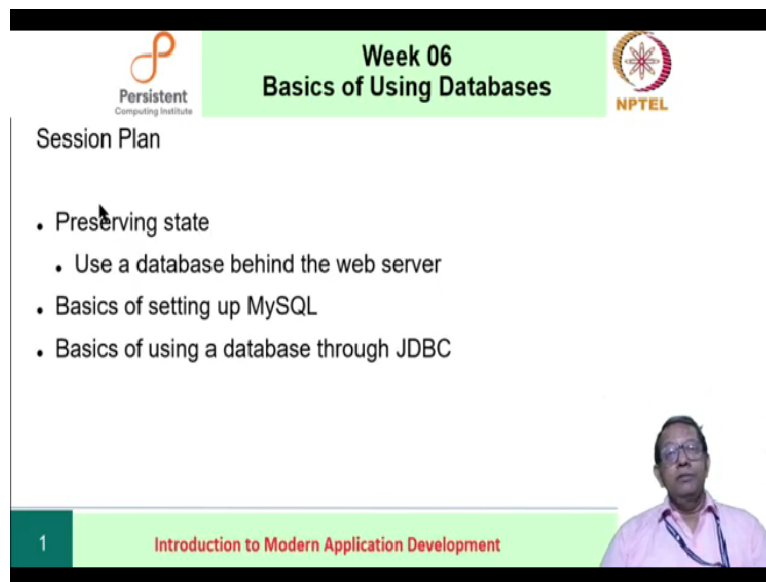


Introduction to Modern Application Development
Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Abhijat Vichare
Persistent Computing Institute
Madhavan Mukund
Chennai Mathematical Institute

Lecture-21
Introduction to JDBC

(Refer Slide Time: 00:12)

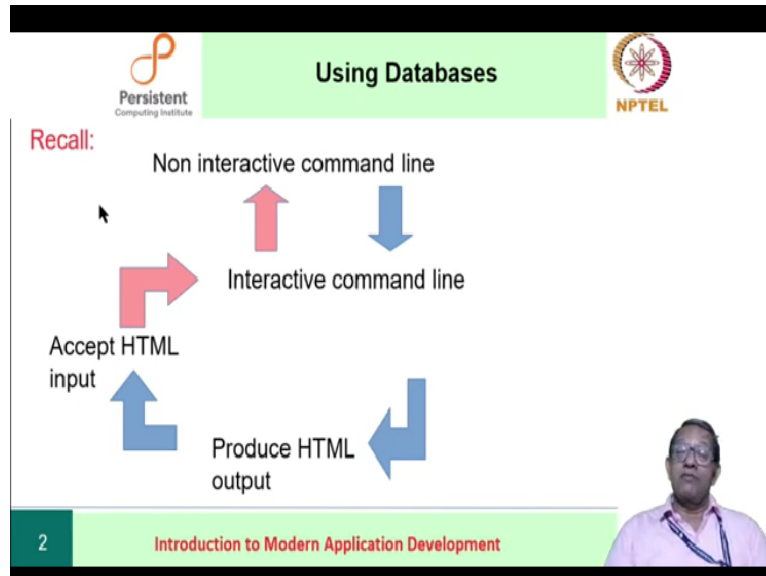


The slide features a header with the Persistent Computing Institute logo on the left, the text "Week 06 Basics of Using Databases" in the center, and the NPTEL logo on the right. Below the header, the "Session Plan" is listed with four bullet points: "Preserving state", "Use a database behind the web server", "Basics of setting up MySQL", and "Basics of using a database through JDBC". At the bottom left, a small box contains the number "1" and the text "Introduction to Modern Application Development". On the bottom right, there is a video feed of Prof. Aamod Sane.

Hello, everyone, welcome to the 6th week of the course on introduction to modern applications development. We have come almost halfway through the course. Last session, we saw how to create a very rudimentary web application. In this session, we will look at the basic aspects of including a database in our web application. The plan is basically as follows. We will look at the idea of preserving state.

This will take us back to our very first sessions, we will look at some basics of setting up my SQL which is the database that we will use. And we will also look at how to use JDBC in order to use the database that we set up.

(Refer Slide Time: 01:22)



Let us take a look back right from the beginning at what we have done. We started with actually a spreadsheet that implemented the basic algorithm of our solution. We then went on to create a non interactive command line Java application that implemented the algorithm in Java. **The non interactive meant that all information required for execution of the program was given before the program started.**

Our command line therefore looked like for example, *"fair share register f1 f2"*, another invocation of the command could be *"fair share expense f2 200"*. A third invocation of the command would be *"fair share report f1"*. These were 3 distinct invocations of the command. And at every point, at every invocation, we gave all the information required to execute that.

We then went on to create a command line, but an interactive version of a program. Oh, but before that, when we had a completely non interactive version command line version of our program, it was we required some mechanism to remember what the previous invocation of the programs have done so far. So, when for example, our current invocation was to request a report for see friend f1.

Then somehow, we had to remember what the previous invocations of the proof fair share program had done. And at that point we had used a file as a way of remembering the information

across various invocations of the program. In contrast, when you take an interactive command line program, in an interactive command line program, we are going to give a sequence of commands, which together represent an entire interaction by the group of friends.

So, when we after starting our fair share program, we have a sequence of commands register f1 f2. expense f2 200, report f1. And finally, a new command **end** which told us that there are no more commands coming, once all the commands are received by the fair share program while was running it processed each command individually as before and gave output as required. So, finally, for example, if report f1 was supposed to be given out, the only thing that the command did was to show what was the money that was either owed or to be received by friend f1. That is all that happened.

Contrast these 2 interactions, the non interactive command line part and the interactive one. The interactive one really did not require any external mechanism to remember what was happening. Every invocation of the command was self contained. It started from the beginning of the interaction of the roommates right up till the end, the entire secret quince was done in one go – in the interactive version of our program.

In such a scenario, there is no real need to remember in information across individual commands, because that is anyway a part of the entire program. However, just as we said a couple of moments ago, in a non interactive command line version of the program we would need an ability to remember what previous invocations of the command had possibly done. Well, we will now come back to that situation in a short while.

But continuing, after the interactive command line version of the program we then graduated to decoupling the input and output from our processing, the view and the model part, as Professor Sane would have put it. We did that in 2 stages. We first changed our interactive command line version to produce an HTML page as an output. So, what earlier was simply an output on a terminal was now an HTML page, which could be displayed anywhere on the internet.

The next phase was to introduce the ability to acquire input via HTML. This generalization gave us our very elementary web application where the input was on some node on the internet. Similarly, the output was on some browser on the internet, and the processing was done on our server. So, on the browser of the user you had a form on which commands were typed. These commands were sent to the server. The server actually had our command line, Java program, or interactive version of our Java program. It simply picked up those commands executed them just as before, produced an HTML output which was returned by the server back to the browser that requested.

This completed the whole basic cycle of a web application. But then the problem with this has been that it is using a forms interface where the entire bunch of command is given at one single piece of input to the model or the Java program on the server, it was a command line interactive program.

What we eventually desire is to have individual commands being sent to our server, which will perform the fair share computations as required by the command and return the results, if any. In other words, what we really want is the input and output separated as before using HTML. But the model or the program that runs on the server, we actually want it to be a non-interactive command line program.

Each request from the user whether it is a register, or an expense, or a report is an independent request being sent to – via the web – to our fair share program running on the server. The fair share program will execute each request and keep on and go on to the next request. Because the conceptually now our program is going to be invoked for every individual request, on the server side, we are really thinking of multiple invocations of our fair share program.

But with different commands at each invocation exactly similar to our non interactive command line program. Therefore, just as before when we argued about non interactive command line programs, and we thought that we needed a file to remember whatever information was processed in the previous invocations in the same way, we will now need to do that on this web application system.

It is for this, that we use databases. In the next few slides, we will again, jog our memory a little bit.

(Refer Slide Time: 11:29)

Using Databases

Recall:

File Edit View Insert Format Styles Sheet Data Tools Window Help

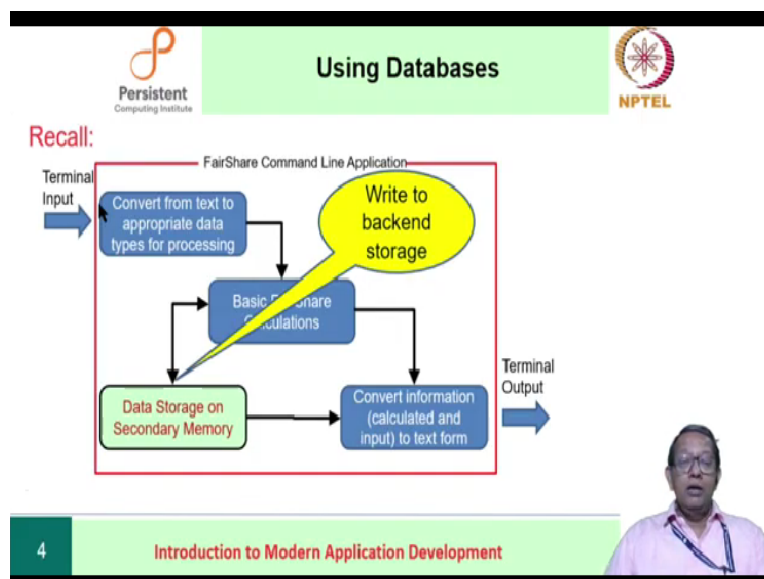
RoommateID RoommateID RoommateID

EventID	Date	Total expense	Per head	F1	F2	F3	F4	F5	F1	F2	F3	F4	F5	Invariant
1		100	20	100	0	0	0	0	80	-20	-20	-20	-20	0
2		100	20	0	100	0	0	0	-20	80	-20	-20	-20	0
3		200	40	0	0	200	0	0	-40	-40	160	-40	-40	0
4		75	15	0	0	0	75	0	-15	-15	-15	60	-15	0
5		150	30	0	0	0	150	-30	-30	-30	-30	120	0	0
6		100	20	100	0	0	0	0	80	-20	-20	-20	-20	0
7		0	0	0	0	0	0	0	0	0	0	0	0	0
8		0	0	0	0	0	0	0	0	0	0	0	0	0
9		0	0	0	0	0	0	0	0	0	0	0	0	0
10		0	0	0	0	0	0	0	0	0	0	0	0	0
11		0	0	0	0	0	0	0	0	0	0	0	0	0
12		0	0	0	0	0	0	0	0	0	0	0	0	0
13		725	145	200	100	200	75	150	55	-45	55	-70	5	0

3 Introduction to Modern Application Development

Here is our first one the spreadsheet, observe that is just laying out data in rows with individual columns talking about individual friends, and their expenses or per head share calculations.

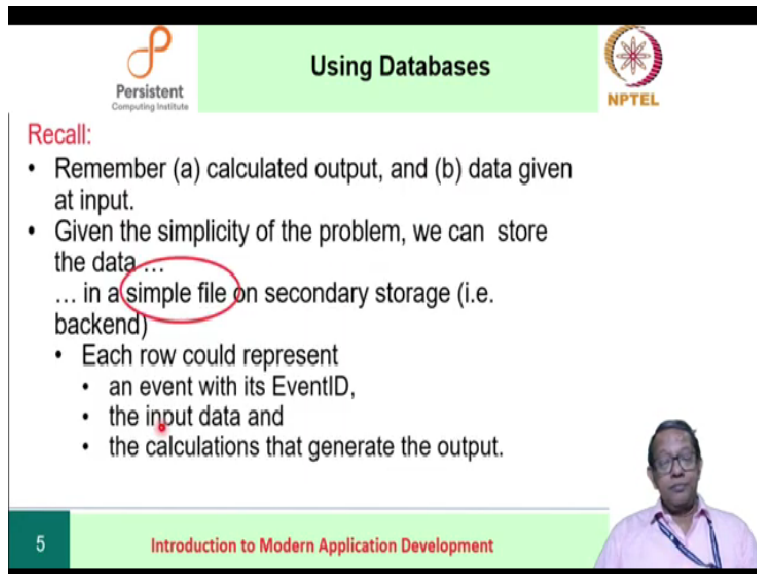
(Refer Slide Time: 11:49)



This is the command line, the sketch of the command line application. In particular we want you to recall this part where we said that we need to write to the backend storage. And this was some

data storage in secondary memory. At that point we had said we will look at databases when we come to that. Well, we have arrived at that point where we now need to consider databases.

(Refer Slide Time: 12:24)

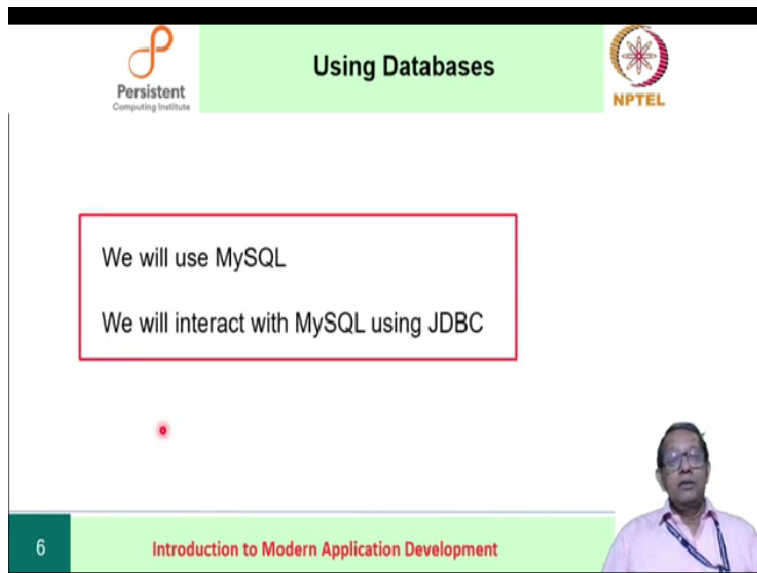


Slide 5: Using Databases. The slide features the Persistent Computing Institute logo on the top left and the NPTEL logo on the top right. The title "Using Databases" is centered at the top. The main content is a "Recall:" section with a bulleted list. The first bullet point is "Remember (a) calculated output, and (b) data given at input." The second bullet point is "Given the simplicity of the problem, we can store the data ... in a simple file on secondary storage (i.e. backend)". The word "simple" in "simple file" is circled in red. The third bullet point is "Each row could represent" followed by a sub-bulleted list: "an event with its EventID," "the input data and," and "the calculations that generate the output." A small video inset of a man in a pink shirt is visible in the bottom right corner. The slide number "5" and the text "Introduction to Modern Application Development" are at the bottom.

- Recall:
- Remember (a) calculated output, and (b) data given at input.
- Given the simplicity of the problem, we can store the data ... in a simple file on secondary storage (i.e. backend)
- Each row could represent
 - an event with its EventID,
 - the input data and
 - the calculations that generate the output.

At that point we had said that we would simply use a file. In that file, also our organization was very similar to the spreadsheet. Each row represented an event. There was input data for that event. And there were per head here calculations for that event.

(Refer Slide Time: 12:51)



Slide 6: Using Databases. The slide features the Persistent Computing Institute logo on the top left and the NPTEL logo on the top right. The title "Using Databases" is centered at the top. The main content is a red-bordered box containing two lines of text: "We will use MySQL" and "We will interact with MySQL using JDBC". A small red dot is visible below the box. A small video inset of a man in a pink shirt is visible in the bottom right corner. The slide number "6" and the text "Introduction to Modern Application Development" are at the bottom.

We will use MySQL

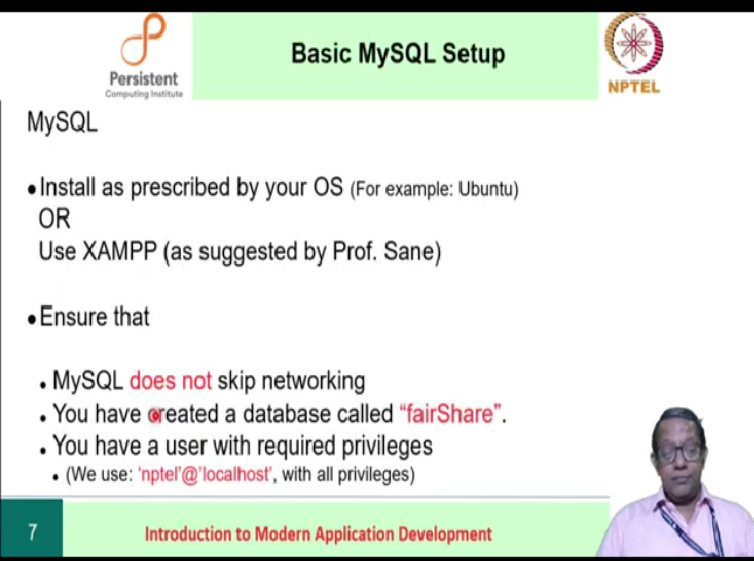
We will interact with MySQL using JDBC

So, we will now use MySQL as the database system that will work along with our HTTP server. And we will interact with the HTTP server using the Java database connectivity (JDBC). In the next few screens, in the next few slides, we will see a few basic ideas about what is required to

get MySQL database working and a JDBC system working. These are bare minimum, some important points that we observe that you could possibly need.

But a full discussion, we suggest you look at the documentation of MySQL or JDBC that comes along with your software.

(Refer Slide Time: 13:58)



Persistent
Computing Institute


Basic MySQL Setup

NPTEL

MySQL

- Install as prescribed by your OS (For example: Ubuntu)
OR
Use XAMPP (as suggested by Prof. Sane)
- Ensure that
 - MySQL **does not** skip networking
 - You have created a database called **"fairShare"**.
 - You have a user with required privileges
 - (We use: 'nptel'@'localhost', with all privileges)

7 Introduction to Modern Application Development



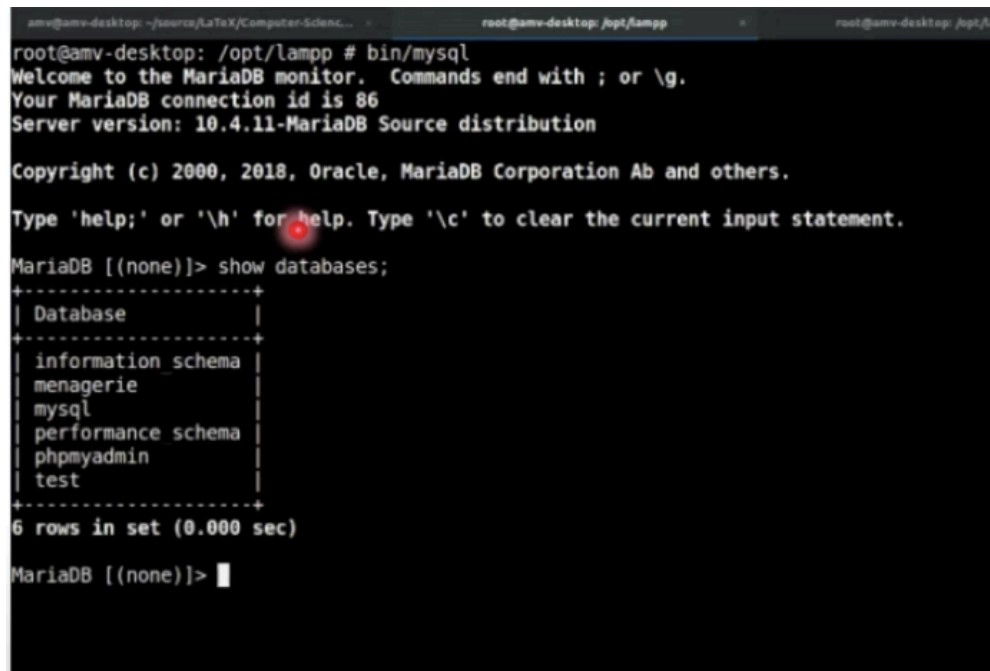
Typical my SQL setup would of course be dependent on your particular OS and has to be done accordingly or you may use XAMPP, as suggested by Professor Sane, which is a overall system, which has MySQL and a lot many other pieces of software typically useful for web development. But it is supposed to be a development system and not a production system. So, XAMPP suggest you use for learning purposes.

The key things that you need to ensure as far as MySQL is concerned are that it does not skip networking, at least in the until the point you understand how to create a very secure MySQL system. You will need to ensure that you have created a database called fairshare. We will shortly show interaction sequence. You also have a user with all the required privileges.

In our example, we will be using 'nptel'@'localhost' as a user. And we would have granted all the privileges that are required. Let us have a look at a basic sequence with of interaction with MySQL. This interaction will set up MySQL for us with our fair share application.

(Video Starts: 15:40)

Here is a screenshot a screen record actually of our interaction with the my SQL system. This illustration is based on xampp that has been set up on my personal web machine.



```
root@amv-desktop: /opt/lampp # bin/mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 86
Server version: 10.4.11-MariaDB Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| menagerie |
| mysql |
| performance_schema |
| phpmyadmin |
| test |
+-----+
6 rows in set (0.000 sec)

MariaDB [(none)]> █
```

We start with the command bin slash my SQL. And voila, that is where you see show databases is going to show the set of databases that are available at this point of time.


```
MariaDB [(none)]> create database fairShareV1;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| fairShareV1 |
| information_schema |
| menagerie |
| mysql |
| performance_schema |
| phpmyadmin |
| test |
+-----+
7 rows in set (0.000 sec)
```

We create a new database as follows create database fair share v1, v1 for version 1. There you go. We now create a table within that database. This table will assume that there are 2 friends. So, event number is an integer, expenses for friend f1 is a double precision 10 digit with 2 decimal digits number. expenses for f2 is again a double precision 10 digit 2 decimal places number per head share of f1.

And per head share of f2 are also double precision 10 digit, 2 decimal places numbers. This is the way one creates a table. You could create as many tables as you need. Well we forgot to select a certain database. We do that by using the command use. Having done that, we can see that the create tables has succeeded. How do we see that the show tables command shows us? Now, we can select all the entries from the events table. We do that from using select star from events. But then there are no entries so far.

```
root@ame-desktop: ~/jupyter/Computer Science... root@ame-desktop: ~/jupyter/jupyter root@ame-desktop: ~/jupyter/jupyter/bin
MariaDB [(none)]> create table events (event_no int, exp_f1 double (10, 2), exp_f2 double (10, 2), phs_f1 double (10, 2), phs_f2 double (10, 2));
ERROR 1046 (3D000): No database selected
MariaDB [(none)]> use fairShareV1;
Database changed
MariaDB [fairShareV1]> create table events (event_no int, exp_f1 double (10, 2), exp_f2 double (10, 2), phs_f1 double (10, 2), phs_f2 double (10, 2));
Query OK, 0 rows affected (0.144 sec)

MariaDB [fairShareV1]> show tables;
+-----+
| Tables_in_fairShareV1 |
+-----+
| events                |
+-----+
1 row in set (0.000 sec)

MariaDB [fairShareV1]> select * from events;
Empty set (0.001 sec)

MariaDB [fairShareV1]> insert into events values (1, 200, 0, 100, -100);
Query OK, 1 row affected (0.023 sec)

MariaDB [fairShareV1]> select * from events;
+-----+
| event_no | exp_f1 | exp_f2 | phs_f1 | phs_f2 |
+-----+
| 1        | 200.00 | 0.00   | 100.00 | -100.00 |
+-----+
1 row in set (0.000 sec)

MariaDB [fairShareV1]> use mysql;
```

How do you insert an event? Well, you insert into events values event number 1 expenses by friend 1 200 expenses by friend 2 0 per head. Share of f 100 and per head share of f2 is - 100. And when you now select star, we can see that, we change our SQL database to MySQL and look at the users table. We want to create a user nptel at local hosts whose password or that is the identification system would be nptel. Notice that we have made a small mistake.

```
slow_log
table_stats
tables_priv
time_zone
time_zone_leap_second
time_zone_name
time_zone_transition
time_zone_transition_type
user
+-----+
31 rows in set (0.000 sec)

MariaDB [mysql]> select user, host from user;
+-----+
| user | host |
+-----+
| root | 127.0.0.1 |
| root | ::1 |
| mysql | localhost |
| pma | localhost |
| root | localhost |
+-----+
6 rows in set (0.000 sec)

MariaDB [mysql]> create user 'nptel'@'localhost' identified by 'nptel';
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'identified by 'nptel'' at line 1
MariaDB [mysql]> create user 'nptel'@'localhost' identified by 'nptel';
Query OK, 0 rows affected (0.000 sec)

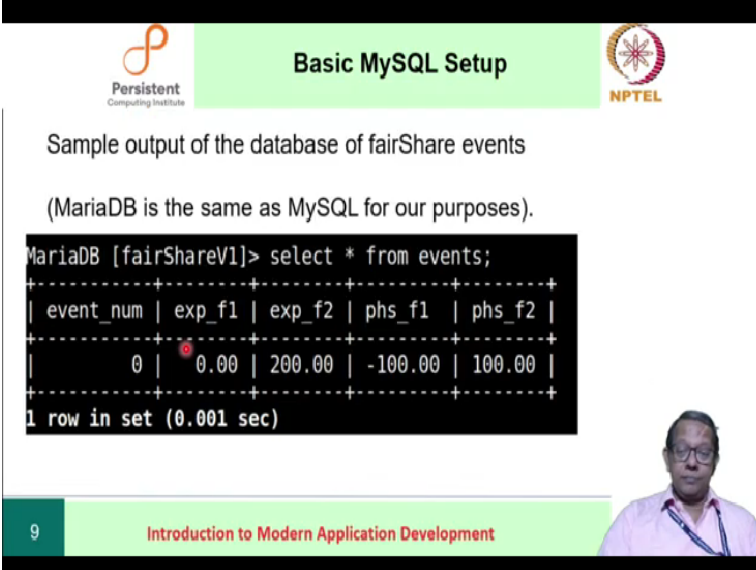
MariaDB [mysql]>
```

Well it is if one looks at it closely it is really just the spelling mistake instead of identifying. I have mistakenly typed it as *identified*. Fortunately, MySQL has told us an error that there is an

error in the SQL syntax and having observed that error, corrected it, this is what happens. A new user npTEL has been created successfully. And we can check that again by performing a select on the user table. There you go. Now you can see the npTEL user. This user also has to be granted the required permissions which we have not shown over here.

(Video Ends: 19:34)

(Refer Slide Time: 19:35)



The slide is titled "Basic MySQL Setup" and features logos for "Persistent Computing Institute" and "NPTEL". It displays a terminal window with the following content:

```
MariaDB [fairShareV1]> select * from events;
```

event_num	exp_f1	exp_f2	phs_f1	phs_f2
0	0.00	200.00	-100.00	100.00

1 row in set (0.001 sec)

Below the terminal window, a small video feed shows a man speaking. The slide footer includes the number "9" and the text "Introduction to Modern Application Development".

Let us have a look at the database. Let us have a look at the table. By the way, MariaDB is the same as my SQL for our purposes. So, although the prompt that you see is MariaDB, think of it as my SQL. It is a very text driven system. We are showing a screenshot of a query select star from events. And there is one event that is shown over here.

What we saw in the interaction sequence just now was a very interactive manual creation of the database, the tables, the entries within the table, and so on and so forth. What we really need to do is to perform some of these tasks via a Java program. So, our Java program will receive commands over the web, execute them compute whatever information is necessary, and then use the database to store information.

So, our Java program will need access to the database access to the particular table within the database and the ability to manipulate that table that is the rows and columns within that table. What we see here is a table with one row.

(Refer Slide Time: 21:25)

Basic MySQL Setup

Sample output of the database of fairShare events

register f1 f2

the same as MySQL for our purposes).

```
MariaDB [fairShareV1]> select * from events;
```

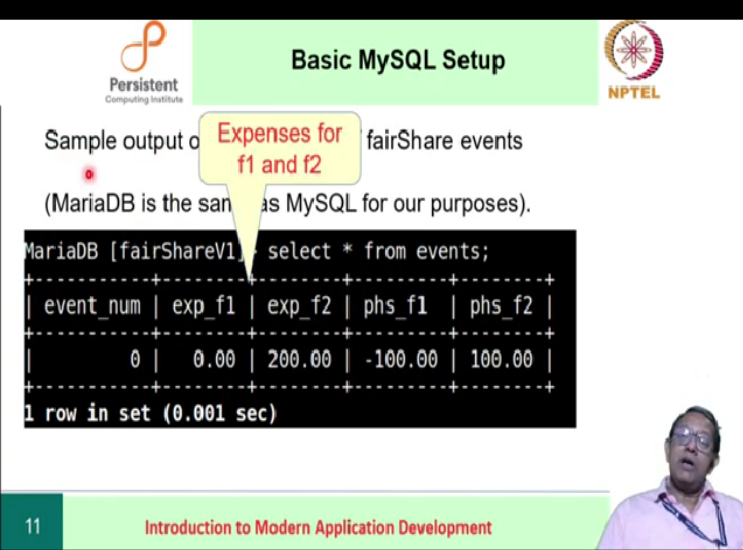
event_num	exp_f1	exp_f2	phs_f1	phs_f2
0	0.00	200.00	-100.00	100.00

1 row in set (0.001 sec)

10 Introduction to Modern Application Development

the table itself with integer even number, double precision 10-digit, 2 decimal places, expenses for friend f1, etc. This the table itself is created when you give the register command. In response to the register command our Java program via the JDBC system should create an empty table which has the event number expenses for each friend and per head share of each friend. This creation of the table is done in response to the register f1 f2 command. We already saw how to create a table manually.

(Refer Slide Time: 22:13)



Basic MySQL Setup

Sample output of the database of fairShare events
(MariaDB is the same as MySQL for our purposes).

```
MariaDB [fairShareV1] select * from events;
```

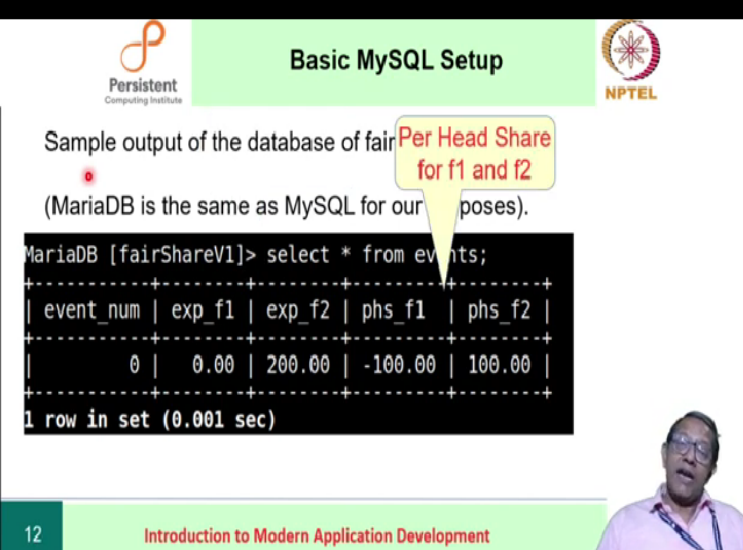
event_num	exp_f1	exp_f2	phs_f1	phs_f2
0	0.00	200.00	-100.00	100.00

1 row in set (0.001 sec)

11 Introduction to Modern Application Development

These columns are expenses for f1 and f2.

(Refer Slide Time: 22:21)



Basic MySQL Setup

Sample output of the database of fairShare events
(MariaDB is the same as MySQL for our purposes).

```
MariaDB [fairShareV1] select * from events;
```


event_num	exp_f1	exp_f2	phs_f1	phs_f2
0	0.00	200.00	-100.00	100.00

1 row in set (0.001 sec)


12 Introduction to Modern Application Development

And these are per head share columns for f1 and f2. Note that the number of columns will depend on the number of friends that you have requested for registration.

(Refer Slide Time: 22:36)



Basic MySQL Setup



Sample output of the database of fairShare events

(MariaDB is the same as MySQL for our purposes).


```

MariaDB [fairShareV1]> select * from events;
+-----+-----+-----+-----+-----+
| event_num | exp_f1 | exp_f2 | phs_f1 | phs_f2 |
+-----+-----+-----+-----+-----+
| 0 | 0.00 | 200.00 | -100.00 | 100.00 |
+-----+-----+-----+-----+-----+
(0.001 sec)
  
```

An event


13

Introduction to Modern Application Development




And event is inserted. We have seen the manual insertion before. In fact, via Java we will actually use the same command and give it to the database except instead of giving it manually, we will construct that command through Java and pass it on to the MySQL database system behind, that essentially is all that Java and database systems require to work together.

(Refer Slide Time: 23:11)



Basic MySQL Setup



Sample Looks like the Spreadsheet! base of fairShare events


(MariaDB is the same as MySQL for our purposes).

```

MariaDB [fairShareV1]> select * from events;
+-----+-----+-----+-----+-----+
| event_num | exp_f1 | exp_f2 | phs_f1 | phs_f2 |
+-----+-----+-----+-----+-----+
| 0 | 0.00 | 200.00 | -100.00 | 100.00 |
+-----+-----+-----+-----+-----+
1 row in set (0.001 sec)
  
```

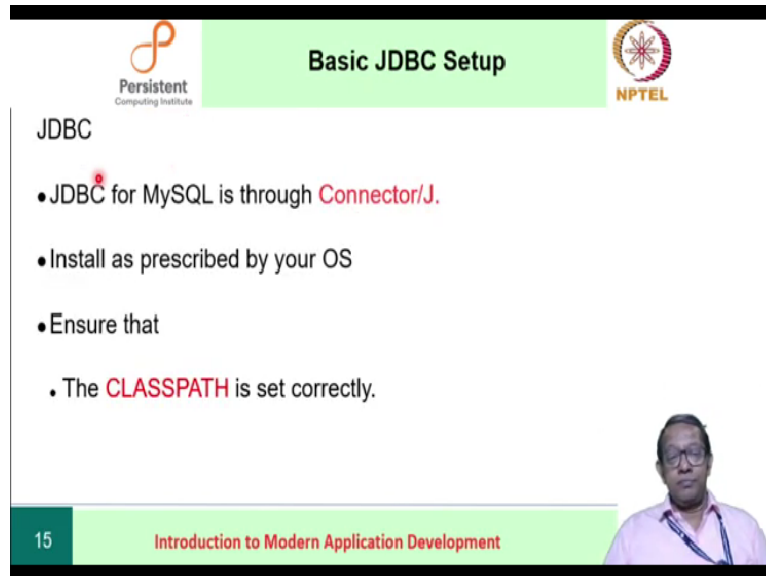
14



Introduction to Modern Application Development



Finally, observe that this table actually looks like a spreadsheet. The views are very, very similar.


(Refer Slide Time: 23:20)



 **Basic JDBC Setup** 

JDBC

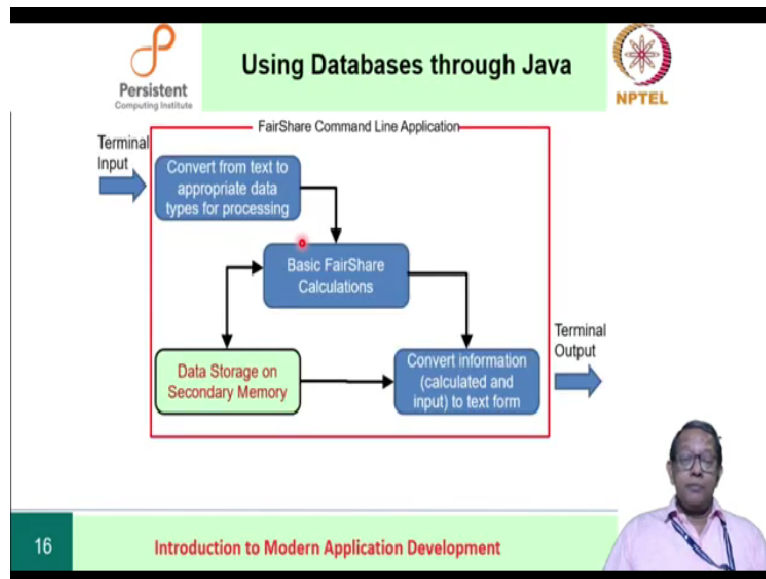
- JDBC for MySQL is through **Connector/J**.
- Install as prescribed by your OS
- Ensure that
 - The **CLASSPATH** is set correctly.

15 Introduction to Modern Application Development 

Here are some notes about basic steps, basic issues, and basic points to remember when you are using JDBC. That is Java database connectivity. In order for Java programs to interact with my SQL, we need a piece of software called as the driver that is usually called as Connector/J, (J for Java). And this is available on the MySQL website.

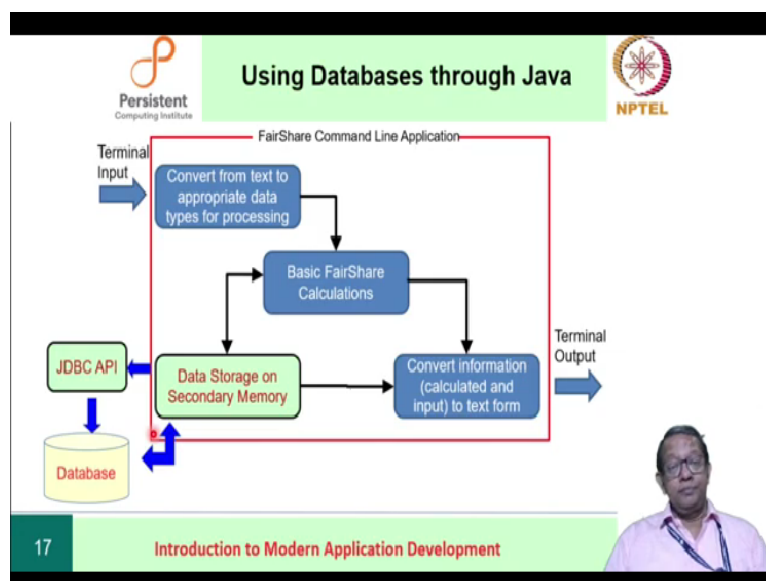
In these slides, I forgot to include the URL for obtaining Connector/J. But in my subsequent slides, or with Professor Sane, we will share the actual URL with you. But it is very easy to find that over the web. Install the connector j system as prescribed by your operating system and ensure that the class path is set correctly. The Connector/J essentially is a Java archive a jar file and the jar file must be put into the class path so that it is accessible to Java programs.

(Refer Slide Time: 24:47)



Okay, let us now go to the final part of this session where we look at how to actually write Java program that interacts with my SQL. Let us start from our command line, the sketch of the command line system. And with a particular focus on data storage in secondary memory. In the earlier session, it was just a box, which was a conceptual box, it just represented an idea. And we now want to implement that idea, realize it in practice.

(Refer Slide Time: 25:28)



So, what we do really is use the JDBC system to interact with the database. Our Java program, the command line Java program, will use Java database API to connect to the MySQL database and use that connection to retrieve or store information into the database. It is conceptually that simple.

(Video Starts: 25:57)

Let us have a look at the actual Java code that can let us have a look at the actual Java code that will illustrate how a Java program interacts with a MySQL database.

At this point, we will assume that there is a MySQL database that has been appropriately created. For example, in a MySQL database there would be a fair share database, which has an events table. Okay, and there is a user nptel with appropriate permissions, all privileges that are required are given. In what follows we are going to make these assumptions. To connect with SQL, you need to import some packages as here.

```
11 import java.sql.Connection;
12 import java.sql.DriverManager;
13 import java.sql.PreparedStatement;
14 import java.sql.ResultSet;
15 import java.sql.SQLException;
16 import java.sql.Statement;
17 import java.sql.Array;
18
```

Let us start from the beginning of this program. This again is, let me state this again, this is a illustration of a procedural style of using Java, there is no object oriented programming here. This is just a procedural style of using Java. So, our program starts with main. And the first thing it does is it connects with to the database using the FSDB object. The FSDB object is a class variable over here. It is an instance of the example JDBC implementation, *exampleJDBC* class.

Here is the example JDBC class. It also includes the packages required for a connecting to databases, in this case my SQL and there are a bunch of methods over here, which will be used as the methods of the FSDB object. That is what we do here. Having instantiated that object, the

first thing we do in main is to connect to a database. Once the connection is done, further operations can be done. For example – For example, you might want to do registration.

This is actually the same code as before, something that we have already seen except now it has been changed to include JDBC. Recall that when we register, we actually create a new table. Because at the point of registration, we exactly know how many friends are there in the group of roommates. So, the table command, CREATE TABLE. The name of the table is events for us. And the various fields of the table, event number being an **int**.

Note that this is a partial string, partial command, although it is a complete string it is a partial command and as we go process the registration part we are going to add to this string and therefore at the end complete this particular command. So, as a part of doing registrations, we are building the command piece by piece that is going to construct the table. So, as we have found a number of roommates for every roommate, you are going to add expenses for that roommate, which is represented using a double precision 10-digit 2 decimal places number.

This is in a loop. So, as many friends that have been as many friends has, as many has been requested, that long table will be created, but this is just the expense for loop, you will have one more for loop, which does the per head share, which creates the per head share field. These data structures are as before and finally the table command is completed, the entire construction of the table command is complete.

Once this is done, you can now use the execute my SQL command method of the FSDB object which is an instance of the example JDBC class. Here is how you would execute my SQL command. We will ignore the query more for a moment, but here is the string, this – the string – the command is expecting an entire SQL command “Create Table” with whatever even number as integer or expenses for fl as double precision or 10 digit, 2 decimal places and so on so forth.

These messages are to trace what is happening. The way command is executed is you use the create statement method to create an SQL statement and within that SQL statement you execute the command (the string, whatever is it). In this case the command is to create a table. That is

what is happening at this point, it is a new table command that has been there. Now that is being executed.

So, we see that the basic interaction with MySQL database via JDBC is very simple. You connect to a database, start executing the individual SQL commands. How do you execute individual commands? While processing the data, construct the SQL command piece by piece until the entire command is complete. Once the entire command is complete use the JDBC interface to execute it.

This is simply a method that we wrote. But these are the methods to create a statement and execute it either as a query or as a command itself. The same thing will be done for, say expenses. In an expense command, we need to insert the record of expenses. That record is fortunately available the maker called method that we had written in our very first program. A slight modification of that would involve adding an extra variable that helps us to construct the insertion of that event into the database.

So, insert into the table values that have been found or processed for that particular event. So, to the event insert command, we will simply concatenate the individual pieces of information. There you go. And finally, we will execute that insertion command using our methods before. We can do this for even report generation. Let us see that command in action. We will first see these 2 files the Java source files as a part of our simple listing of files to convince ourselves that indeed these files exist.

```
amv@amv-desktop: ~/source/LaTeX/Computer-Science/...
$ ls -l fairShare-jdbc-http-interactive-command-line.java exampleJDBC.java
-rw-r--r-- 1 root root 3995 Mar 3 09:43 exampleJDBC.java
-rw-r--r-- 1 amv amv 20219 Mar 3 08:44 fairShare-jdbc-http-interactive-command-line.java
$ javac fairShare-jdbc-http-interactive-command-line.java exampleJDBC.java
$ ls -l fairShareJdbc*.class example*.class
-rw-r--r-- 1 amv amv 3896 Mar 3 11:22 exampleJDBC.class
-rw-r--r-- 1 amv amv 12502 Mar 3 11:22 fairShareJdbcHtmlHttpInteractive.class
$ java fairShareJdbcHtmlHttpInteractive < yy.in
$ cat yy.in
command=register+f1+f2%00%Aexpense+f2++200%00%0Areport++f1%00%0Aend
$ java fairShareJdbcHtmlHttpInteractive < yy.in
Loading class 'com.mysql.jdbc.Driver'. This is deprecated. The new driver class is 'com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
Success: Connected to database at: jdbc:mysql://localhost:3306/fairShareV1 by user "nptel" with password "nptel"

INFO: CREATE TABLE events (event_num int, exp_f1 double (10, 2), exp_f2 double (10, 2), phs_f1 double (10, 2), phs_f2 double (10, 2));
executeSqlCommand(): Executing MySql statement ...CREATE TABLE events (event_num int, exp_f1 double (10, 2), exp_f2 double (10, 2), phs_f1 double (10, 2), phs_f2 double (10, 2))
SQL Exception: Table 'events' already exists
INFO: INSERT INTO events VALUES (0, 0.00, 200.00, -100.00, 100.00)
executeSqlCommand(): Executing MySql statement ...INSERT INTO events VALUES (0, 0.00, 200.00, -100.00, 100.00)
INFO: SELECT phs_f1 FROM events
executeSqlCommand(): Executing MySql statement ...SELECT phs_f1 FROM events
$
```

We will then use the Java compiler to compile them. We can compile both of them in one shot and we will get individual class files. We will see that these class files indeed exist. We will run the fair share program via the Java interpreter, but wait, we need to see the input! So, for the moment, we will comment it and simply look at the input which is very much what is expected what is obtained by our command as a result of HTML interaction as a result of an HTTP interaction, the so-called post interaction that is what we see here.

And then of course, when we run Java when we run the program, we will see the result of interacting with the MySQL database. This illustrator program is by design incomplete, it only gets us started it for example does not show how do you update a certain field of a given event, which of course you would need in order to say update the various the per head share of individual friend if need be, or how do you get the values of a friend.

The per head share of a friend and sum them up so that you can report for that friend. Those details are not included in this particular example, illustrative example, therefore you to work out. **(Video Ends: 38:20)**

That is all that we have in this session. In the next sessions, we will go on to servlets. In the meantime, please extend your initial Java program, so that it can do interact with my SQL database. Please set up my SQL and JDBC, so that these things work.

Thank you. See you in the next session.