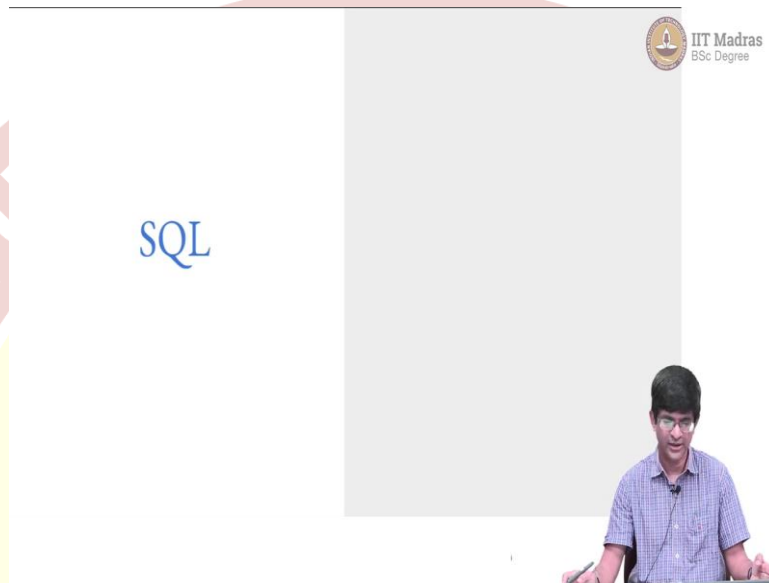


IIT Madras
ONLINE DEGREE

Modern Application Development – 1
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology Madras
SQL

Hello, everyone and welcome to this course on Modern Application Development.

(Refer Slide Time: 00:16)



So, now we are going to get to SQL, the Structured Query Language and try and understand it in a little bit more detail. Like I said, this course application development is not really something where we can go into the details of how databases are implemented or even the details of the syntax of SQL or languages just like that.

Ideally, you would need to have done a course in databases already, in order to benefit from all of this. On the other hand, purely from an implementation point of view, you can get by with just having a superficial understanding of what databases are capable of doing. Especially because we will be using a wrapper around the database in order to actually do our querying. So, we will not be writing raw SQL statements for the most part, at least. But understanding how SQL statements work and how queries happen inside the database is very useful to actually get a better picture of how the internals of the system work.

(Refer Slide Time: 01:30)

Relational Databases



- From IBM ~ 1970s
- Data stored in Tabular format:
 - Columns of tables: fields (name, address, department, ...)
 - Rows of tables: individual entries (student1, student2, ...)
- Key: unique way of accessing a given row
 - **Primary key**: important for fast access on large databases
 - **Foreign key**: connect to a different table - Relationships



So, as I said earlier, relational databases originated from some work done at IBM in the 1970s. It essentially indicates that you have data which is stored in a tabular format, so there are columns of tables, each column corresponds to a field within some kind of a structured data object and there are rows in the tables which correspond to individual elements, individual entries, may be multiple student records.

There are different terms that are here. So, rows are sometimes called records, columns are called fields and so on and one of the things which is important from the point of view of any implementation of the database is that you have a concept of a key, which is some unique way of accessing a given row in a database.

So, primary keys are important for fast access on large databases and what I mean by that is a primary key having a primary key, which is something like an integer, makes it easy to directly hook into some part of a database or some entry in the database and pull out a row corresponding to one particular value.

Rather than searching through the different rows in order to let us say, do a string match against a given name. If we use a primary key, which is an integer, there are much faster ways by which we can access the correct row and pull the data out. Foreign keys on the other hand, as we saw in the case of the entity relationship diagrams, are used in order to connect to a different table. So, a foreign key would indicate relationships between data, which is there in different tables.

(Refer Slide Time: 03:11)

Queries

- Retrieve data from the database:

eg. "Find students with name beginning with A"

"Find all courses offered in 2021"



Now with all of this, somehow magically, the database manages to store the information, it associates primary keys, foreign keys and so on with the data. What do we do with it? Ultimately, the goal of whatever we are trying to do is to figure out how to retrieve data from the database. Examples of queries that we might have for the database could be something like, find all students whose name begins with the letter A or find all courses that are offered in 2021 or probably, instead of students the letter A it might be all students of EE or maybe EE who are going to graduate or who joined in the year 2018 or something like that.

So, retrieving data from the database, you could if you had the primary key of the entry that you wanted, of course it is straight you just pull the data out. But the strength of the database comes when you can actually search through the data itself. So, even though we prefer to have primary keys that directly allow us to pull out the data corresponding to the database, there are usually some kind of indexing, which is done on the other keys.

So, for example, the roll number or the course id, in such a way that we would like it to be something that can be searched through quite fast and the whole process of using a database is the question of constructing the right set of queries, which will allow us to extract the data that we need from what is stored inside the database.

Structured Query Language (SQL)

- English - like, but structured
- Quite verbose
- Specific mathematical operations:
 - Inner Join
 - Outer Join



And that is where Structured Query Language or SQL, there are places where this is pronounced SQL, I prefer to call it SQL. It is, as far as I can tell a matter of choice, there is no sort of standardised definition over here. So, I will be using the term SQL in order to refer to this. So, SQL is a language which is sort of like English. I mean, the attempt was to make it something like, can I write queries the way that I would normally construct them in English.

But it is a more structured language because it has a specific syntax to it, you cannot just sort of, you know, have grammatically incorrect sentences and expect the computer to understand what you are saying. Now, the problem with having English like languages is that it ends up being quite verbose meaning that you need to write or type in a lot of text in order to get it to do anything useful and there are specific mathematical operations that are defined as part of the SQL operations set, which actually have specific meanings and it is based around those that all the optimizations on databases are done.

Now, you will come across terms like inner join, outer join and so on, which are things where what I will be doing is just giving some very high-level intuitive explanations for what they are, they have very specific internal meanings. But, the important point, at least from the point of view of using the database is to have a broad idea of how you can extract the right data that you need for a given requirement.

(Refer Slide Time: 06:27)

Example: Inner Join

Name	IDNumber	hostelID
Sunil Shashi	MAD001	1
Chetana Anantha	MAD002	2
Madhur Prakash	MAD003	2
Nihal Surya	MAD004	3
Shweta Lalita	MAD005	2
Raghu Balwinder	MAD006	3
Gulshan Kuldeep	MAD007	1
Kishan Shrivatsa	MAD008	1
Purnima Sunil	MAD009	2
Nikitha Madhavi	MAD010	1
Lilavati Prabhakar	MAD011	3
Rama Yamuna	MAD012	3

ID	Name	Capacity
1	Jamuna	300
2	Ganga	300
3	Brahmaputra	500



So, as an example of an inner join, let us say that I had this information and sort of expanded upon the student data set over here by adding one more column called the hostelID and I have a separate table out here which indicates hostels where I could have for example, IDs corresponding to each of the hostels a name corresponding to the hostel and some capacity, the number of students who can be in that hostel.

So, as you can see, this is just, I took the spreadsheet that I had may be added one more sheet to it and put in some extra information out here and in the student sheet, I basically added one more column. So that is, as I mentioned earlier, the power of the spreadsheet, it allows you to easily add fields or columns to a given data set.

(Refer Slide Time: 07:13)

Student - Hostel mapping

```
select Students.name, Hostels.name
  from Students
 inner join Hostels
on Students.hostelID = Hostels.ID
```

Sunil Shashi, Jamuna
Chetana Anantha, Ganga



Example: Inner Join

Name	IDNumber	hostelID
Sunil Shashi	MAD001	1
Chetana Anantha	MAD002	2
Madhur Prakash	MAD003	2
Nihal Surya	MAD004	3
Shweta Lalita	MAD005	2
Raghu Balwinder	MAD006	3
Gulshan Kuldeep	MAD007	1
Kishan Shrivatsa	MAD008	1
Purnima Sunil	MAD009	2
Nikitha Madhavi	MAD010	1
Lilavati Prabhakar	MAD011	3
Rama Yamuna	MAD012	3

ID	Name	Capacity
1	Jamuna	300
2	Ganga	300
3	Brahmaputra	500



Now, if I wanted to find the student to hostel mapping, that is to say I want to know which student is assigned to which hostel, I mean, this information is sort of available over here, but what I have is the student name and the hostelID not the hostel name. What I would like is a list of entries, which basically tells me that, this student is in this hostel, and so on, that information.

I would have an SQL query, which looks something like this, it would, usually be a select query and what it is saying is select the names of students and the names of the hostel, from the first table is students, and I am joining students to the table hostels, so these are two different tables in my database or in my spreadsheet and I am joining them together, in order to create some kind of a student hostel combination and what I am looking for is, can I find cases where the student hostelID is equal to the hostel, that is the ID in the hostel table.

And once I have that, it will basically pull out the corresponding student name and the hostel name. Because I have matched the student hostelID to the Id from the hostel table, it means that, when I am printing this out, it will print the name corresponding to that hostel which matched the student's hostelID.

Reasonably intuitive, easy enough to understand what it is trying to do over here. But you have to be a little careful, especially when you are trying to do, why did we need to do an inner join, the inner join is essentially saying take these two tables and join them together. Now, what is an inner join, what is an outer join and so on are mostly out of the scope of this course. So, I am not going to get into that over here.

Again, like I said, you can get by with a lot of what needs to be done in this course, without really understanding that, but having some knowledge of how those things work is useful simply from the point of view of, optimising performance of your system. Because if it turns out that you have a lot of complicated joints, in your queries that can slow your system down.

And of course, if we run a query like this, what would end up is that we would start getting data something like this, there would be one student's name and it will print out the corresponding hostel name, the next student's name, the next hostel name and so on, down the line.

Now, you might have noticed that while talking about this, I basically gave an example using spreadsheets and you know, in fact, I talked about tables in spreadsheets and so on. Now, spreadsheets as such, Google sheets or even other kind of spreadsheets, they do not have a good support for SQL directly and especially when you are in a situation where you want to sort of join across multiple tables that are on different sheets, it becomes hard to do that using, you do not really have a structured query language as part of the spreadsheet itself.

So, doing something like this with spreadsheets may not be easy, but with any kind of simple database and we will be looking at examples of that, when we are talking about the practical assignments in this course. You would find that something like this is very straightforward to set up and use.

(Refer Slide Time: 10:40)

Cartesian Product

- N entries in table 1
- M entries in table 2
- $M \times N$ combinations - filter on them

Powerful SQL queries can be constructed

IIT Madras BSc Degree

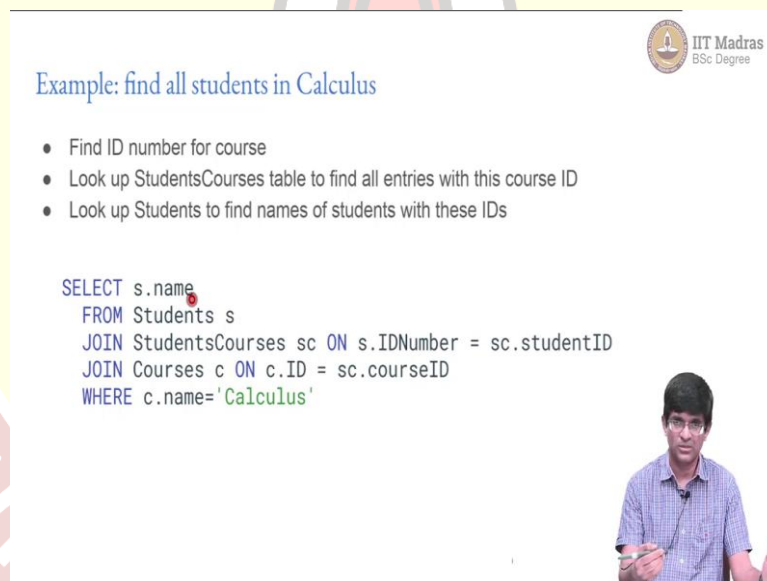
Now, similar to the inner join, which was basically sort of just trying to find combinations of students in hostels, there is something called the Cartesian product and the Cartesian product

is sort of saying that if I have N entries in table 1 and M entries in table 2, it basically is going to create N cross M, grid, a large grid out there, which would have, let us say, N rows, M columns and I would have entries out here, which basically correspond to the combinations of this row and in this table and the corresponding row in the other table.

Now, based on that, you can then sort of filter on the entries that are present in this combination and quite powerful queries can be constructed in this way. But as you can imagine, if N and M are large, the size of this Cartesian product can suddenly become extremely big and difficult to manage.

So, you need to be careful about how you use these different join queries in your actual searching through the data and for the most part, as long as your data structures and data types themselves are reasonably straightforward and simple to implement, this should not be a problem. But there are cases where you might need to keep this in mind.

(Refer Slide Time: 12:09)



Now, as one specific example, I am you know, once again, looking at the data that we had earlier, there are a list of students and there is a list of courses and supposing the question that comes up is how would you find all the students who are taking calculus. Now, how does that work? I first need to find out, what course is this? So, I only have the course name, which means that I now need to go and sort of do the inverse.

Because if I if you remember, the students courses table does not have course names in it. So, I need to somehow figure out, how can I go from a course name to a course id and from that course id, maybe I can filter on that students courses list in order to find out the student ids

and from there, maybe I can get back the list of students that is their names and one possible way away by which something like this could be constructed would be a sequence, which goes something like this, which essentially says, select s.names.

So, what is this s over here? It is essentially because I have from students s. So, I am sort of aliasing this students table using the letter S. I do not want to keep typing students.names, it is just too long for me. So, select s.name from students, which I am aliasing as s. But, if I just say select s.name from students s, it will just print out all the names of the students. What I wanted to do instead is to take the students table and join it on the students courses table.

But only in places where the student IDNumber that is from s and the studentID in the sc that is a student's courses table are the same. Are we done? Not yet, we need to go a bit further. I also would like to join on the courses and make sure that the courseID in the sc table is the same as the IDNumber in the courses table.

But where does this name calculus come in? That is the last part where I can give c.name is equal to calculus. So, if you look at an SQL query like this, what you need to be able to understand is something like you know, I have a name, the where is going to filter on that name, filter what? This c that is the courses table.

So, where c.name is equal to calculus, I will only have one c.ID corresponding to that and now I am doing this joint which means that I now have only this one c.ID corresponding to the c.name and that c.ID is being matched against the sc.courseID. Which means that at the end of this, I will now have some entries from the SC that is the student's courses table, which match with this courseID.

Now, for each of those entries, I am going to go and start looking at this line, which basically says, for each of those rows, go and check whether the studentID matches with the s.IDNumber, from where, from the actual table students and then finally, print out the names of those students. So, SQL in other words can be used in order to construct fairly complicated queries.

So, something which should normally have taken us multiple steps to do, can be done using a single statement. That is not necessarily the most efficient way of doing such a search. But on the other hand, can be quite effective and efficient, when we are actually trying to do something of this sort. So, with all of this in mind.

(Refer Slide Time: 15:57)

Summary

- Models - persistent data storage
- Mechanisms:
 - CSV, Spreadsheets, SQL, NoSQL
- Entities and Relationships
 - Different ways of representing

No details on display, views, or what kind of updates permitted



Let us summarise what we know about models at this point. The first thing is that it represents the data that you are working with, the actual underlying data model that we want to actually build upon. It corresponds to a persistent data storage, which means that I need to have some way of saving this data, representing it internally in my memory, as well as in case I need to restart my application or my server, I need to be able to save it out to disk and retrieve it as required.

There are multiple mechanisms for both representing the data as well as saving the data, we looked at comma separated values, spreadsheets, SQL, and NoSQL type of databases and of course, the data structures themselves are sort of assumed to be certain kinds of in memory, either a array or a dictionary or set of classes, various different ways by which I could have that.

Ultimately these models can be thought of in terms of entities, some kind of an abstract entity that holds information and the relationships between those entities and these are all different ways of representing this information. The most important point that I hope is an important takeaway from all of these lectures on the model is, you would have realised that by now that there was absolutely no discussion about views for example, how should this data be displayed?

HTML, in other words, was never mentioned throughout the course of this entire set of lectures on the model. Because of the separation that we are trying to keep in place, the model only refers to how we store the data and as we will see later, the controller will then have the job of taking inputs from the view and telling the model what to do and the model

then sort of updates in some way and gives information back to the view. As we will see, views sort of extract information using queries from the model, so that they get for example, a list of students or a list of courses all of those are extracted using queries.

But the model itself does not care what the view looks like. It only responds to a query and that separation is what allows us to, for example use the same database, whether it is a web app or a standalone application or it could be something where let us say I want to migrate from MySQL to SQLite to NoSQL or maybe spreadsheets right or the other way around. All of that can happen in an abstract behind the scenes manner without changing the rest of the implementation.

