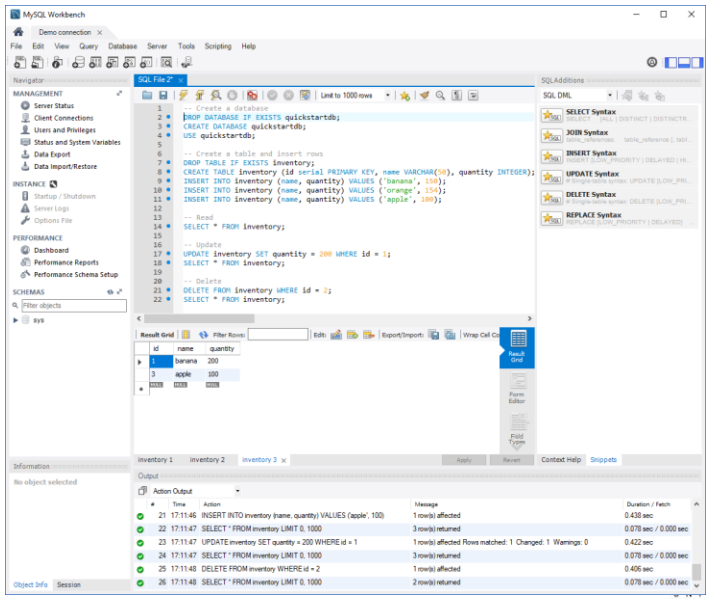
- We will play with two flavors of databases:
  - MySQL
  - SQLite

MySQL server



Windows: can install as service

MySQL Workbench



44

44

SQLite: embedded database

- scaled down version of SQL
- "Embedded RDBMS"
- Implements most of SQL92
- ACID Compliant
  - Implements serializable transactions that are atomic, consistent, isolated, durable (ACID)
- Not a client/server architecture
- Serverless
- Zero-configuration
- Faster than popular client/server database engines form most common operations.
- Cross platform Unix (Linux, Mac), Windows.

- May not good for product, but good for development /learning

45

45

SQLite just works – Serverless, zero configuration

Serverless

- Most SQL database, including MySQL, are implemented as a separate server process. Programs that want to access the database communicate with the server using some kind of interprocess communication (typically TCP/IP) to send request to the server and to receive back results.
- SQLite does not work this way. In SQLite, the process that wants to access the database reads and writes directly from the database files on disk.
- There is no intermediary sever process.
  - no separate server process to install. Set up,. Configure ..
  - Any program that is able to access the disk is able to use an SQLite database.

Zero configuration

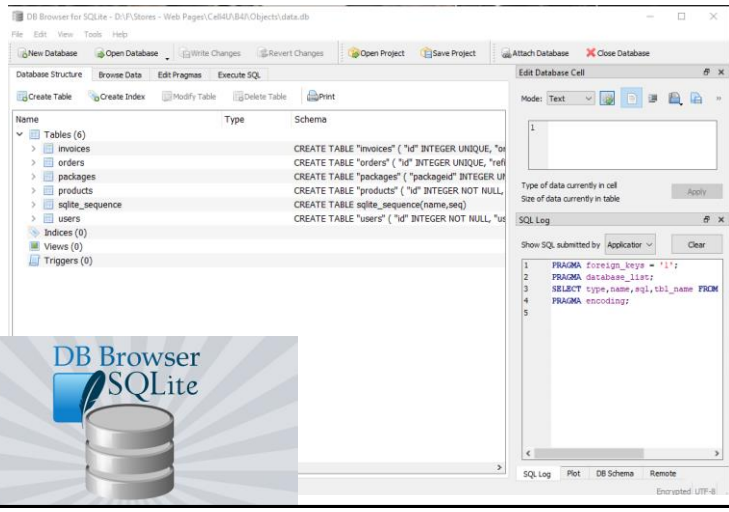
- SQLite does not need to be installed before it is used
- No setup procedure
- Uses no configuration files

46

46

Tools for interacting with SQLite databases

- Command line *sqlite3*
- DB browser for SQLite
- SQLite studio
- …



47

Next topic

- How to use Java application to talk to database?
- How to use Java web application to talk to databases?

48

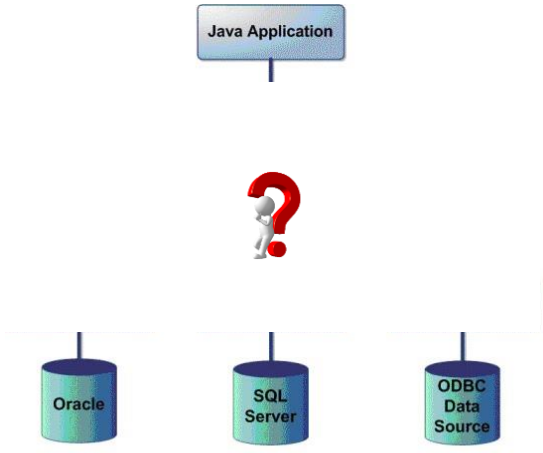
JDBC

- JDBC Introduction
  - Basics
  - Scrollable ResultSet
  - Metadata
- Improve: PreparedStatement
- Improve: Data source
- SQLite
- DAO Design pattern
- SQL injection

49

49

How can Java talk with different databases



52

52

What is JDBC

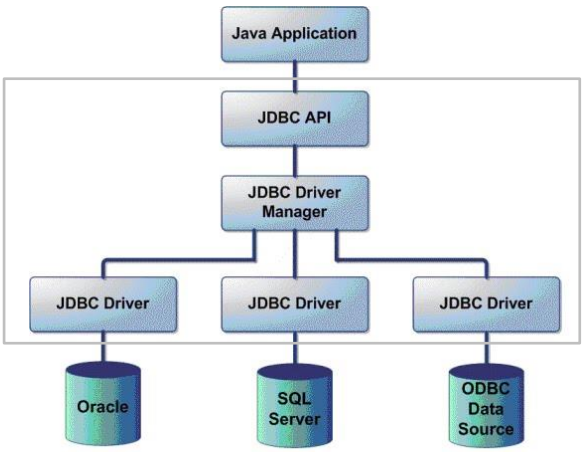
- JDBC: acronym of Java DataBase Connectivity; though Sun Microsystems claims that it is not the full form.
- JDBC is a standard Java API for <u>independent</u> database connection between a Java program and with wide range of relational database -- in a uniform way
- provides programming tools for DB access that are **platform independent *and* vendor independent**.
  - Run on MAC, Linux, Window, read data from MS Access, Oracle, MySQL, embedded DB …

53

53

JDBC
Definition

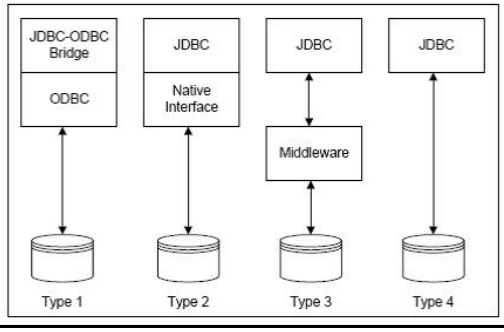
- JDBC: Java Database Connectivity
  - It provides a standard library for Java programs to connect to a database and send it commands using SQL
  - It generalizes common database access functions into a set of common classes and methods
  - Abstracts vendor specific details into a code library making the connectivity to multiple databases transparent to user
- JDBC API Standardizes:
  - Way to establish connection to database
  - Approach to initiating queries
  - Method to create stored procedures
  - Data structure of the query result



JDBC driver: translate generalized JDBC cal
into vendor specific database calls

Types of JDBC driver

- JDBC uses drivers to translate generalized JDBC calls into vendor-specific database calls
  - Drivers exist for most popular databases
  - Four Classes of JDBC drivers exist
    - Type 1; JDBC-ODBC bridge driver.
    - Type 2; native API partly java driver.
    - Type 3; net protocols all java driver.
    - Type 4; native protocols all java driver.



56

56

JDBC Driver Types

Type 1: translates JDBC to ODBC;
relies on ODBC driver to communicate with database, thus requires deployment and configuration of an ODBC driver;
SUN includes JDBC/ODBC bridge with JDK;
ok for playing with the concepts; never intended for production use.

Type 2: written partly in Java, partly in native code;
native code communicates with database and client API;
thus requires installation of some platform-specific code in addition to Java libraries.

*Type 3:*pure Java client library;
uses database independent protocol to communicate db requests to server;
server translates requests into db specific protocol;
client independent of actual db, thus easier to deploy than type 2.

Type 4: pure Java library that translates JDBC requests directly to db specific protocol.

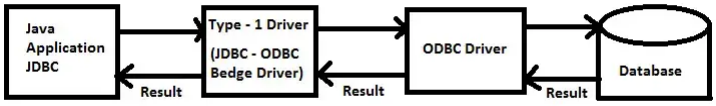
60

60

Type-1 driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
- The ODBC bridge driver is needed to be installed in individual client machines.
- Type-1 driver isn't written in java, that's why it isn't a portable driver.
- This driver software is built-in with JDK so no need to install separately.
- It is a database independent driver.



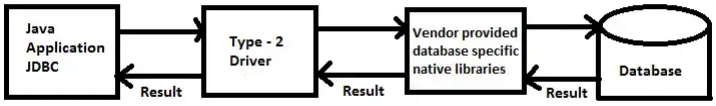
61

61

Type-2 driver

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

- Driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on client machine.
- Type-2 driver isn't written in java, that's why it isn't a portable driver
- It is a database dependent driver.



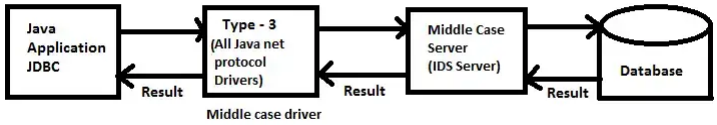
62

62

Type-3 driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Network support is required on client machine.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
- Switch facility to switch over from one database to another database.



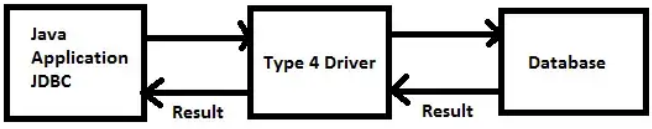
63

63

Type-4 driver

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.



64

64

JDBC Interface classes

- Java.sql package
  - Driver
  - DriverManager
  - Connection
  - Statement
  - PreparedStatement
  - CallableStatement
  - ResultSet
  - ResultSetMetaData
  - DatabaseMetatData

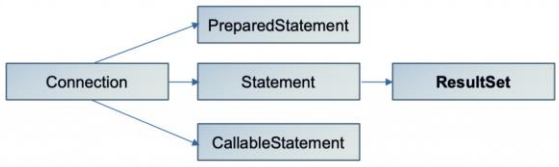
✓ JDBC API defines the interfaces.
✓ Drive vendors provide implementation of the interfaces
✓ Programmers use the interfaces

- **Driver Manager:** This class manages a list of database drivers.
  - It provides static, 'factory'methods such as **registerDriver**() and **getConnection**().
- **Driver:** This interface handles the communication with the database server. Translates API calls into operations for specific database.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
  - It provides methods such as **close**(), **commit**(), **createStatement**(), **prepareStatement**() etc.

66

JDBC Interface classes

- Java.sql package
  - Driver
  - DriverManager
  - Connection
  - Statement
  - PreparedStatement
  - CallableStatement
  - ResultSet
  - ResultSetMetaData
  - DatabaseMetatData



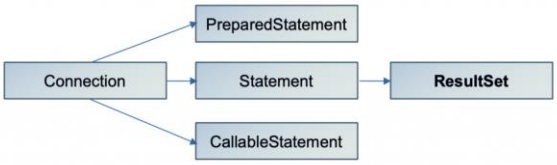
- **Statement:** You use objects created from this interface to submit the SQL statements to the database.
  - It provides methods such as **executeQuery**(), **executeUpdate**() **executeBatch**(), etc. to execute the statements.

- **PreparedStatement:** This represents a precompiled SQL statement. An SQL statement is compiled and stored in a prepared statement and you can later execute this multiple times
  - can get an object of this interface using the method of the Connection interface named **prepareStatement**().

67

JDBC Interface classes

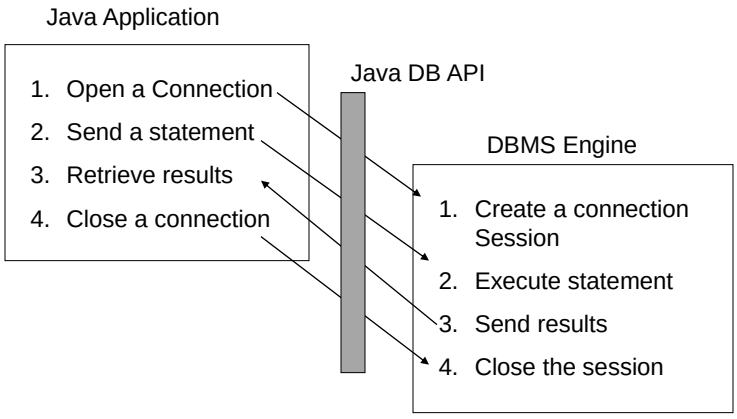
- Java.sql package
  - Driver
  - DriverManager
  - Connection
  - Statement
  - PreparedStatement
  - CallableStatement
  - ResultSet
  - ResultSetMetaData
  - DatabaseMetatData



- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **ResultSetMetaData** This interface is used to get the information about the result set such as, number of columns, name of the column, data type of the column, schema of the result set, table name, etc
  - It provides methods such as **getColumnCount**(), **getColumnName**(),
- **DatabaseMetaData** This interface is used to get the information about the database schema such as username, table name

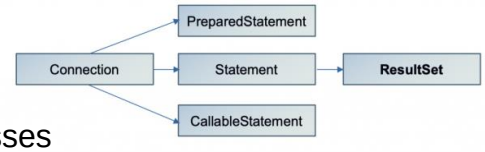
68

What you expect in DB API

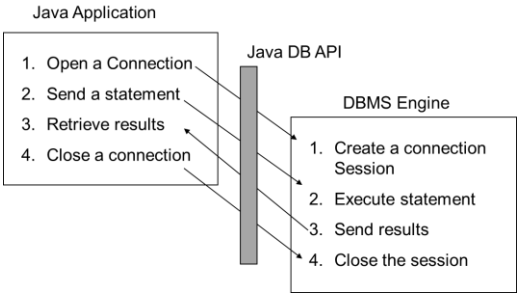


69

Steps to connect?



- Import the necessary classes
- Load the JDBC driver
- Established the connection
- Create the Statement object (from connection).
- Send/Execute a query (on statement).
- Process the results.
- Close the connection.



70

70

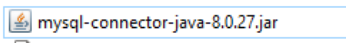
Steps to connect?

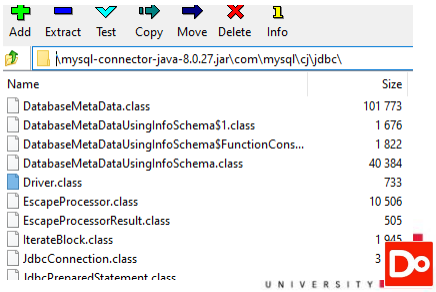
1. Dynamically loads vendor specific driver class.

   Class.forName();

   For MySql driver:-

   Class.forName("com.mysql.cj.jdbc.Driver");





At run time the class loader locates the driver class and loads it
The name of the driver is a literal string thus the driver does not need to be present at compile time

71

71

Steps to connect?

1. Dynamically loads vendor specific driver class
   Class.forName();

2. Establish the connection // static, 'factory' method
   Connection con = DriverManager.getConnection
   ("url", "user_name", "passwd")

72

72

Steps to connect?

1. Dynamically loads vendor specific driver class
   Class.forName();

2. Establish the connection // static, 'factory' method
   Connection con = DriverManager.getConnection
   ("url", "user_name", "passwd")

3. Create the Statement object:
   Statement stmt = con.createStatement();

73

73

Steps to connect?

4. Execute the query
   Two kind of query: Query and Update

   For the SELECT query:
   *String sqlQuery = "SELECT * FROM EMP";*
   stmt.executeQuery (sqlQuery)

   For the INSERT/UPDATE/CREATE/DELETE query:
   String sqlQuery = "INSERT INTO EMP VALUE (47, "TEDDY')";
   stmt.executeUpdate (sqlQuery)

   String sqlQuery = "Create table ABC (Age…)";
   stmt.executeUpdate (sqlQuery)

74

74

Steps to connect?

5. Process the result:
   executeQuery() returns *ResultSet*, which represents a table of data returned by a Statement object.
   - It maintains a cursor which initially located before the first row.
   - need to move the cursor to the first row of data e.g. next() method)

```
ResultSet rs = stmt.executeQuery (sql);
while (rs.next()){   // first time, make it point to the first row
  System.out.println( rs.getString(1) );  // column index start 1
  System.out.println( rs.getInt(2) );
}
```

75

75

Steps to connect?

5. Process the result:
   executeQuery() returns *ResultSet*, which represents a table of data returned by a Statement object.
   - It maintains a cursor which initially located before the first row.
   - need to move the cursor to the first row of data e.g. *next()* method)

```
ResultSet rs = stmt.executeQuery (sql);
while (rs.next()){   // first time, make it point to the first row
  System.out.println( rs.getString(1) );  // column index start 1
  System.out.println( rs.getInt(2) );
}
```

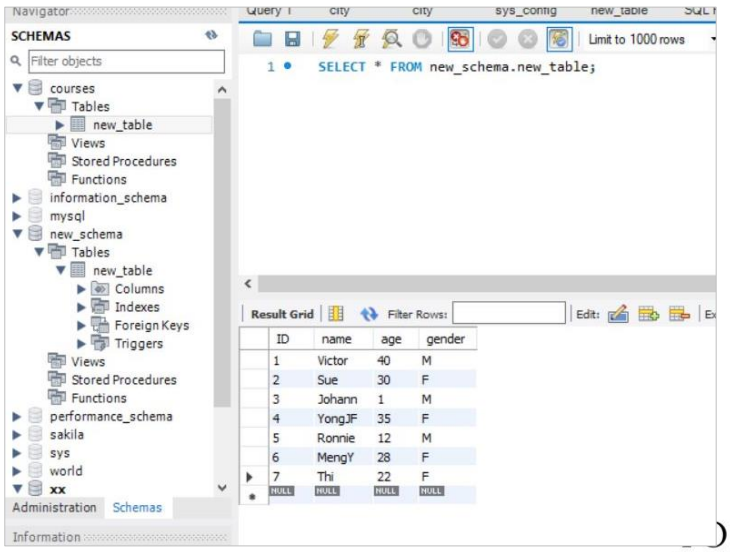
6. Close the connection
   release all the resources that the connection is holding

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

76

76

An example

- 

77

77

An example

```
import java.sql.*;

Class.forName("com.mysql.cj.jdbc.Driver");   // may not need actually

Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/new_schema","root","Yu26607");

Statement stmt=con.createStatement();



ResultSet rs=stmt.executeQuery("select * from new_table where gender='F' ");

while(rs.next())
  System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+ rs.getInt(3)+"\t"
        +rs.getString(4));

   con.close();

} catch(Exception e){ System.out.println(e);}
```

Load driver

Connect db

Create statement

Execute query

Get results

Close connection

2	Sue	30	F
4	YongJF	35	F
6	MengY	28	F
7	Thi	22	F

78

78

An example

```
import java.sql.*;

Class.forName("com.mysql.cj.jdbc.Driver");   // may not need actually

Connection con = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/new_schema","root","Yu26607");

Statement stmt=con.createStatement();

stmt.executeUpdate("insert into new_table (ID, name, age, gender) VALUES ('18', 'V Yu',
'23', 'F')");

ResultSet rs=stmt.executeQuery("select * from new_table where gender='F' ");

while(rs.next())
  System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+ rs.getInt(3)+"\t"
        +rs.getString(4));

   con.close();

} catch(Exception e){ System.out.println(e);}
```

2	Sue	30	F
4	YongJF	35	F
6	MengY	28	F
7	Thi	22	F
18	V yu	23	F

79

79