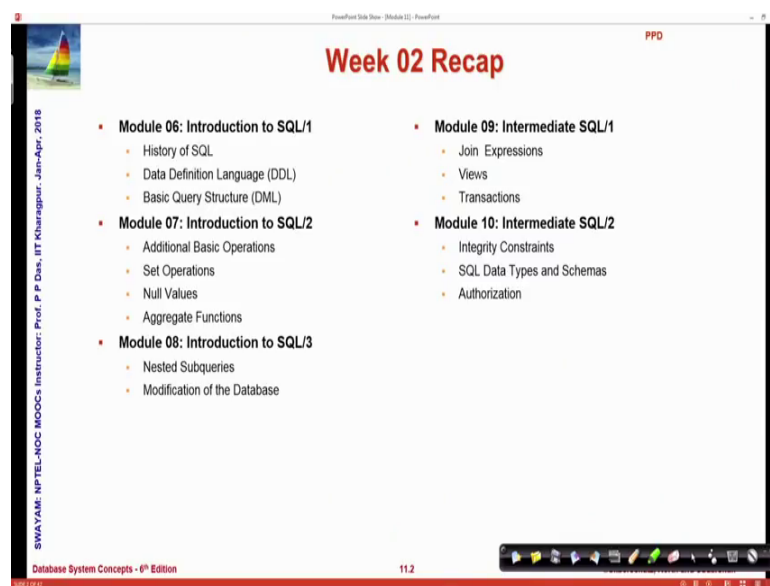


Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 11
Advanced SQL

Welcome to module eleven of database management system. This will be on advanced SQL.

(Refer Slide Time: 00:31)



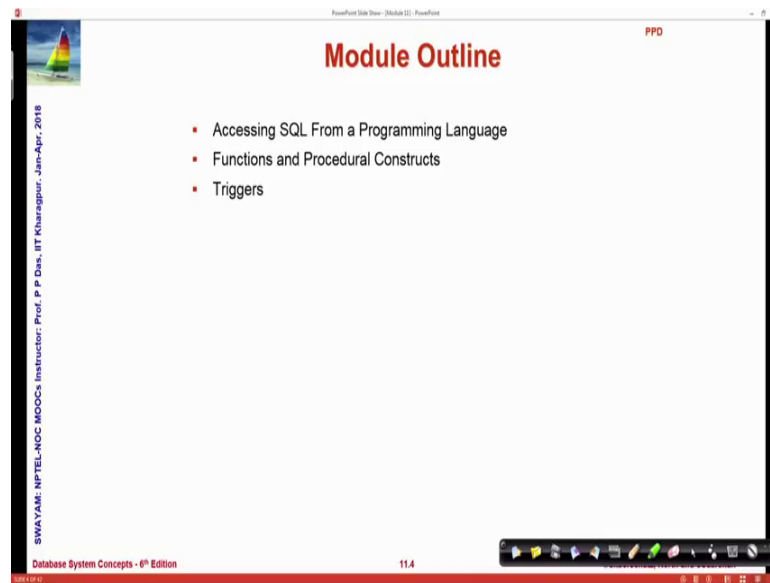
The screenshot shows a presentation slide titled "Week 02 Recap" with a red header. The slide lists the following modules and topics:

- **Module 06: Introduction to SQL1**
 - History of SQL
 - Data Definition Language (DDL)
 - Basic Query Structure (DML)
- **Module 07: Introduction to SQL2**
 - Additional Basic Operations
 - Set Operations
 - Null Values
 - Aggregate Functions
- **Module 08: Introduction to SQL3**
 - Nested Subqueries
 - Modification of the Database
- **Module 09: Intermediate SQL1**
 - Join Expressions
 - Views
 - Transactions
- **Module 10: Intermediate SQL2**
 - Integrity Constraints
 - SQL Data Types and Schemas
 - Authorization

Additional details from the slide: A vertical sidebar on the left contains the text "SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr, 2018". The bottom of the slide shows "Database System Concepts - 6th Edition" and the slide number "11.2".

Before we start let me quickly recap what we did last week in the five modules. Last week we totally spent on discussing first the introductory features of SQL how to create data and how to write basic queries, and we studied about all different kinds of SQL operations at theoretic operation, handling of null values aggregation, nested queries and so on. And then we did an intermediate level of SQL query formation in terms of joint expression, views and integrities different kinds of SQL data types and importantly authorization.

(Refer Slide Time: 01:24)



SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Module Outline

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

Database System Concepts - 6th Edition 11.4

PPD

This slide, titled 'Module Outline', is the 11.4th slide in a presentation. It features a vertical sidebar on the left with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018'. The main content area has a red header 'Module Outline' and a bulleted list of three items: 'Accessing SQL From a Programming Language', 'Functions and Procedural Constructs', and 'Triggers'. The footer includes 'Database System Concepts - 6th Edition' and the slide number '11.4'. A 'PPD' logo is in the top right corner, and a navigation bar is at the bottom.

In the context, in this context, we now take up some more of the SQL features which are somewhat advanced. And we will try to understand how SQL can be used from a programming language, and familiarize with functions and procedures in SQL. This will violate some of the basic premises that we started with in saying that SQL is a declarative language only because as you understand functions procedure as procedural language features we will see how to handle those, and we will take a look into another important feature of Krieger's.

(Refer Slide Time: 02:07)



SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

ACCESSING SQL FROM A PROGRAMMING LANGUAGE

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

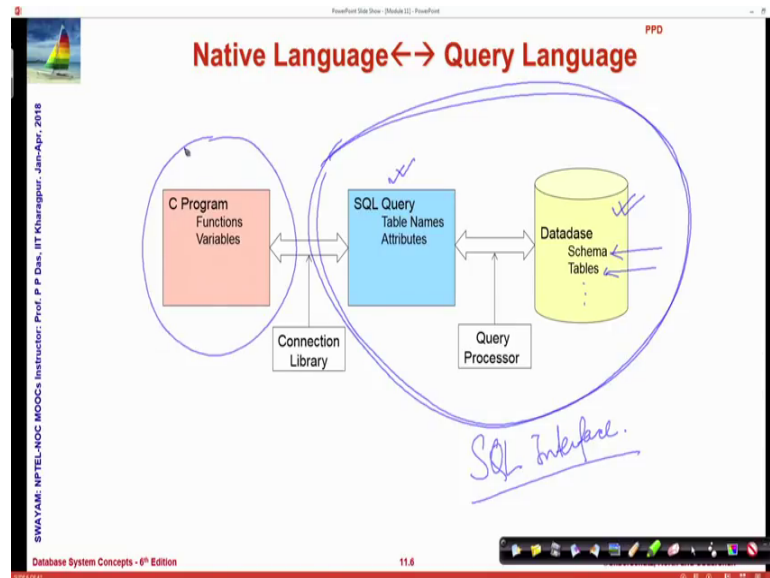
Database System Concepts - 6th Edition 11.5

PPD

This slide, titled 'ACCESSING SQL FROM A PROGRAMMING LANGUAGE', is the 11.5th slide in a presentation. It features a vertical sidebar on the left with the text 'SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018'. The main content area has a red header 'ACCESSING SQL FROM A PROGRAMMING LANGUAGE' and a bulleted list of three items: 'Accessing SQL From a Programming Language', 'Functions and Procedural Constructs', and 'Triggers'. The footer includes 'Database System Concepts - 6th Edition' and the slide number '11.5'. A 'PPD' logo is in the top right corner, and a navigation bar is at the bottom.

First with accessing SQL so, this is the module outline accessing SQL from a programming language.

(Refer Slide Time: 02:11)



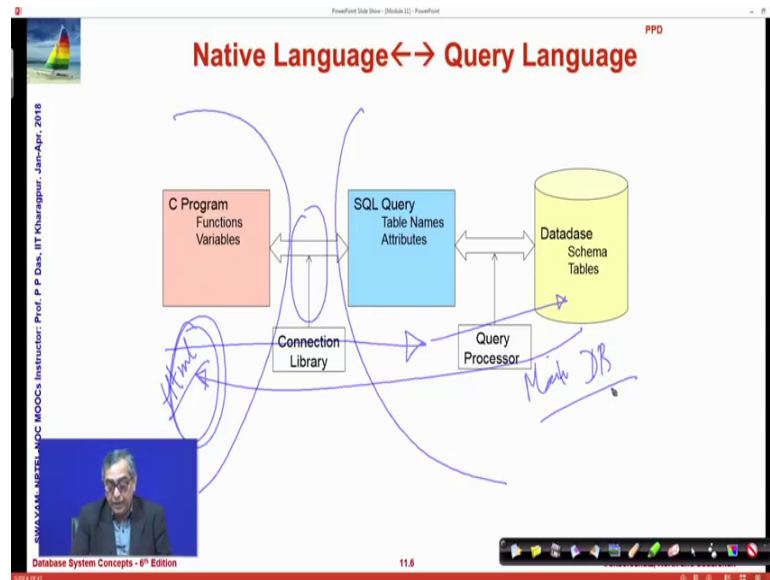
So, this is just kind of an abstract view that you can think of that what we have been doing so far naturally there is a database which has primarily two things schema and tables. Of course, there are many other things there are index, there is authorization that triggers and all that, but there is a database which stores everything.

And so far we have been dealing with only this part of the information that I can write certain SQL query to define table, using table names, attributes and certain logic and that goes through certain query processor works on the database to get me either create the desired effect. Either that is creating table instances or creating index or extracting certain relation values, defining some views and so on. So, all these have to happen through an SQL interface. So, this has to happen through an SQL interface.

So, whatever system we are using whether you are using mySQL, Postgres, Oracle, Cybers SQL server whatever you using that has some interface through which these SQL queries can be executed, so that is kind of a standalone complete system. But, in general, we will have lot of more requirements in terms of the application. For example, the application might require some graphics, SQL does not have support for graphics; application might require certain numerical algorithms to be executed might require some geometric computations to be done poly intersection of polygons to be computed

and so on and so forth. It might require some network programming and so on. So, there is a need to do all these and certainly these are best done in terms of certain native language that could be C, c plus plus, java, c sharp, visual basic, python any of the native languages which has support for a variety of different tasks.

(Refer Slide Time: 04:20)



So, what is critical is can we make a bridge between these two that is can we take the advantages of the SQL domain and the advantages of the normal imperative procedural programming domain together. That is I can do some graphic representation and then certainly go to the database extract some information and then present it in the graphics and vice versa. For example, whenever we access say gmail, we get to see them in terms of an html presentation which is some kind of a graphics rendering, but the all the mail entry certainly come from some database.

So, somewhere there is a connection by which when I say get my inbox mails and somewhere the information goes over from this to the SQL query up to the database and the result is brought back to me. I see that in the html page here, but here there must be certain mail database, where all these information exists. So, what I am trying to come at is it is critical that the application programs or you know high level programming normal programming languages native languages should be able to interface with SQL. And what we discuss here is two different mechanisms for this interfacing. The first one is using a connection library and this is what I will specifically show you.

(Refer Slide Time: 05:50)

The screenshot shows a presentation slide with the following content:

- Accessing SQL From a Programming Language**
- API (application-program interface) for a program to interact with a database server
- Application makes calls to
 - Connect with the database server
 - Send SQL commands to the database server
 - Fetch tuples of result one-by-one into program variables
- Various tools:
 - JDBC (Java Database Connectivity) works with Java
 - ODBC (Open Database Connectivity) works with C, C++, C#, Visual Basic, and Python
 - Other API's such as ADO.NET sit on top of ODBC
 - Embedded SQL

Handwritten blue circles highlight the phrases "Send SQL commands to the database server" and "Fetch tuples of result one-by-one into program variables".

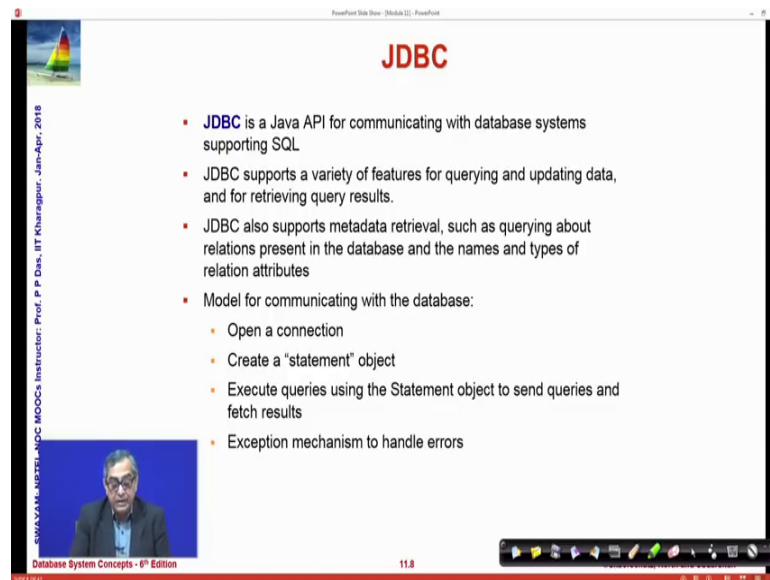
Vertical text on the left side of the slide reads: "MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr, 2018".

At the bottom of the slide, it says "Database System Concepts - 6th Edition" and "11.7".

So, for if you are using a connection library then you have a set of APIs application programming interfaces these are basically functions in that library. So, that with that application can connect to the database because certainly the database is somewhere else is a different server. And send an SQL command to the database server so that you can say that this is what I want. And then a result is computed the result of that computation the table the can be brought back those tuples can be brought back to the application program. Mind you when we are here when you are sending the information in terms of the database server, we are talking SQL command, we are talking about attributes tables of the SQL space.

Whereas, when I want it in the program I want it in terms of program variables. So, there has to be certain correspondence made between them. There are a variety of tools available which allow you to do this JDBC is common very commonly known which is specific for java. We have an open database connectivity APIs which has different versions for different languages these are the common languages that it is used with. And as we will see later on that there is another mechanism for doing the same thing called the embedded SQL, we will come to that later.

(Refer Slide Time: 07:15)



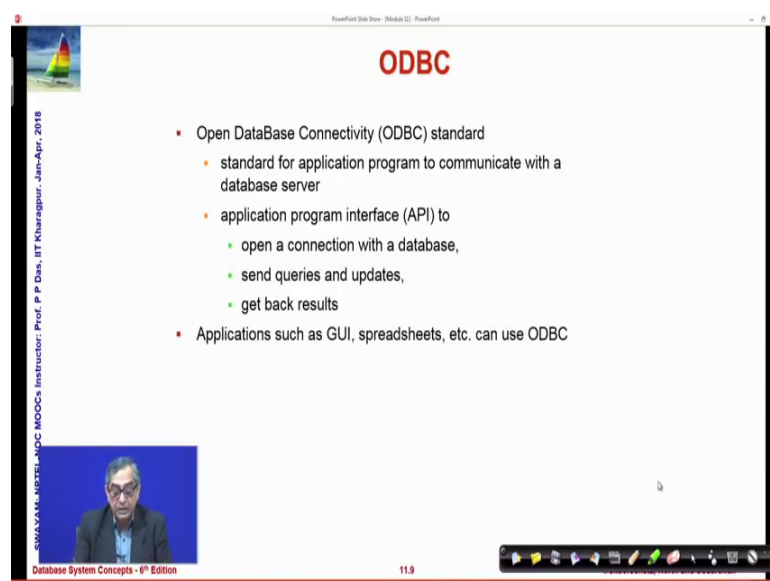
The screenshot shows a presentation slide titled "JDBC" in red text. The slide content is as follows:

- **JDBC** is a Java API for communicating with database systems supporting SQL
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes
- Model for communicating with the database:
 - Open a connection
 - Create a "statement" object
 - Execute queries using the Statement object to send queries and fetch results
 - Exception mechanism to handle errors

On the left side of the slide, there is a vertical text: "©2013 ANIL K. JAIN, IIT Kharagpur, Jan-Apr, 2018". At the bottom left, there is a small video inset of a man speaking. At the bottom right, there is a navigation bar with icons and the number "11.8". The footer of the slide reads "Database System Concepts - 6th Edition".

So, JDBC is a java API, I will not go into details here if you know java you should be able to quickly look up. So, it is a it communicates with the database by opening a connection creating what java calls a statement object and executes the query using the statement objects and that is used to send the query as well as to get back the result. So, java is object oriented as you know so statement object is used as a as an encapsulation which travels between the java program and the SQL query processor. And since the query gets back the result since the query gets back there is that kind of there is exception mechanism to handle errors which is common for java.

(Refer Slide Time: 08:00)



The screenshot shows a presentation slide titled "ODBC" in red text. The slide content is as follows:

- Open DataBase Connectivity (ODBC) standard
 - standard for application program to communicate with a database server
 - application program interface (API) to
 - open a connection with a database,
 - send queries and updates,
 - get back results
- Applications such as GUI, spreadsheets, etc. can use ODBC

On the left side of the slide, there is a vertical text: "©2013 ANIL K. JAIN, IIT Kharagpur, Jan-Apr, 2018". At the bottom left, there is a small video inset of a man speaking. At the bottom right, there is a navigation bar with icons and the number "11.9". The footer of the slide reads "Database System Concepts - 6th Edition".

What you see in contrast does a similar thing, but since it is to cater for different languages it has got a softer model it has got less powerful model. It is a standard application program to communicate with database server. So, again it has to open a connection to the database, send queries and updates and get back results. So, these are these are the three basic things, these three other three basic things that certainly needs to be done if I want to easily work across the application programming domain and the database programming domain. So, applications such as GUI, spreadsheet, etcetera can use ODBC.

(Refer Slide Time: 08:39)

ODBC - Python Example

- The code uses a data source named "SQLS" from the odbc.ini file to connect and issue a query.
- It creates a table, inserts data using literal and parameterized statements and fetches the data

```

import pyodbc

conn = pyodbc.connect('DSN=SQLS;UID=test01;PWD=test01')
cursor=conn.cursor()
cursor.execute("create table rvtest (coll int, col2 float,
col3 varchar(10))")
cursor.execute("insert into rvtest values(1, 10.0,
'ABC')")
cursor.execute("select * from rvtest")

while True:
    row=cursor.fetchone()
    if not row:
        break
    print(row)

cursor.execute("delete from rvtest")
cursor.execute("insert into rvtest values (?, ?, ?)", 2,
20.0, 'XYZ')
cursor.execute("select * from rvtest")

while True:
    row=cursor.fetchone()
    if not row:
        break
    print(row)

```

Source: <https://dzone.com/articles/tutorial-connecting-to-odbc-data-sources-with-pyth>

So, yeah I am just quickly showing you an example we will talk about application programming mode in a in a later module. So, this is a python example. So, python for this the ODBC for python is known as py by ODBC library. So, you need to import that. And then using that you can connect. So, if you have to connect you have to say which database who is the user, what is the password, because authentication needs to happen. So, SQLS is a database here and with this user with this password is connecting you that is successful you get a conn object and on the conn object you have something what is known as a cursor. Cursor is nothing but if you think about it is it is kind of a pointer. So, it can be used to point to either a row or a whole query or a table.

So, you get back a cursor object. So, then this is if you look into this part, this part is nothing but a pure SQL query. So, you take that as a string and pass it on to the execute

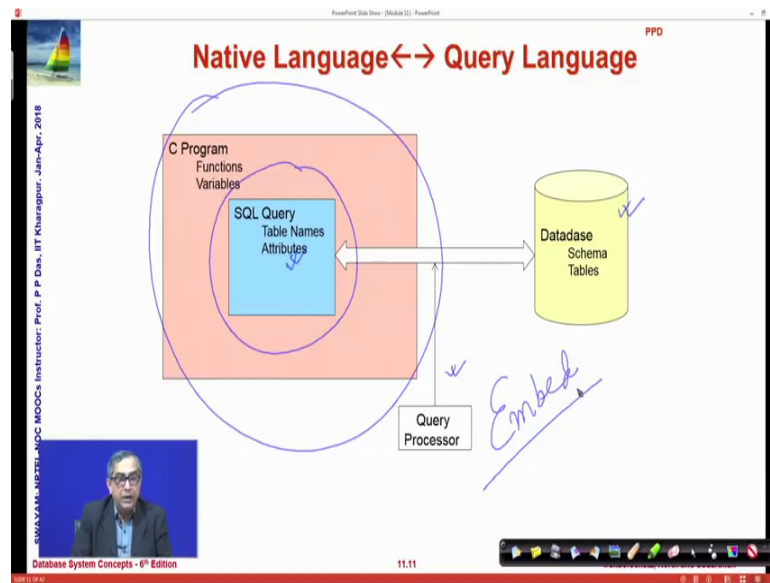
method of cursor. So, what it does is, so this is this has done the first task which is connecting to the database. Now, you are basically putting the query to the database in terms of doing saying that this execute. Similarly, so if that will get executed, so the table is created naturally there is no result to get if you have created a table. Now, you do an insert. So, again this particular record, you can see that within this double quote, this whole insert syntax you take that as a string and just give it to execute as a parameter.

In the third, that you do is do a select. So, you have done this we have inserted a record, created a table inserted a record in that and then now you are doing a select. So, certainly we expect one record to come back. So, these are the SQL executions and then your get back result. So, cursor has another method which is known as fetch one. So, what it will do that if your result table has multiple entries, then the cursor will if will be will start with the first row and fetch one will bring the whole of this first row as a row vector.

So, it will bring in as all the components as one as a python string. And then you check whether it is empty or not, if it is empty then the I mean there is nothing to bring back, so you are done. So, you break otherwise you simply print the row, so you are printing there. And then you go back again this is while true. So, what happens is the cursor will advance to the next row and get you the next row. Again you do the same thing go back it will come back to the same, but once you get an empty result, you know that there is nothing more to proceed and you break.

So, this is what is illustrated then there are some more this particular record is deleted, then again another record is inserted. And another select is done. So, this is how a native program here in this case python can interact with the database and do any of the SQL tasks that we were doing earlier with SQL interface, now we can be done from the python, so that is a basic ODBC mechanism. I particularly chose python to just give you a different flavour, you can we will have assignments on doing it for C and using jdbc on java as well.

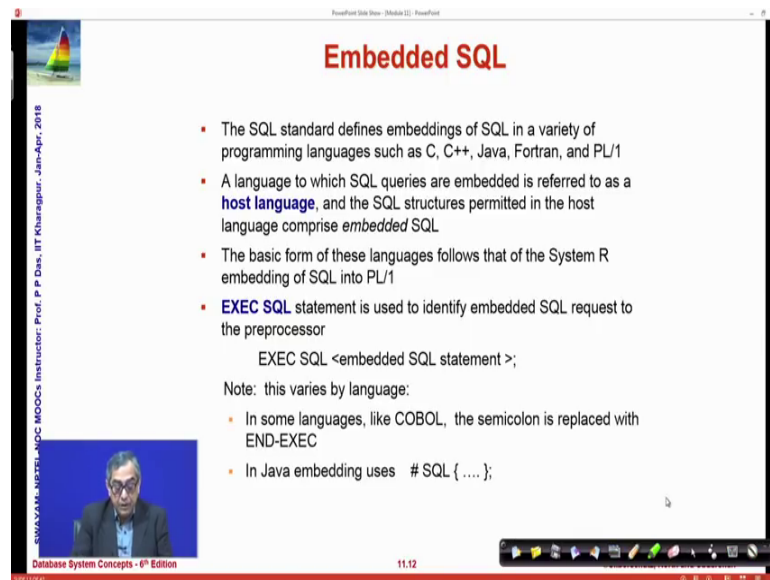
(Refer Slide Time: 12:19)



Now, I am come I will come back to the same interaction issue, but this time you can see that I am using a different diagram. The database is the same; this part is the same; the query processing is same, but instead of having a connection library, now I have put the SQL query itself as a part of the native program, the C program. So, this is what is called embedding.

So, you say I embed I put the SQL as a part of C, but naturally SQL is not C. So, we need to put certain additional syntax in C, so that I can directly write SQL as a part of the C program. I am talking about C, again just as an example if this is true for other several other languages which can be used as used for embedding SQL within them.

(Refer Slide Time: 13:09)



The screenshot shows a presentation slide titled "Embedded SQL". The slide content is as follows:

Embedded SQL

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, C++, Java, Fortran, and PL/1
- A language to which SQL queries are embedded is referred to as a **host language**, and the SQL structures permitted in the host language comprise *embedded SQL*
- The basic form of these languages follows that of the System R embedding of SQL into PL/1
- **EXEC SQL** statement is used to identify embedded SQL request to the preprocessor

```
EXEC SQL <embedded SQL statement >;
```

Note: this varies by language:

- In some languages, like COBOL, the semicolon is replaced with END-EXEC
- In Java embedding uses `# SQL { ... };`

At the bottom of the slide, there is a small video inset of a man speaking, a taskbar with the date 11.12, and a footer that reads "Database System Concepts - 6th Edition".

So, this is called the embedded SQL, it works for C, C plus plus, java fortran etcetera. And the language native language in which your embedding is called the is known as a host language. And basic form of these languages allow that the are come from the System R. So, what is important is this particular statement EXEC-SQL. This EXEC-SQL written inside the body of a C program will tell the C compiler that this part is not C; this part is actually embedded SQL.

And it will be treated differently; it will be compiled by the SQL compiler within the C compiler, so that is the basic structure, so EXEC-SQL. And then you put the pure SQL statement the embedded SQL statement. So, let us go forward and see some of this, there are different syntax for different native language embedding.

(Refer Slide Time: 14:11)

The slide is titled "Embedded SQL (Cont.)" and contains the following content:

- Before executing any SQL statements, the program must first connect to the database. This is done using:
`EXEC-SQL connect to server user user-name using password;`
Here, *server* identifies the server to which a connection is to be established
- Variables of the host language can be used within embedded SQL statements. They are preceded by a colon (:) to distinguish from SQL variables (e.g., `:credit_amount`)
- Variables used as above must be declared within DECLARE section, as illustrated below. The syntax for declaring the variables, however, follows the usual host language syntax

```
EXEC-SQL BEGIN DECLARE SECTION
int :credit-amount;
EXEC-SQL END DECLARE SECTION;
```

The slide also features a small video inset of a man speaking and a taskbar at the bottom.

So, to be able to connect you say EXEC-SQL and connect to server username using password. So, we saw similar things in terms of ODBC based connection these three information needs to be specified which database server, who is the user, what is the password. So, here you say that in this form. And this particular statement you can write as a part of the C program. Now, naturally the question here is in the in the earlier case when we are using the connection library, your results came back through the cursor which you then could deal because the cursors are necessarily objects in your native language. So, in python the cursor was a particular object in the pyodbc library.

But now you have embedded the SQL in terms of your c program. So, naturally the results or whatever you are doing in C which needs to have a communication with the SQL statement need to be differentiated from the SQL names themselves. So, any native language variable that needs to be treated in SQL will be marked with a colon preceding it. So, credit amount of this a colon credit amount, so it becomes a SQL variable provided credit amount itself is a C variable, so that is the basic you know connection mechanics. There are far more details in that, but I am just giving you the glimpse.

Now, any region where you write the SQL, you can write it as exact SQL begin declare section, and END declare section this is where you specify what are the different what are the different C variables C declarations that need to be used for this SQL definition. I will just show you an example soon so that.

(Refer Slide Time: 16:15)

The slide is titled "Embedded SQL (Cont.)" and contains the following text:

- To write an embedded SQL query, we use the **declare c cursor for <SQL query>** statement. The variable *c* is used to identify the query
- Example:
 - From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable *credit_amount* in the host language
 - Specify the query in SQL as follows:

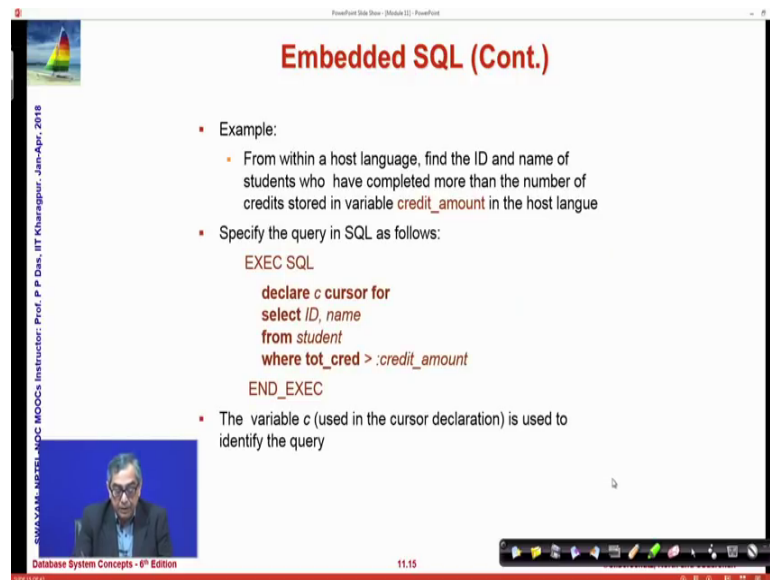
```
EXEC SQL
  declare c cursor for
  select ID, name
  from student
  where tot_cred > :credit_amount
END_EXEC
```

The slide also features a small video feed of a speaker in the bottom left corner and a taskbar at the bottom. The text "Database System Concepts - 6th Edition" is visible at the bottom left, and "11.14" is at the bottom center.

Similar to the ODBC style, you have a you will declare a cursor. So, this is to write the embedded SQL, you use declared c cursor for such and such SQL queries, so then that C, variable C will become your handle in the in the C language to be able to answer handle the query results. So, here is an example. So, you can see that credit amount is a variable in the host language.

So, you are using it in SQL with colon credit amount which says that it is this host language variable. So, that you can set a particular credit amount and go with that. And this is the query, and you have set a cursor on that which you can make use of in the exact SQL.

(Refer Slide Time: 17:14)



Embedded SQL (Cont.)

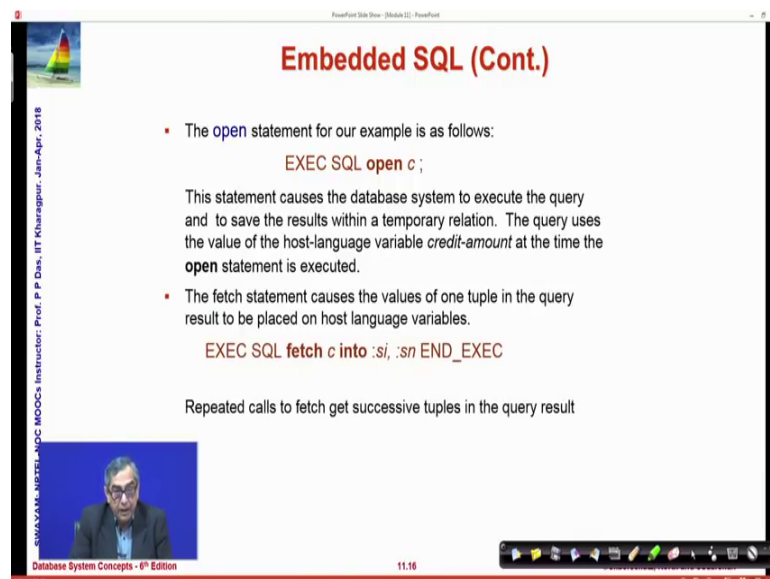
- Example:
 - From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable `credit_amount` in the host language
 - Specify the query in SQL as follows:

```
EXEC SQL
declare c cursor for
select ID, name
from student
where tot_cred > :credit_amount
END_EXEC
```
- The variable `c` (used in the cursor declaration) is used to identify the query

Database System Concepts - 6th Edition 11:15

So, let us this is the example continued.

(Refer Slide Time: 17:18)



Embedded SQL (Cont.)

- The `open` statement for our example is as follows:

```
EXEC SQL open c ;
```

This statement causes the database system to execute the query and to save the results within a temporary relation. The query uses the value of the host-language variable `credit-amount` at the time the `open` statement is executed.
- The `fetch` statement causes the values of one tuple in the query result to be placed on host language variables.

```
EXEC SQL fetch c into :s1, :s2 END_EXEC
```

Repeated calls to `fetch` get successive tuples in the query result

Database System Concepts - 6th Edition 11:16

Let us look at other features. You can once you have set a query you can actually execute that by the `open` statement. So, you say `EXEC-SQL open c`, the cursor. So, that will execute the query that you have associated with that cursor. And then once that has been done, then you can fetch the results into that cursor one by one; one tuple at a time.

(Refer Slide Time: 17:47)

Embedded SQL (Cont.)

- A variable called SQLSTATE in the SQL communication area (SQLCA) gets set to '02000' to indicate no more data is available
- The **close** statement causes the database system to delete the temporary relation that holds the result of the query.

EXEC SQL close c ;

Note: above details vary with language. For example, the Java embedding defines Java iterators to step through result tuples.

Database System Concepts - 6th Edition 11.17

Once you are done with all that then you simply close.

(Refer Slide Time: 17:52)

Embedded SQL - C Example

- The program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen

```
int main() {
    EXEC SQL INCLUDE SQLCA;
    EXEC SQL BEGIN DECLARE SECTION;
    int orderID; /* Employee ID (from user) */
    int custID; /* Retrieved customer ID */
    char salesperson[20] /* Retrieved salesperson name */
    char status[6] /* Retrieved order status */
    EXEC SQL END DECLARE SECTION;

    /* Set up error processing */
    EXEC SQL WHENEVER SQLERROR GOTO query_error;
    EXEC SQL WHENEVER NOT FOUND GOTO bad_number;

    /* Prompt the user for order number */
    printf("Enter order number: ");
    scanf("%d", &orderID);

    /* Execute the SQL query */
    EXEC SQL SELECT custID, salesperson, status
    FROM Orders
    WHERE orderID = :orderID
    INTO :custID, :salesperson, :status;

    /* Display the results */
    printf("Customer number: %d\n", custID);
    printf("Salesperson: %s\n", salesperson);
    printf("Status: %s\n", status);
    exit();

query_error:
    printf("SQL error: %d\n", sqlca->sqlcode);
    exit();

bad_number:
    printf("Invalid order number.\n");
    exit();
}
```

- The statement used to return the data is a singleton SELECT statement; that is, it returns only a single row of data. Therefore, the code example does not declare or use cursors

Source: <https://docs.microsoft.com/en-us/sql/odbc/reference/embedded-sql-example>

Database System Concepts - 6th Edition 11.18

So, let us look at an example. This is a program which will prompt the user for an order number, and retrieve the customer number. I mean given an order number it will retrieve the customer number, salesperson, status of the order and it will display that as the retrieved information on the screen. So, here is a C program. It starts on here with the main. So, you can see that this says that EXEC-SQL include, SQLCA, SQLCA is the communication area. So, there is an exchange going on between the C program

and SQL program. So, the area that is used by both for this transaction is known as SQLCA.

Then you have the declare section. So, you are saying these are SQL exec declaration. So, these are, but within that what you have are pure c declaration, but all the declarations of c that are put within this can be used in SQL query with a colon at the beginning. So, that the value can be exchanged to the SQLCA then in the next you are specifying what will happen if you have an error.

So, you say SQL look at this EXEC-SQL whenever SQL error go to query error. So, it will go to this level. If it is not found that is no result is there it will go to this. So, you by making sure that if there is some error that happens in the SQL part, what will happen in your C program. And then subsequently you have sample C program which reads the order number, and after having read that you are doing this.

So, what does it do, EXEC-SQL is to say that this is embedded; this is the select query starts, the fields, the relation, the condition. And then there are three attributes. So, you are setting an association with three variables in the C program. So, if I want to the result of this select will be a table of three attributes three columns, so we are giving a name to each one of these three attributes in terms of our C program.

So, there is a cust id. So, I say the cust id attribute in SQL is colon cust id here which is basically cust id variable in the C program. Let me clean up again. So, this is cust id; this is in SQL; this is my C program, and this is my C program variable in SQL which corresponds to this its similarly order based. Similarly, this is in SQL attribute this is a array of character here. And this is a correspondence; this is a correspondence.

So, now, you can easily see that once this has been done I will be able to get all these values here, normally the program would be longer the you will have to iterate over the cursor as you did in case in the ODBC case. But in this case since we are using one order number we know that there will be only one record. So, we have not shown the iteration on the cursor.

So, once you have got this you are you are using those values to print out the result. And in case this query has got into some problem because of SQL, then it will automatically jump to SQL query this particular level. If it has got a bad number which means that no

such record was found, it was a null table then it will immediately take you to this. So, this is how the embedded SQL worked.

So, there are these are two different styles the ODBC style and the embedding style are two different styles. If you have if you depending on the preference you use that earlier days people used to use more of embedding, I believe that now the preference is more for the ODBC kind of connection oriented system ODBC jdbc kind of because they are certainly more programmer friendly this.

(Refer Slide Time: 22:05)

Updates Through Embedded SQL

- Embedded SQL expressions for database modification (**update**, **insert**, and **delete**)
- Can update tuples fetched by cursor by declaring that the cursor is for update

EXEC SQL

```
declare c cursor for
select *
from instructor
where dept_name = 'Music'
for update
```

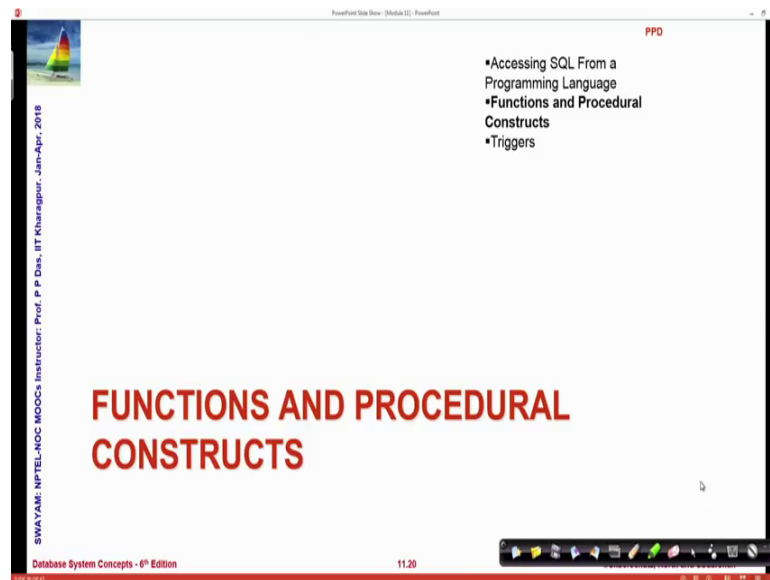
- We then iterate through the tuples by performing **fetch** operations on the cursor (as illustrated earlier), and after fetching each tuple we execute the following code:

```
update instructor
set salary = salary + 1000
where current of c
```

Database System Concepts - 6th Edition 11:19

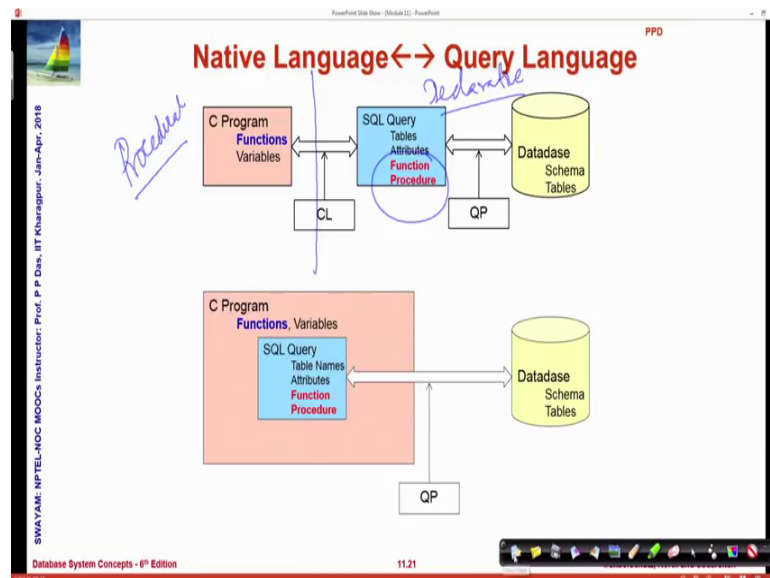
You can do updates through embedded SQL as well. So, I will not go through the details you can just go through this and understand that.

(Refer Slide Time: 22:15)



Let me move onto the next advanced part of SQL which is function and procedural construct.

(Refer Slide Time: 22:27)



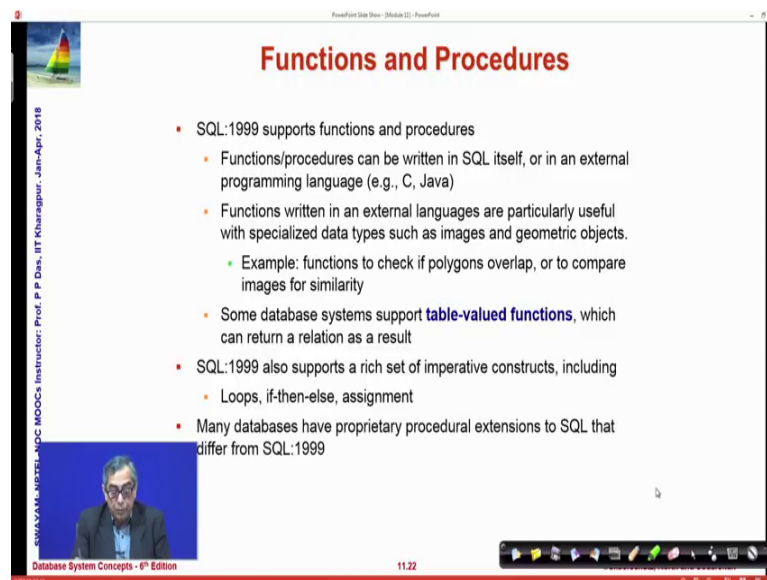
Now mind you again this is these are the two models that we have I have already shown you. The two models in which the application program the native language program and the query language can interact the ODBC mechanism and the embedded mechanism. But what we are now empowering is so far our basic premise was that this site is a procedural I discussed this at the very beginning. And this side is declarative. So, in SQL

you do not say that how you find out a result, you say what you want as a result, these are more like predicates. And in C program, you cannot specify what you want as a result you would rather say that do step one, step two, step three and you will get this result.

So, C program is all full of functions again I am talking about C as a placeholder it is true for most of the programming languages we use otherwise. So, there are procedural languages. So, procedures in C are functions; whereas, in SQL you had select from where kind of conditional clause.

Now, what we are saying that SQL also in later version have allowed certain functions and procedures, which can be part of SQL. It also has allowed certain imperative constructs like case like loop like while, repeat those kind of to make certain kind of procedural programming easier in SQL. And naturally these again can be used in conjunction with the connection oriented applications with the native or embedded oriented mechanisms. So, we will just take a quick look into some of these function and procedural features.

(Refer Slide Time: 24:15)



The screenshot shows a presentation slide with the title "Functions and Procedures" in red. The slide content is as follows:

- SQL:1999 supports functions and procedures
 - Functions/procedures can be written in SQL itself, or in an external programming language (e.g., C, Java)
 - Functions written in an external languages are particularly useful with specialized data types such as images and geometric objects.
 - Example: functions to check if polygons overlap, or to compare images for similarity
 - Some database systems support **table-valued functions**, which can return a relation as a result
- SQL:1999 also supports a rich set of imperative constructs, including
 - Loops, if-then-else, assignment
- Many databases have proprietary procedural extensions to SQL that differ from SQL:1999

On the left side of the slide, there is a vertical text: "COMPUTER NETWORKS AND MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018". At the bottom left, there is a small video inset showing a man speaking. At the bottom right, there is a Windows taskbar with the time 11:22. The slide footer reads "Database System Concepts - 6th Edition".

So, this started coming in from SQL 1999. And you can have functions and procedure written in SQL itself and function and procedures written external language or the host language also. So, both of them are available. What is interesting is some database systems support functions which are table valued. Functions, we always know functions

written objects only, but in SQL you can have functions which have table valued which written new functions. So, and certain imperative constructs have come in.

(Refer Slide Time: 24:56)

SQL Functions

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

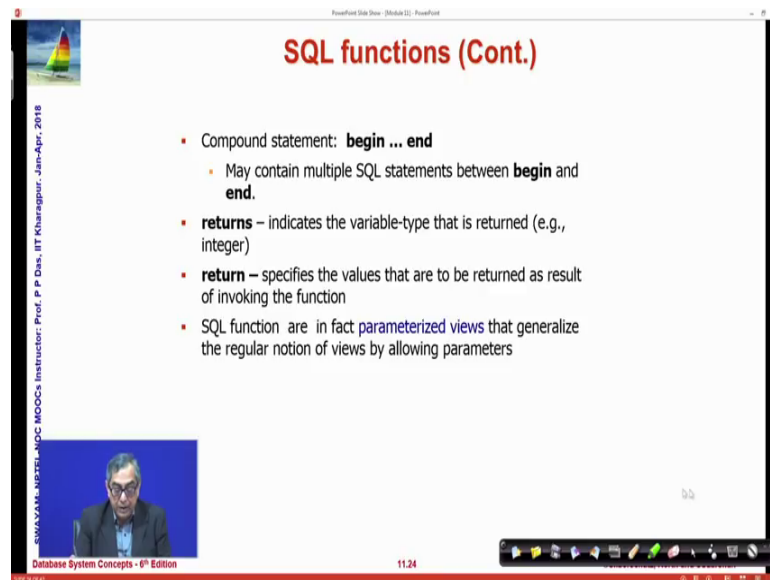
```
create function dept_count (dept_name varchar(20))  
returns integer  
begin  
declare d_count integer;  
select count (*) into d_count  
from instructor  
where instructor.dept_name = dept_name  
return d_count;  
end
```
- The function `dept_count` can be used to find the department names and budget of all departments with more than 12 instructors.

```
select dept_name, budget  
from department  
where dept_count (dept_name) > 12
```

So, there are several databases have their proprietary constructs and all that also. So, at this level of the course since we are not focusing particularly on any specific database systems, we will not talk about those. We can do enough in terms of the standard SQL itself. So, this is how you define a function which looks very similar to the SQL definition, create function, give it a name certainly here are the parameters. And here is a return type. So, if you are familiar with C, C plus plus, Java you I mean it is just that the syntax is different, but the elements are same.

There is a begin end in the scope. And this is the pure SQL that you are writing here. So, this is a function which is not a function in C, this is a function in SQL. So, this you can write this function in SQL. And once you have written that function then you can actually use that. So, if you look at the function is department name, then separately I am writing a query, I am using this department name as function. So, whenever I whenever this query will get executed, this function will be called, and the corresponding values will get returned. So, this is the basic mechanism ok.

(Refer Slide Time: 26:16)

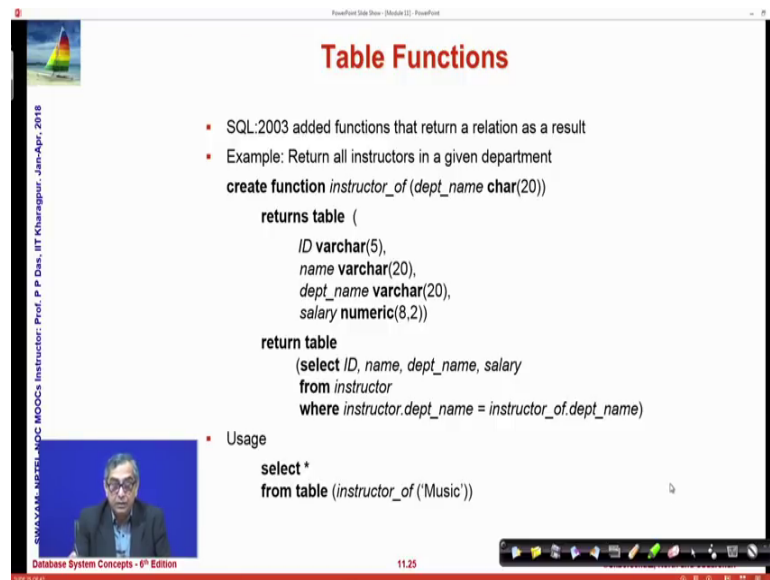


The slide is titled "SQL functions (Cont.)" and contains a bulleted list of details about SQL functions. On the left side, there is a vertical text string: "SWAYAM-NPTEL-IITK MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018". At the bottom left, there is a small video feed of a man with glasses. At the bottom right, there is a navigation bar with icons and the text "Database System Concepts - 6th Edition" and "11.24".

- Compound statement: **begin ... end**
 - May contain multiple SQL statements between **begin** and **end**.
- **returns** – indicates the variable-type that is returned (e.g., integer)
- **return** – specifies the values that are to be returned as result of invoking the function
- SQL function are in fact **parameterized views** that generalize the regular notion of views by allowing parameters

So, so there are SQL functions have several details which you can go through it has returned naturally.

(Refer Slide Time: 26:23)

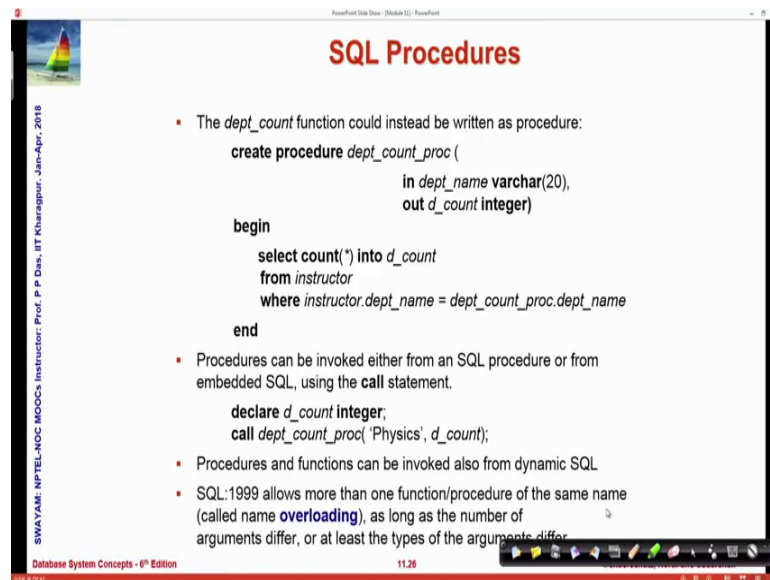


The slide is titled "Table Functions" and contains a bulleted list of details about table functions. On the left side, there is a vertical text string: "SWAYAM-NPTEL-IITK MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018". At the bottom left, there is a small video feed of a man with glasses. At the bottom right, there is a navigation bar with icons and the text "Database System Concepts - 6th Edition" and "11.25".

- SQL:2003 added functions that return a relation as a result
- Example: Return all instructors in a given department
create function *instructor_of* (*dept_name* char(20))
returns table (
 ID varchar(5),
 name varchar(20),
 dept_name varchar(20),
 salary numeric(8,2))
return table
 (select *ID*, *name*, *dept_name*, *salary*
 from *instructor*
 where *instructor.dept_name* = *instructor_of.dept_name*)
- Usage
select *
from table (*instructor_of* ('Music'))

And you can have table functions where it can return table. So, the syntax is given again its clear to see what will happen if you return a table which is computed from the select from where query that you have within the function.

(Refer Slide Time: 26:37)



The slide is titled "SQL Procedures" in red. It contains a list of bullet points and a code block. The code block shows the creation of a procedure named `dept_count_proc` that takes a department name as input and returns the count of instructors in that department. The slide also discusses how procedures can be invoked and that SQL:1999 supports overloading.

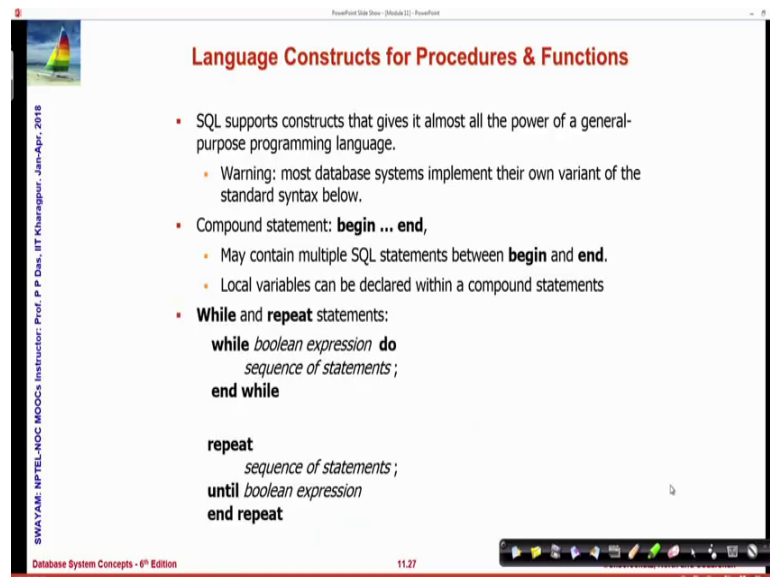
```
create procedure dept_count_proc (  
    in dept_name varchar(20),  
    out d_count integer)  
  
begin  
    select count(*) into d_count  
    from instructor  
    where instructor.dept_name = dept_count_proc.dept_name  
  
end
```

- The `dept_count` function could instead be written as procedure:
- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the `call` statement.

```
declare d_count integer;  
call dept_count_proc('Physics', d_count);
```
- Procedures and functions can be invoked also from dynamic SQL
- SQL:1999 allows more than one function/procedure of the same name (called name **overloading**), as long as the number of arguments differ, or at least the types of the arguments differ.

I can have SQL procedures which just performs some action, but does not have a return value so to say. And you can use them they can declare and call those procedures explicitly. Procedures can be functions and procedures can be overloaded as well if you are on SQL 99.

(Refer Slide Time: 27:08)



The slide is titled "Language Constructs for Procedures & Functions" in red. It contains a list of bullet points and code blocks. The code blocks show the syntax for `while` and `repeat` statements.

- SQL supports constructs that gives it almost all the power of a general-purpose programming language.
 - Warning: most database systems implement their own variant of the standard syntax below.
- Compound statement: **begin ... end**,
 - May contain multiple SQL statements between **begin** and **end**.
 - Local variables can be declared within a compound statements
- **While** and **repeat** statements:

```
while boolean expression do  
    sequence of statements;  
end while
```



```
repeat  
    sequence of statements;  
until boolean expression  
end repeat
```

Now, there are several language constructs as I mentioned SQL does allow while repeat. So, I am I am just covering them in terms of completeness it is not that these are frequently used features or I recommend that you do lot of them right here. If you need

to do procedural thing its always better to do them outside, but in some cases it may be easier to code a query if you can write a while, repeat kind of loop.

(Refer Slide Time: 27:37)

Language Constructs (Cont.)

- **For loop**
 - Permits iteration over all results of a query
- Example: Find the budget of all departments

```
declare n integer default 0;
for r as
  select budget from department
do
  set n = n + r.budget
end for
```

Database System Concepts - 6th Edition

You can write a for loop which iterates over the records of a table which is certainly very convenient. So, if you want to do something over the records of a table compute something that the for loop will become easier.

(Refer Slide Time: 27:50)

Language Constructs (Cont.)

- Conditional statements (**if-then-else**)
SQL:1999 also supports a **case** statement similar to C case statement
- Example procedure: registers student after ensuring classroom capacity is not exceeded
 - Returns 0 on success and -1 if capacity is exceeded
 - See book (page 177) for details
- Signaling of exception conditions, and declaring handlers for exceptions

```
declare out_of_classroom_seats condition
declare exit handler for out_of_classroom_seats
begin
...
.. signal out_of_classroom_seats
end
```

- The handler here is **exit** -- causes enclosing **begin..end** to be exited
- Other actions possible on exception

Database System Concepts - 6th Edition

You have a conditional statement case actually we have seen the case already. So, which is very easy in terms of coding many of the features so, here are some of them then you

have exceptions. So, I am not going through these, these in depth, but this is just to making you aware that while SQL continues to be predominantly a declarative language, it does have quite a bit of procedural support which in an appropriate time can be used if required. So, you can look up the manual for that. And there are a whole lot of things you can do with the external language routines.

(Refer Slide Time: 28:30)

External Language Routines*

- SQL:1999 permits the use of functions and procedures written in other languages such as C or C++
- Declaring external language procedures and functions

```
create procedure dept_count_proc(in dept_name varchar(20),
                                out count integer)
language C
external name '/usr/avi/bin/dept_count_proc'

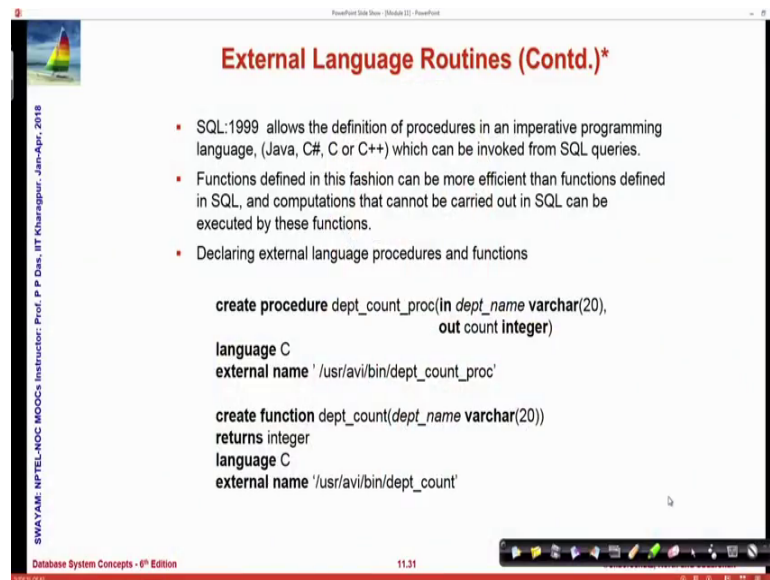
create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'
```

Database System Concepts - 6th Edition 11:30

That is can write a function in C and actually call it from SQL, this is just doing the other way round. Earlier in terms of embedding or in terms of ODBC, a C function was executing a query. Now, you are doing the reverse you are saying that I can write a SQL query which uses a C function that already exist. So, there are external ways of binding, I will not go through these slides in depth, because again the you will have to come a certain way before you can actually start using such features.

But get to know that there could be as from SQL you can use any external language library; and if some library is good for certain computation which is needed for your query which is a non database kind of computation then you can make use of this external language routines.

(Refer Slide Time: 29:23)



SWAYAM: NPTEL-NOC IITK Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

External Language Routines (Contd.)*

- SQL:1999 allows the definition of procedures in an imperative programming language, (Java, C#, C or C++) which can be invoked from SQL queries.
- Functions defined in this fashion can be more efficient than functions defined in SQL, and computations that cannot be carried out in SQL can be executed by these functions.
- Declaring external language procedures and functions

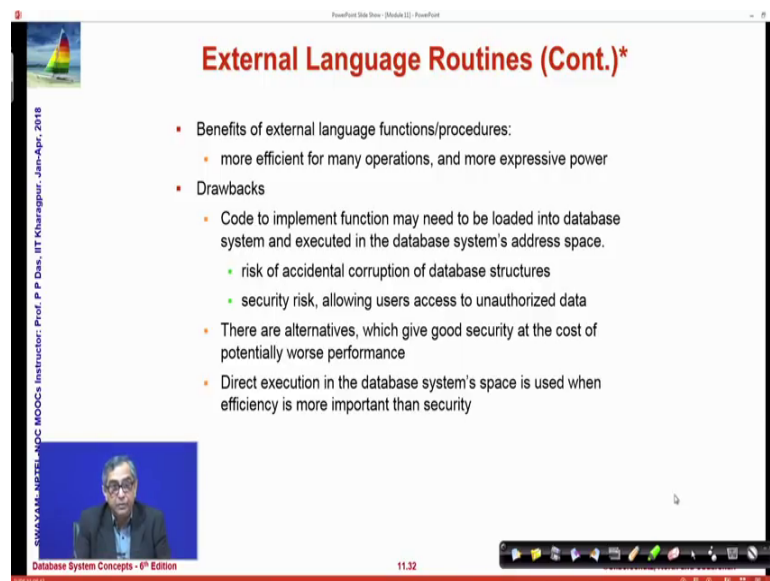
```
create procedure dept_count_proc(in dept_name varchar(20),
                                out count integer)
language C
external name '/usr/avi/bin/dept_count_proc'

create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'
```

Database System Concepts - 6th Edition 11.31

So, I have put together all the basic features that external language routines will need.

(Refer Slide Time: 29:30)



SWAYAM: NPTEL-NOC IITK Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

External Language Routines (Cont.)*

- Benefits of external language functions/procedures:
 - more efficient for many operations, and more expressive power
- Drawbacks
 - Code to implement function may need to be loaded into database system and executed in the database system's address space.
 - risk of accidental corruption of database structures
 - security risk, allowing users access to unauthorized data
 - There are alternatives, which give good security at the cost of potentially worse performance
 - Direct execution in the database system's space is used when efficiency is more important than security

Database System Concepts - 6th Edition 11.32

But again I would tell you that there are benefits, but there are lot of drawbacks in using them. And I would not recommend that you frequently use these features. You should primarily restrict to SQL programming, and then anything that you need to do in the native, you should go to choose your language and do that.

(Refer Slide Time: 29:48)

Security with External Language Routines*

- To deal with security problems, we can do on of the following:
 - Use **sandbox** techniques
 - That is, use a safe language like Java, which cannot be used to access/damage other parts of the database code.
 - Run external language functions/procedures in a separate process, with no access to the database process' memory.
 - Parameters and results communicated via inter-process communication
- Both have performance overheads
- Many database systems support both above approaches as well as direct executing in database system address space.

Database System Concepts - 6th Edition 11.33

There are security issues also that you will need to understand here.

(Refer Slide Time: 29:58)

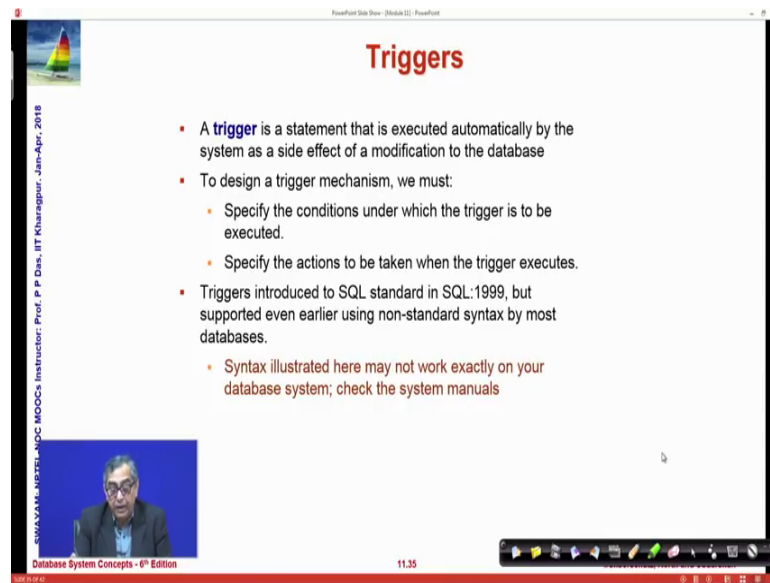
TRIGGERS

- Accessing SQL From a Programming Language
- Functions and Procedural Constructs
- Triggers

Database System Concepts - 6th Edition 11.34

Finally, before we close I will just mention a another feature called triggers, triggers are very important.

(Refer Slide Time: 30:04)



The screenshot shows a presentation slide titled "Triggers" in red text. The slide content is as follows:

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database
- To design a trigger mechanism, we must:
 - Specify the conditions under which the trigger is to be executed.
 - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
 - Syntax illustrated here may not work exactly on your database system; check the system manuals

On the left side of the slide, there is a vertical text: "©2012 WILEY-INTERSCIENCE MOOCs Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr, 2018". At the bottom left, there is a small video inset of a man speaking. At the bottom right, there is a Windows taskbar showing the time as 11:35 and the text "Database System Concepts - 6th Edition".

A trigger is a statement that is executed automatically when something happens in the database. So, you want that well things are happening. And well I want to know if a particular value has exceeded a certain level or if something has become null or some violatory things are happening and so on. So, how do you know that because a database is being accessed by hundreds and thousands of people and with hundreds of tables and millions of records.

So, triggers are a mechanism by which you can set that under this condition, I want a trigger, I want something specifically to happen. So, again they were introduced in 99, but earlier also triggers were there, but in 99 standard they became formal earlier they were somewhat you know differently structured. So, you might find that the system that you are using for practice the trigger in that may have a different format and semantics then the what we are discussing here.

(Refer Slide Time: 31:12)

Triggering Events and Actions in SQL

- Triggering event can be **insert, delete or update**
- Triggers on update can be restricted to specific attributes
 - For example, **after update of takes on grade**
- Values of attributes before and after an update can be referenced
 - **referencing old row as** : for deletes and updates
 - **referencing new row as** : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. For example, convert blank grades to null.

```
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
when (nrow.grade = '')
begin atomic
set nrow.grade = null;
end;
```

Database System Concepts - 6th Edition 11:38

So, the most common triggering events are insert, delete, update. So, if something some update is happening, so I can say that after this update, I want such and such things to happen. Or I can during the update I can one that well I am doing an update. So, there is an old value which is typically referred to as old row, the row that is getting updated. And there is a new row the new set of values that are getting created. So, I might want between the old row and the new row that certain things happen.

So, I can say that well just look into this. So, again the syntax is all similar. Create trigger, trigger there is a name of the trigger and this is the condition. Before update of takes, takes is a relation referencing new row as row n row. So, this is the new value that we set. Now, what are you saying, saying that for each row what you do when n grid is blank n row dot grade is blank that is if this is if you are updating and you have got a got you are going to update, a grade value which is blank then you simply set it to null.

So, it is possible that the grade that has come in and grades are characters and what has come in from the input and is going to get updated is a show a certain grade to be now because it may not have been decided. Now, you do not want those blank values to be present. You want because blank cannot be checked, we have we have checkers for null and so on.

So, you want to set that to null. So, trigger can make this thing happened because otherwise how will you know what value is actually getting changed. So, there could be several ways trigger can be used.

(Refer Slide Time: 33:23)

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
and (orow.grade = 'F' or orow.grade is null)
begin atomic
update student
set tot_cred= tot_cred +
(select credits
from course
where course.course_id= nrow.course_id)
where student.id= nrow.id;
end;
```

Database System Concepts - 6th Edition

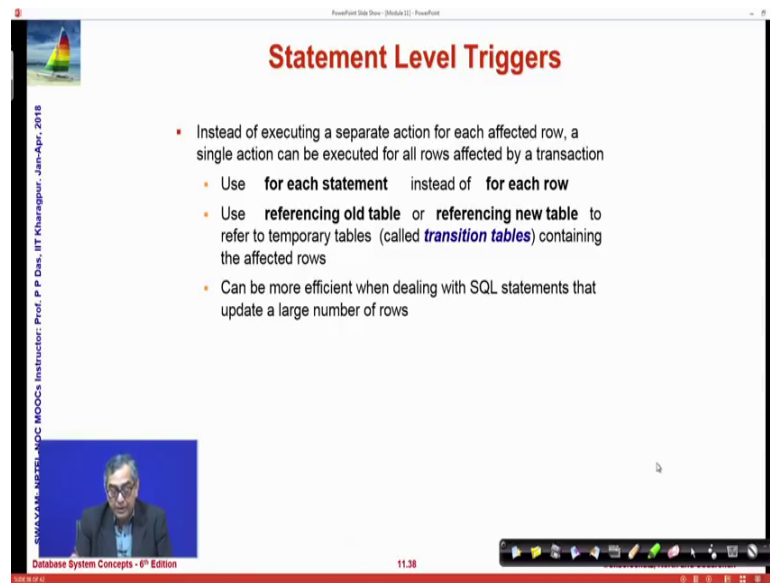
11:37

For example, this is showing one that as you change the grades then certainly based on the grades credit earned value is computed. So, as a greatest change if it is now once grades have been entered say for 200 students. Now, after reviews grades for three of them are getting changed. So, how do you know that for those students, the change of the grade may impact the computation of the on credits.

So, you would like to update that on credits or it may or may not be required. So, the trigger will tell you that after update. So, whenever the update happens you take the old and the new value n row and o row, and then you are putting some conditions that if that new row is grade is f, and is not null; old row is grade or it is not null, then you try to do this.

So, if it was failure, and if it continues to be failure, new grade is not f; if it is f then you do not have to do anything; if it is null it do not have to do anything. But if it is if it was f or it was null, and now it has become a different grade then certainly the computation to update the credits earned is required. So, and the trigger gives you the right point when you can do this because otherwise you will not know in terms of millions of updates happening when this particular thing is going on.

(Refer Slide Time: 35:08)



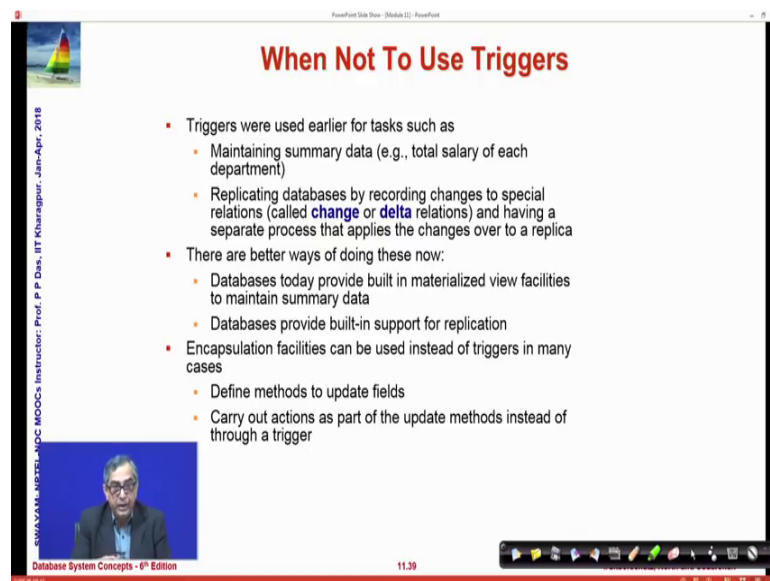
The slide is titled "Statement Level Triggers" in red text. It contains a bulleted list of points. On the left side, there is a vertical text string "S.W.A.S.A.T.H. N.P.S.T.U. - IGC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018" and a small video inset of a man speaking. The bottom of the slide shows a Windows taskbar with the time 11:38 and the text "Database System Concepts - 6th Edition".

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
 - Use **for each statement** instead of **for each row**
 - Use **referencing old table** or **referencing new table** to refer to temporary tables (called **transition tables**) containing the affected rows
 - Can be more efficient when dealing with SQL statements that update a large number of rows

Triggers can be on statements as well you can decide leave for your reading.

(Refer Slide Time: 35:12)



The slide is titled "When Not To Use Triggers" in red text. It contains a bulleted list of points. On the left side, there is a vertical text string "S.W.A.S.A.T.H. N.P.S.T.U. - IGC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018" and a small video inset of a man speaking. The bottom of the slide shows a Windows taskbar with the time 11:39 and the text "Database System Concepts - 6th Edition".

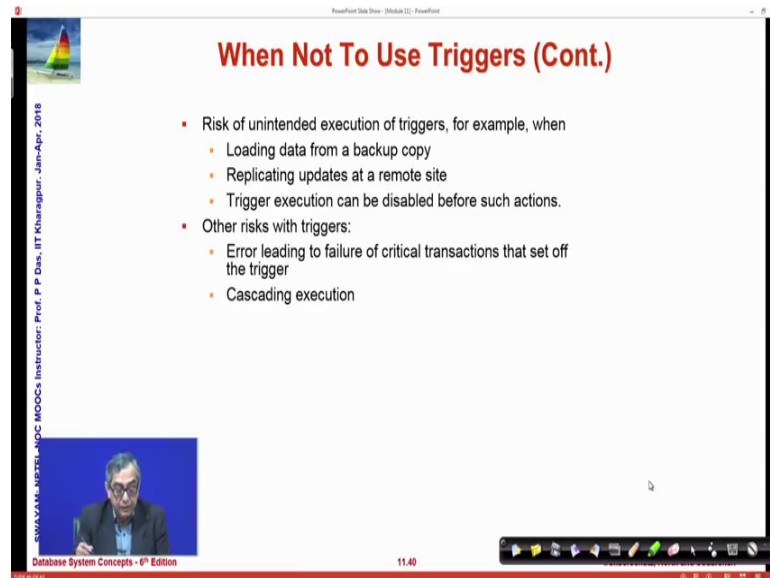
When Not To Use Triggers

- Triggers were used earlier for tasks such as
 - Maintaining summary data (e.g., total salary of each department)
 - Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
 - Databases today provide built-in materialized view facilities to maintain summary data
 - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
 - Define methods to update fields
 - Carry out actions as part of the update methods instead of through a trigger

But you have to be careful that triggers sounds very interesting and what happens particularly with the early stage of programming people get overboard with triggers and start using them severely, but triggers do have a lot of overhead. So, you should not many of the things that triggers can do, can be done through other means for example, by materialized, views and so on. So, as we go along we will mention that these are the problems that need to solve get solved by triggers; otherwise normally you should think

twice before you actually use a triggers. So, there could be different other ways of solving the same problem.

(Refer Slide Time: 35:55)



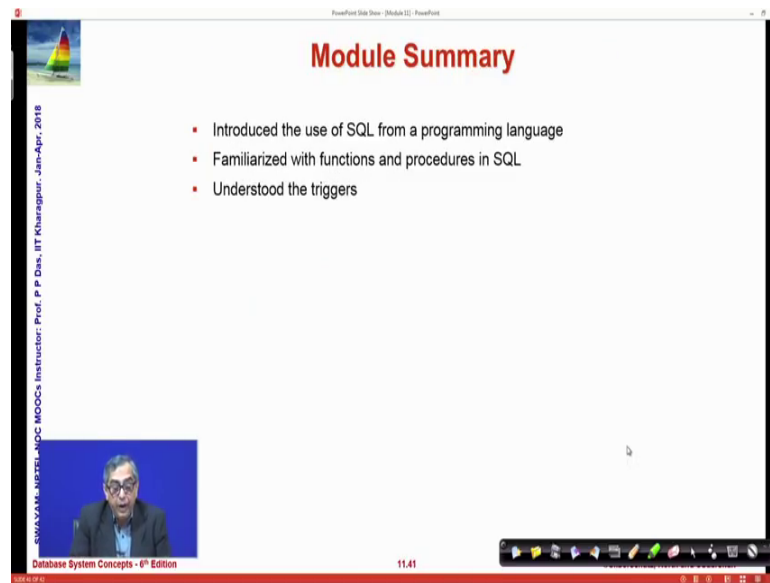
The image shows a presentation slide titled "When Not To Use Triggers (Cont.)". The slide is part of a video recording, as evidenced by the "Zoom Meeting" watermark and the "Database System Concepts - 6th Edition" text at the bottom. The slide content is as follows:

- Risk of unintended execution of triggers, for example, when
 - Loading data from a backup copy
 - Replicating updates at a remote site
 - Trigger execution can be disabled before such actions.
- Other risks with triggers:
 - Error leading to failure of critical transactions that set off the trigger
 - Cascading execution

And because you have to keep in mind that triggers are expensive because once you have triggers and actually internally database for every update. If you have an update trigger on a field or a relation, then with every transaction with every change the database has to check if your trigger is true or not and so therefore, there is a cost to that. The other thing is there are triggers are for the live execution.

So, if you have offline for example, you are loading the data from a backup copy or you are replicating your database at a remote site and so on, then you have to put off the trigger; otherwise you know falsely the triggers will start happening and that may have a catastrophic effect that might trigger of different alarms and all that. So, you have to be careful with triggers in that manner so, cascading executions and all those.

(Refer Slide Time: 36:46)



The screenshot shows a presentation slide titled "Module Summary" in red text. The slide is part of a video lecture, as indicated by the "FreePost" window title and the presence of a video player interface at the bottom. On the left side of the slide, there is a vertical text overlay: "©2018 ANU, IIT Kharagpur, IIT Kharagpur, Jan-Apr, 2018" and "MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018". The main content of the slide is a bulleted list:

- Introduced the use of SQL from a programming language
- Familiarized with functions and procedures in SQL
- Understood the triggers

At the bottom of the slide, there is a small video thumbnail of the instructor, a clock showing "11:41", and a taskbar with various application icons.

So, to summarize we have and this kind of closes our direct discussion on SQL. So, we have introduced the use of the very important aspect the use of SQL from a programming language, the interface between the native language and query language boundary. And that is something which will be extremely useful for application programming. And we are familiarized with you know the imperative extensions of SQL, the procedural extensions of SQL in terms of functions and procedures that you can directly write in SQL. And we have just introduced concept of triggers, so that you can sniff what is going on in your database.

So, there is the quite a few other features of SQL as well majority of them are advanced features dealing with olap and several others, which I chose to skip at this level of the course. So, this will close our discussion on SQL. And next we will move onto the design of the database looking into the algebra and the modelling.