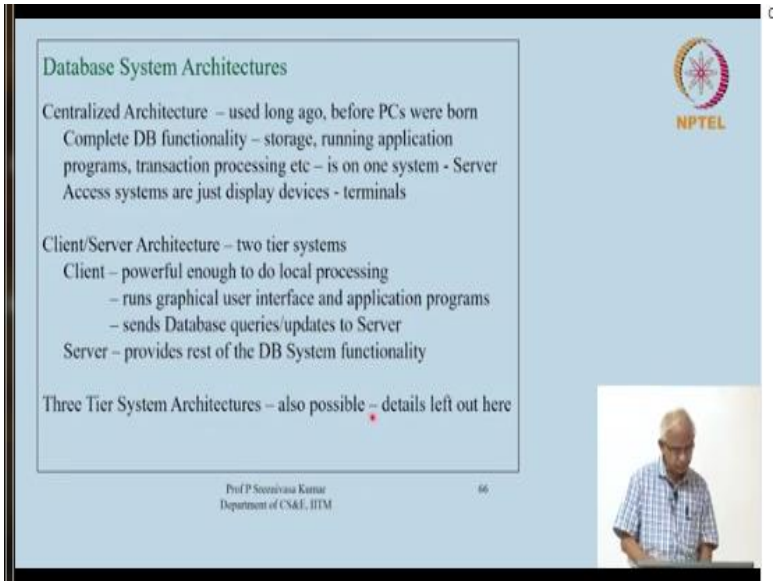
**Database Systems**
**Dr. Sreenivasa Kumar**
**Department of Computer Science and Engineering**
**Indian Institute of Technology-Madras**

**Lecture-18**
**Programmatic access of SQL**

So in today's lecture I want to give you a overview of this application development process.

**(Refer Slide Time: 00:29)**




So we have seen that applications play an important role in the whole setup. So what are the various approaches that are available for developing the applications we will see in that. Before we do that let us see have a discuss the overall system architecture how is the whole thing architecture. The database and the other machines that interact with this database. So one of the first things that were tried out was long ago, before even the PCs were born, there used to be mainframe systems and then the centralized architecture was practiced.

In the sense that all the complete DB functionality, the storage, running of the applications, transaction processing, concurrency control, all of that is on one system with the central server. And then the whatever the access system are used to be dumb machines. So they do not have much processing capability, they were just like terminals, right, so that is the initial architecture. But nowadays we go for multi tier kind of architectures.

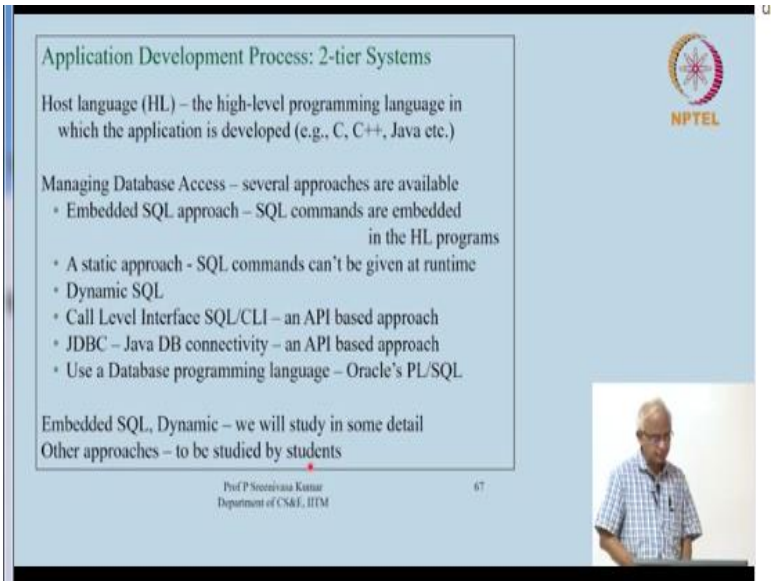
So the basic, you know client server architecture is, what is we will use in this set of slides. So we will talk about the 2 tier kind of systems, where the client is you know reasonably powerful and then it can do some local processing. So typically PCs and workstations connect to databases. So they can do since they can do some local processing, it is possible for us to you know run the user interface and even the application programs can be run on this systems, so the client systems.

And then they will be sending the database access request alone to the database server. So the database server is the one that holds all the data authenticates the users who are supposed to use the data and then grants permissions to use and also keeps track of all the modifications all the complete functionality of databases like this taking care of the storage, transaction, processing and concurrency control, recovery all that is in the server ok.

So this is one kind of popular architecture you would also see actually this is the basic kind of you know client server architecture. It is also possible for us to have 3 tier architectures where the kind of the middle layer actually has and these applications are actually moved to the middle layer. So the middle layer is called an application servers layer or even actively web server layer.

So the clients will do only will have only the user interfaces like they collect the parameters from the end users and then invoke applications themselves run in the middle layer and then the database has run in the last layer. So the database servers will be in the last layer, so that kind of multi tier architectures also are possible.

**(Refer Slide Time: 04:31)**




Now the application we will for illustrating the principles we will just take 2 tier systems into consideration. So we will call this whatever the language that in which the application is being developed as a host language. So it can be t typical like C, C++ or Java. And then we will discuss what are the various approaches that are there for managing this database access ok. Several approaches are available broadly they are like this embedded SQL approach, where SQL commands are embedded in the host language programs.

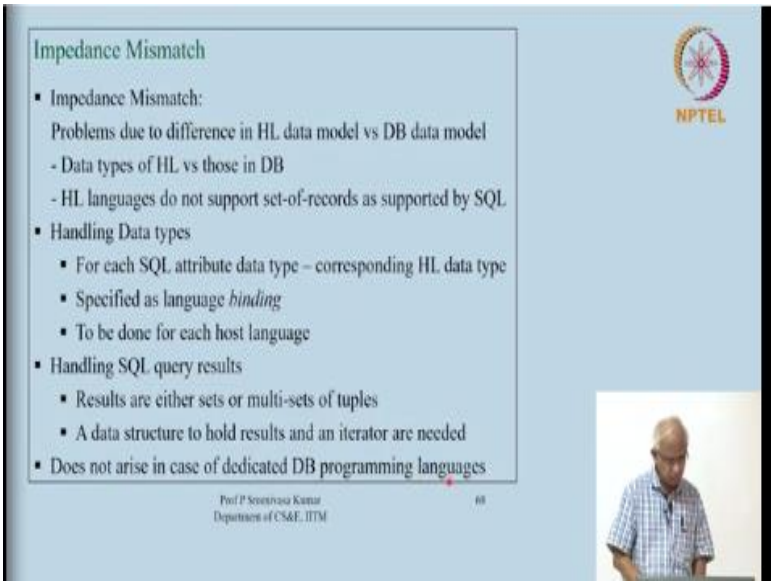
This is some sense a little static kind of approach we will see why it is called a static approach. And there are actually both are possible something called dynamic SQL also we will discuss within the same embedded SQL approach. Then we have there is something called call level interface SQL CLI call level interface, this is part of the SQL standard. So this is a complete in API based approach application program interface. So there is a bunch of library programs, library function calls that are made available by the RDBMS vendors right.

And that will constitute the application program interface. So using these API calls, it is possible for us to interact with the database. So that is one approach and we will see why that approach is some details about that approach. And then for the same kind of approach, if we are doing it through Java programming for Java, then we have what is called JDBC it is an approach where we have an API through which connections to the database servers can be established and then we can work with the database servers.

So to illustrate the principles I will focus on some of these things. So the third approach where you can even have a dedicated database programming language a dedicated programming language itself can be used. So the example for this is Oracle's PL SQL is a programming language slash SQL they call it PL SQL. So this is their own proprietary language which has been developed it is a general purpose language developed with the purpose of developing database applications ok.

So this embedded SQL and dynamic SQL we will go into a little bit detail in some detail other approach I will leave it to you to kind of study yourselves ok.

**(Refer Slide Time: 08:12)**




There is one term that is used in this context called impedance mismatch. So this one actually the term itself comes from electronics kind of domain because impedance is normally used in electronics, circuits and things like that. But anyway it basically, what it talks about is it illustrates the problems that might arise due to the difference between the data model that is there in the host languages, typical programming languages versus the database data models.

Of course, you can see the differences are in kind of 2 forms first thing is the data types of the host language and then the data types provided as part of SQL can be different right. So you can

mostly of course, it is they are kind of you know one can map the various data types that are available in SQL to kind of nearest you know data types in the host languages.

Another major thing is that the host languages you know typically cannot deal with a bunch of data records at a time. So this is one major difference, see SQL when you give an SQL query to the database server. It gives you a bunch of tuples, a collection of tuples, sometimes even it is a set or sometimes it is a multiset right. So a whole bunch of tuples will be written t. Now, whereas in a typical programming language scenario you have a if you are opening a file of records, you will open one by one and then you know operate with them alright.

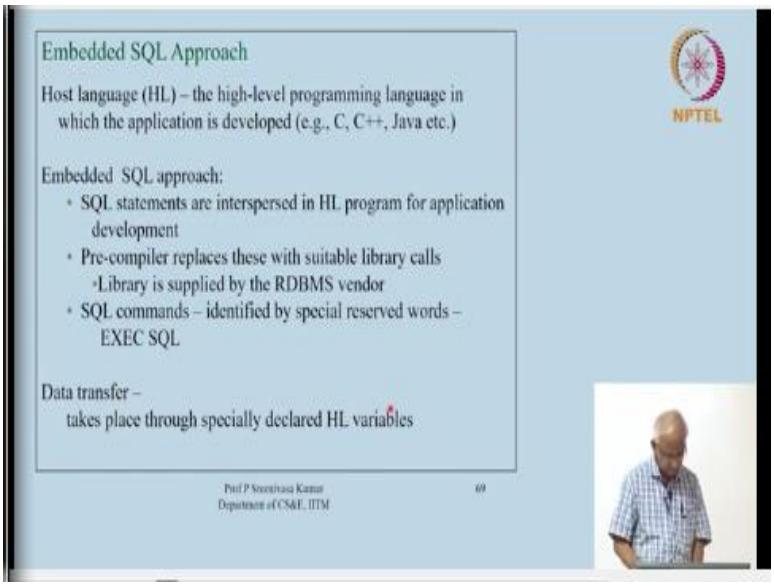
So you do not have a means of handling a bunch of records at a time, so we need to handle this. So for handling this data type mismatches basically for each of these SQL attribute you know data types that are possible in the standard, what are the corresponding you know high level language data types will be specified as part of some kind of a binding. It is called a language binding and this language binding has to be done for all the host languages it is more like an indication thing saying that this is the nearest the host language data type that we can make use of.

So and in order to handle this query results issue, we need to kind of have some kind of a data structure and then provide an iterator to go through those data structures. The data structure could be as simple as an array of you know records and then you have to provide some mechanism to examine the rows of that particular array. So that is how we will handle this entire issue of impedance mismatch between the data model of relational databases versus the data model that is available in the typical imperative programming languages ok.

So with this kind of background, we will see how ok, so one final point here is that this kind of a impedance mismatch will not arise if you are using a dedicated database programming language. Because this would have been in some sense taken care at the designs of the programming language itself, the programming language will have constructs to handle bunch of records coming from SQL queries and things like that.

So this impedance mismatch does not arise in the case of dedicated database programming languages. But there is some downsides to using dedicated DB programming languages.

**(Refer Slide Time: 12:47)**



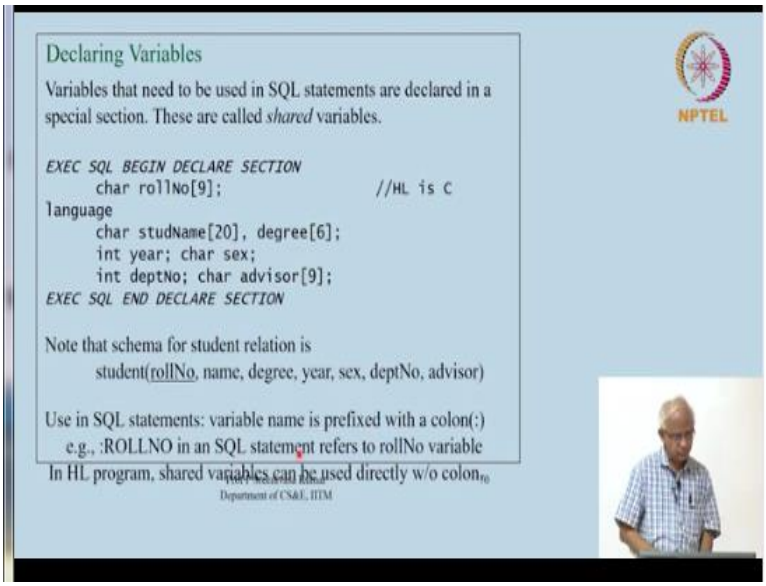

Now let us look at one of the simple and easier to use approach which is this embedded SQL. So in this embedded SQL will gagging assume that there is a C, C++ or Java as one of the imperative languages as the host language high level language, we will call it HL. What we do here is to embed SQL commands statements interspersed in the program itself. Then of course, if you put SQL commands inside C programs obviously the C compiler will grip and then it will throw it out right.

So we need to have a pre compiler, so you know in order to distinguish this SQL commands that I have been put inside in the C program. So what the pre compiler, so you have to give some indication as to where SQL commands start and things like that. So we will use some reserved words like this Exec SQL in order to indicate, where SQL commands are coming in C programs. And a pre compiler will replace all these segments, where SQL commands have been put by an appropriate library function calls to the functions that are provided by the RDBMS vendors. So if you are using a DB 2 database provide by IBM vendor.

Then you will typically have the runtime I mean the library package that has been developed in order to handle this embedded SQL by the vendor and that will be included in your it has to be

included in your C program. And then the pre compiler will replace all these segments, where SQL commands have been used by appropriate function calls to these functions ok. So that is how it will work and the kind of data transfer from the database to your program will happen through specially declared host language variables ok. So we will see how exactly this host will declare this things ok.

**(Refer Slide Time: 15:36)**



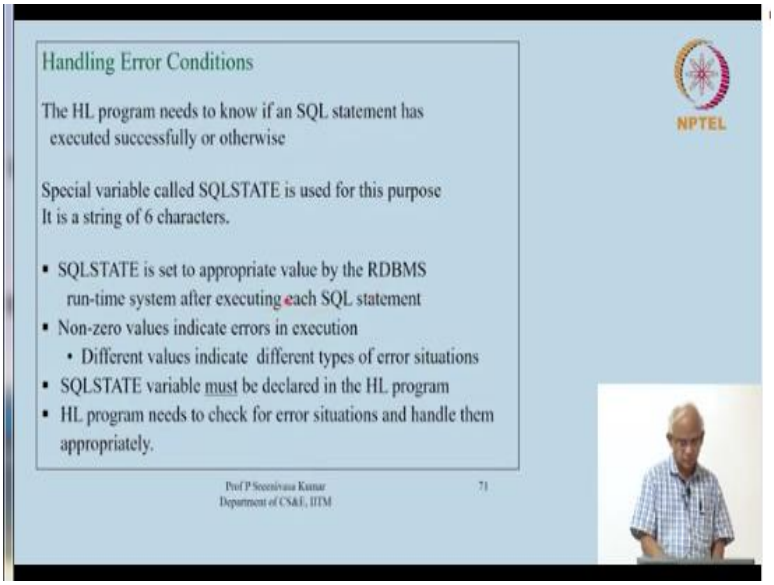

So you need a bunch of variables, so in order to be used in inside SQL commands, SQL statements and so these will be declared as part of a special section. So and these variables are called shared variables ok. So let us look at this set of things here is ok this is a comment, so this language it should actually be here. So this is EXEC SQL begin declare section. So this is where you know all the declarations have to be given and you can see that these are the typical variable and arrays that are declared inside a C program.

And these variables are available for your rest of the C program. You can you know assign values to them you can make use of those values and all that but they can also be used inside SQL commands. And if you use them inside the SQL commands, actually we should distinguish them and then we will do that by actually putting a colon before these variables. Of course notice that these variables have been declared keeping in mind this schema of the student relation that we have. So, all these variables will correspond to the attributes that are there in the students schema.

So when we use them in SQL statements these variables names you know will be prefixed with a colon. So colon roll number in an SQL command a statement will basically refer to this particular roll number variable. And of course in your C program, of course you will use this shared variables as the r right without any colon and things like that ok. So this is how you know shared variables can be used.

In order to basically pass data from your environment, where your C program is running to the database environment basically, so through these SQL commands. So you will initiate these things in your program and then use them inside an SQL embedded SQL command and then those values will be passed on to the database server that is how the whole mechanism will work ok. So let us look at some issues here.

**(Refer Slide Time: 18:43)**




Now, one other thing that you have to be worried about is the occurrence of errors ok. So what if you have given an SQL command, but then you know there is some issue in the data inside the database and what you wanted to execute actually does not is not feasible for some reason. So for this we have something called SQLSTATE which is a special kind of a variable through which the database server will indicate as to what happened when you gave this SQL command to that server.

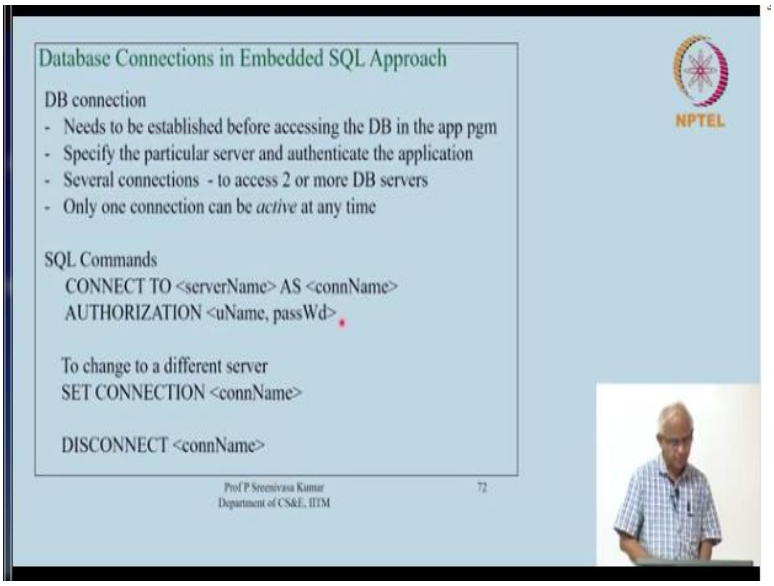
So if everything that is you know if the command is successfully run then it will be set to 0. Otherwise some non 0 value will be set and that non 0 value will actually indicate to you as to what kind of an error has happened during the execution of the command. It says this is a string of 6 characters and you need to actually declare this. So SQL state is going to be set to a some appropriate value by the RDBMS run time system ok.

After executing each of these comments, so when your program is running you have issued an SQL command. And it goes to the database server the database tries to run it and there is some issue and then it will set this SQLSTATE. So this is part of the shared variables, so it will set this value to appropriate thing. And so in your program after issuing each of these SQL commands, in the immediate next statement you must check what is the value of the SQLSTATE it is everything fine or is something wrong.

If something is wrong then you should program the appropriate sequence fractions based on that. So non 0 values indicate errors in execution and it is part of the standard as to what those non 0 values stand for and things like that you must refer to the standard to figure out all that. And then based on that you should put some series of cases and things like that and then handle the situation.

So this SQL state is a special variable and it has to be declared in the declared section. I did not show it in the previous slide, but it has to be there in that as an array of 6 characters as a string of 6 characters ok. So this is how communication between the your program and the database server happens. So database server indicates things to you through this SQLSTATE.

**(Refer Slide Time: 21:48)**




Now before you start working with the database it is required that you set up a connection to the database it is required that set up a connection. So while doing that, you will be specifying as to what is the particular server which host this database. And then and your credentials you know you have to authenticate the application has to authenticate itself to the database server.

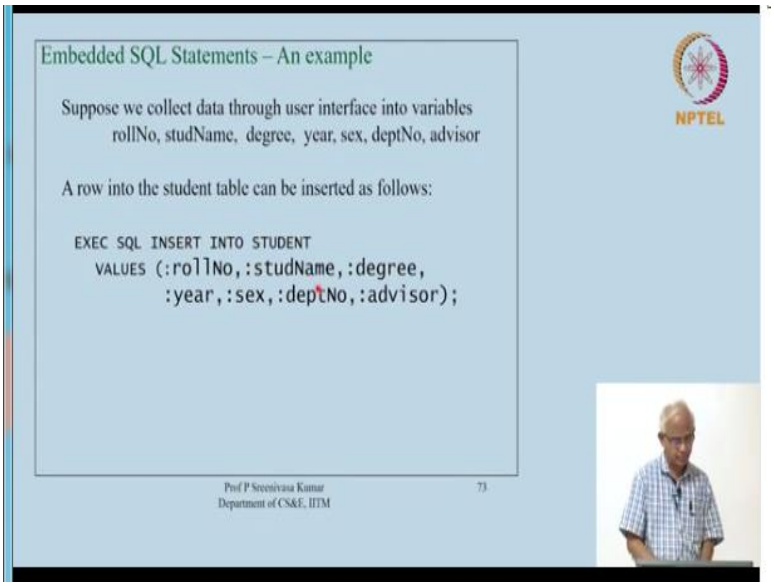
So it has to give the user name password and things like that and typically, it is possible for the application program to kind of work with multiple database servers at a time. But at any point of time, you know only one connection can be active as far as embedded SQL is concerned. At any point of time only one connection can be active that means you are connected to one database server at that particular point of time.

Of course you can once you are done with that particular database you can close that connection and then reconnect to another database, so that is possible. So these are the SQL commands that will handle these connections. So connect to what is the server name and then there will be a connection name that is going to be given for this particular connection. And then immediately you will have to follow it up with authorization in which you give user name passwords and things, so you authenticate yourself.

And in order to kind of changed a different server you would basically use this command to set connection to a new connection. Of course you can disconnect you have to disconnect from one

server in order to before you can connect to another server ok. So these are all part of developing the application program, so before you do any actions you basically set up a connection.

**(Refer Slide Time: 24: 19)**



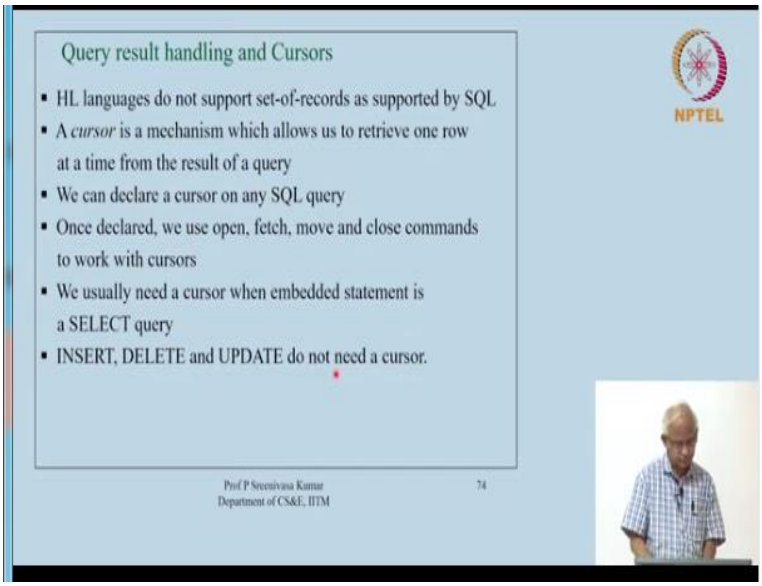

Let us look at some examples of typical SQL statements and then we will look at how to handle this query results and all that. Let us say suppose we collect the data through some graphical user interface into these variables. So say roll numbers, student name, degree, year and all that, so we have actually collected some data from the end user, now this has to be basically inserted into the database.

So you can issue a command like this, so EXEC SQL insert, so this is the standard SQL command to insert tuples into database. Now we do not have a tuple of values, but instead those values are in these host language variables right. The host language variables we have set up those variables have been initiated for with these values. And so that is why you would give a typical command like this insert into student table values.

So instead of giving directly values, you will give the variables that hold those values and you notice that these variables are all having colon before that. Because they are the shared variables that are being referred in the SQL commands. And then you will so this command would go to the database server and into try to insert it and if you have not created the student table before.

Then it will crib obviously and then you know you will have to check this SQL state to figure out what has happened after that.

**(Refer Slide Time: 26:12)**



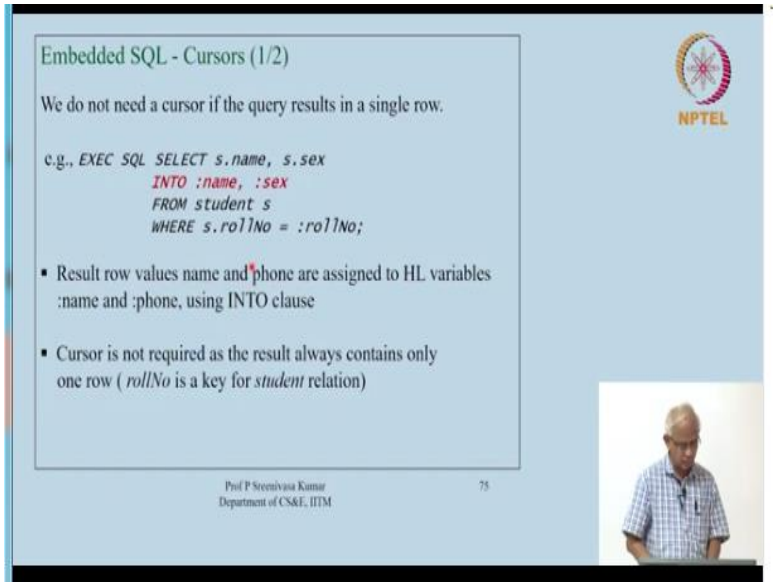

Now let us look at handling query results and in this context we will bring in a term called cursor into the picture. Though you will be able to handle a set of records in the sense of you know opening a file of records and then operating with that in a host language. We need some kind of a data structure to kind of you know receive this results from the database server that is the whole issue about this mismatch.

So what is a cursor is basically is a mechanism which allows us to retrieve 1 row at a time 1 tuple at a time from this result of a query. A cursor is required when you expect multiple you know tuples being written by the thing. If you very well know that you know 1 tuple is going to come, then you actually do not need a cursor ok. So for any SQL query, we can declare a cursor, so as part of the embedded SQL commands, we can declare a cursor for a particular SQL query.

So once you declare it, it will have a name a cursor will have a name and then you can use that name to kind of open that cursor. When you open the cursor the SQL query will be sent to the database server and then will be executed the results will be available in some time kind of a temporary storage. And then you can fetch those tuples and then move the cursor etc, and then finally close it.

So when usually we will need this when the embedded statement is in a select query that typically returns multiple tuples. Of course, the insert, delete, update all these things they do not need a cursor because they are dealing with. Even if they deal with multiple rows, your input is unique for each of these things, right. So you do not have to deal with results from the database server ok.

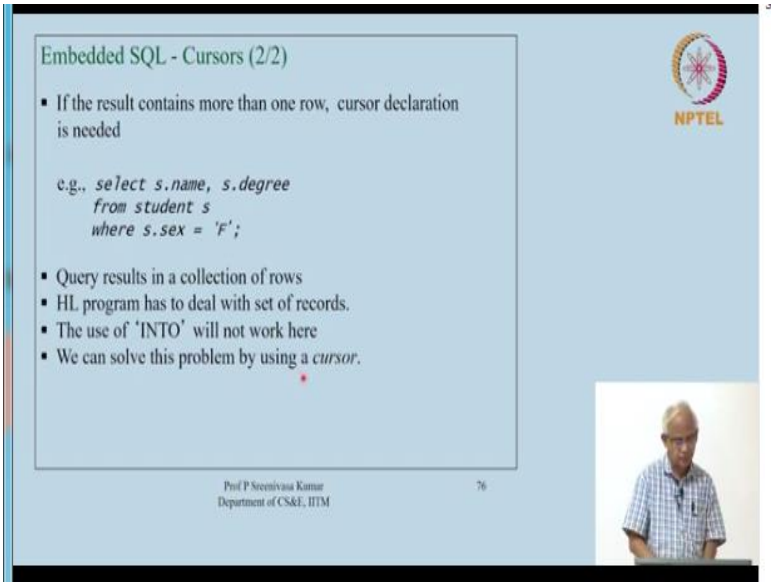
**(Refer Slide Time: 29:10)**




So here is a case where we do not need a cursor, so here is select s name s dot sex into look at this new clause. So you can now give variables in your program which will receive these values into and then you from student s where s dot roll number equals. So you already have initiated the roll number variable with some specific roll number. And now you want to give this command saying that get me the name and sex values of this particular student.

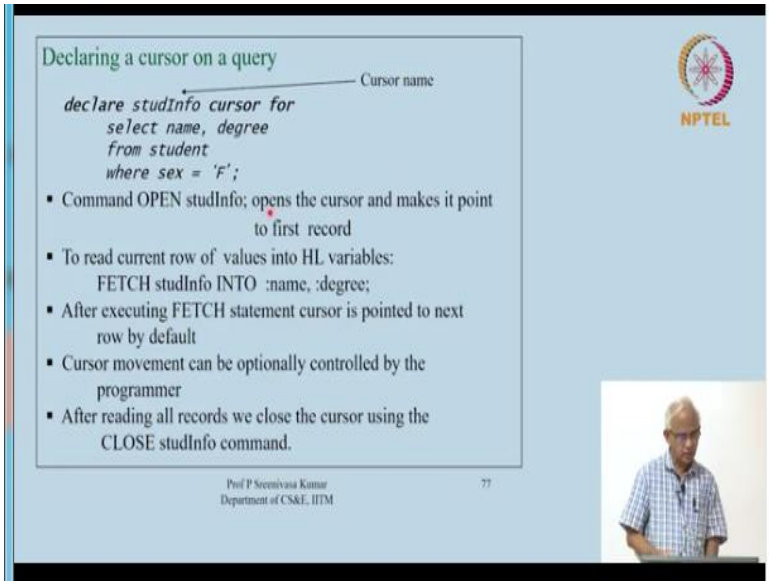
So from student s s dot roll number equals particular roll number, so the values. So since roll number is a key, you know that you will only get 1 set of values. And so there is really no need for setting up any cursor for in this particular case even though it is a query. Whereas typically this is not the case in general and you would required to declare what are called cursors.

**(Refer Slide Time: 30:30)**




So look at this query select s dot name s dot degree from student x where is student sex is F. So all name and degree details of girls students you are getting. So obviously there are multiple rows and so you cannot use this into approach because it is a problem, so here is how you would handle that.

**(Refer Slide Time: 31:02)**




So here is a command, where you are declaring the cursor for a particular query. So it is called declare student info is the cursor name. And this is again a keyword cursor for and then you give the query. So once you declare this, it is there you know as a definition alone, only when you open it, if you give an open command, open the student info cursor then the command actually goes to the database server.

And then you get a bunch of tuples returned by the database server. And then they will be you know in some kind of a temporary storage it is all handled by the by the embedded SQL approach now. So the pre compiler will take these things and then appropriately create library you know function calls to handle this kind of situation now, it will create some temporary storage and things like that.

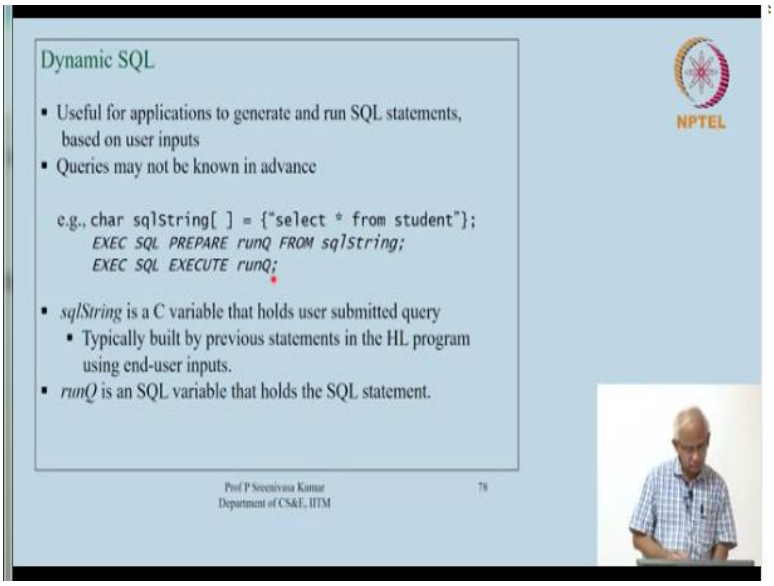
And then get ready to handle various commands related to cursors ok. Now to read the current row of the values into the HL, the host language variables, you basically use this command FETCH and then give the cursor name INTO. And then a sequence of variable names as many as there are columns in that particular. Now after this FETCH then the cursor is pointed to the next row by default.

But of course you can actually control as to whether the cursor should move to the next one or should move to the previous one or should actually jump to the fifth one next or etc. So all those things I have, they are there as part of the cursor mechanism, but I have skipped mentioned them in the slides. But there are this command, the FETCH command is actually much more complex than what I have shown you here.

It can have optionally other parts where you know you can say that ok after doing this you jump to the fifth row or something if you, really needed like that. So you can control the cursor as to where it goes, it can be optionally controlled by the programmer. Then after reading all the records that have come, you can actually close the student command that particular cursor. So this is how you kind of you know handle this situation that the database server typically returns set or multi set of tuples.

And then you have to make use of those values in your host language program. Maybe you will just simply display them or print them or something like that but then you need to get hold of them or maybe you are actually examining each of them and then trying to figure out the next actions to be taken extra ok. So this is how embedded SQL typically works.

**(Refer Slide Time: 35:14)**




And then let us move on to something called dynamic SQL where ok, so if you are using embedded SQL then you know the SQL query or command to be given at the time of writing the program application program you know what exactly is the command to be given. You may not know the exact parameters ok, what role number to give and things like that. You may not know that you will pick up from the host language variables which are initialized through some other function.

And then but you know the command right that is what you will embed here ok. So what if you do not know the kind of query that you need to post to the database server, what if that query has to be figured out at runtime. So this is one major issue and in order to handle that, one simple approach that is as part of the embedded SQL is to do this. So you have a so it will give you commands like prepare and then execute ok.

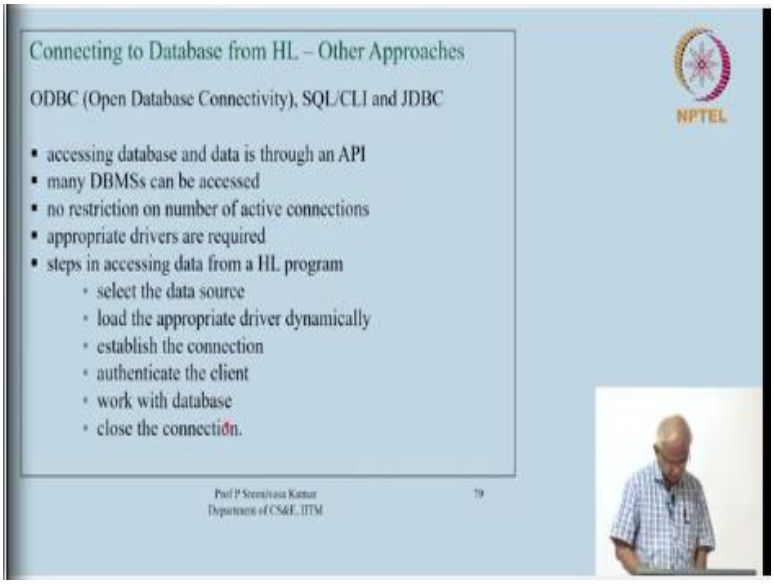
So I will show you so basically, what prepare does is to take a string we call it SQL string here, but you can call it anything. Take a string and then treat it as a query ok, so it is a character string. And this prepare from will basically prepare some query called runQ from this string ok. And then if you EXEC execute that thing then that particular query will actually be executed and then you will start getting the results and then you can use cursors and things like that.

Now how do you, so this particular string is actually a character string that is declared in your program. And in this case of course I will simply you know instantiated with a constant string like this here. But it this may need not be the case, what typically does do I mean you will check some various conditions and then probably interact with the end user and then depending on the input that you get from the end user. You decide as to what are the tables from which the information is needed to be you know obtained and then what are the conditions to be.

So you will actually dynamically form the query and then you know putting characters into this SQL string to reflect that query got point. So you will at run time after interacting with the end user, decide as to what is the query to be fired to the database server and formulate that query into a character string. And then use this prepare, runQ from SQL query from that string and then say execute.

So why it is split into 2 prepare and execute is basically that you may want to run that query multiple times after having prepared it ok. So that is why this separation, so this is very briefly about this, you know dynamic formation of SQL queries which is actually very much required. Because this makes your program highly you know flexible. In sense it can handle the end user requests various different kinds of end user requests and then has the ability to dynamically form the query and then send it to the database server ok.

**(Refer Slide Time: 40:00)**



Now I will briefly touch upon the other approaches here which are the APA based approaches. So before SQL CLI was there then there was this ODBC open database connectivity, again function set of function calls. And then JDBC is also similar kind of thing which is mentioned, it is to be used for application programs being developed in Java. So as against the embedded SQL approach the approach here is to access the database through this API function calls.

And who provides this APIs obviously the database when it is actually part of the standard SQL CLI and JDBC ODBC, these are all standards. So there are such standard function calls that have function calls have been standard as and how you implement it will depend on your database system. So each database management system vendor will implement exactly these function calls and then give you a database DBMS when a JDBC driver for his or her for their system right.

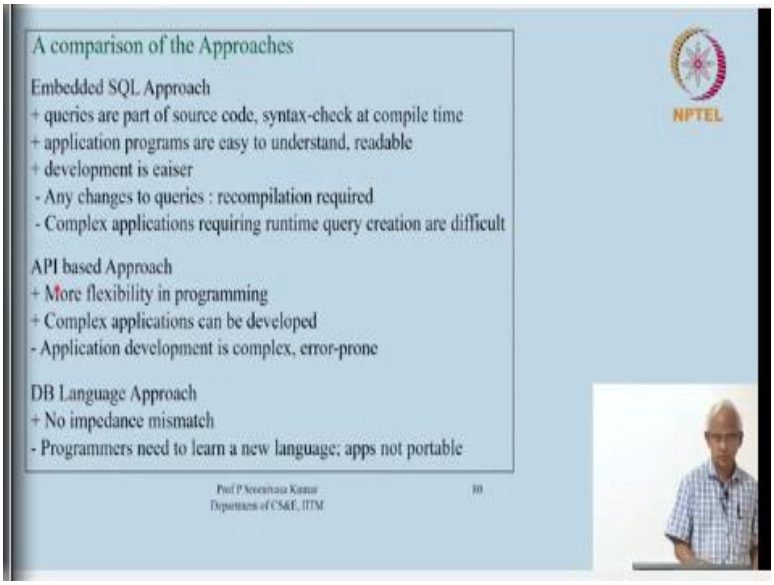
So the advantage of this using this approach is that it is pretty flexible you can you know, you can dynamically prepare SQL queries and things like that. And also many databases can be simultaneously accessed and there is no restriction on how many number of active connections you can have with databases. And then of course in order to use them, you will require appropriate these drivers.

These drivers are the ones that are provided by the RDBMS vendors who will implement these standardized you know these SQL CLI and JDBC they give you the templates for function calls and those are implemented by the database. Since they are standardized, so you can actually use your program in order to kind of access either a oracle database or a DB to a database or any other database as long as they provide you this JDBC or ODBC connectivity they ok.

So typically how it goes is, so at runtime you select your data source and then load the appropriate driver dynamically ok. So you are so you figure out that it is a you are interacting with an oracle database. And then you will load the oracles a JDBC let us say you are writing a Java program a JDBC driver. And then that will give, so I am of course not giving details about this but it has now various function calls in which now you can establish connections you can manage connections and all that.

So using those function calls you basically establish connections, authenticate yourself, authenticate the program and basically then pipe the SQL commands that you want to actually. You may either have them statically with you or you may dynamically prepare this SQL commands and things like that. So you will pipe the SQL commands to them and then the close connections. So this is a typical this is kind of required and this is how you work with these things.

**(Refer Slide Time: 43:59)**




So as a comparison between all these approaches, well in the embedded SQL approaches, you know the queries are part of the source code. So actually syntax checking at the compile time is possible right. And application programs are actually easy to understand because the SQL commands are also there as part of this as the thing there and the development is somewhat easier.

But then any changes to these queries you know you will have to recompile the whole application, it has to go through this compilation cycle, generating the executable and all that. So complex applications requiring runtime you know query creation are difficult to or not possible actually using limited SQL. So one has to go for APA based approach for that.

So APA based approach has flexibility but then the applications and complex applications dealing with multiple database servers can be developed ok. And typically this kind of thing is

required in practice supposing you are handling let us say accommodation requests for you know people coming in for some attending some event like a conferences or something like that.

And then you gave choice for people to you know choose 2, 3 accommodation options. And then you want to reserve some room for them as per their request say, and so each of these accommodation places are independent. So they will have their own database server to give you a confirmation as they saying that yes, we have room and then we will reserve this room for them.

So in your application program you would be dynamically figuring out as to what is the you know hotel or accommodation place that a particular end user wants. And then you will try to establish a connection to that hotel database server. And that may be running in oracle RDB 2 you do not know. And then you will you know make the reservation get the confirmation and things like that then go and do the confirmation to the end user.

So in a typical scenario practical scenarios you will need to handle connections with multiple databases and you need to dynamically prepare queries. So application development process is complex and also error prone one has to check these application programs thoroughly before you launch. Of course the DB programming language option is always there. But then you know of course there is no this impedance mismatch problems and things like that do not arise because it is specifically developed for database programming.

But then the downside of this is the programmers need to learn a brand new language ok, if you already know C and C++ then you cannot go comfortably you know go on developing database applications. But you need to now pick up this new language which maybe similar to existing language but it is still a new language. But another major issue is that you get locked to that programming language your application programs do not are not portable.

That is one major issue, application programs become so these are typically you know provided only by. So whoever provides so you are kind of limited to that database language, you cannot use a general purpose language. So if you use a general purpose language, then you can move

your application to another you know deal with another database server and things like that. So with this these are the various issues, so you have to depending on the need of your application, you have to choose these approaches and then go ahead and then make use of them.

The APA based thing, I obviously did not cover much but because it has a lot more details and one has to pick up these details of the API's ok, so with that ok we are closing SQL.