INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

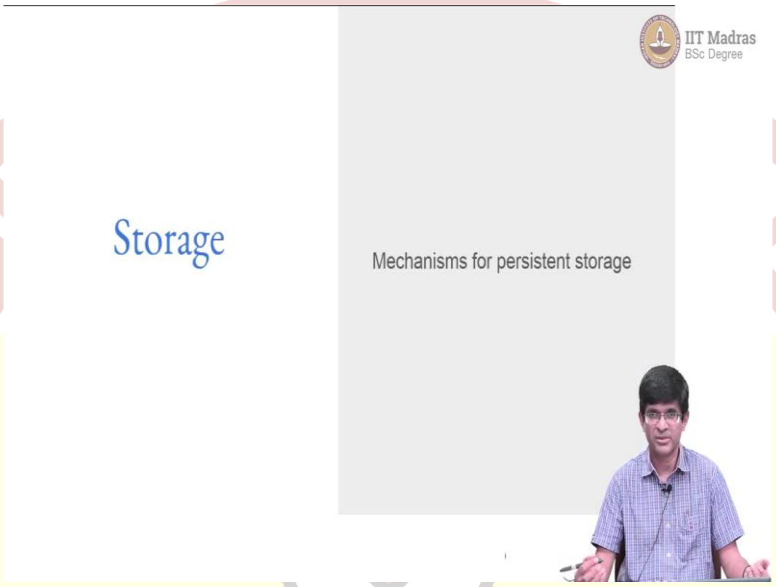
IIT Madras

ONLINE DEGREE

Modern Application Development – 1
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Mechanisms for Persistent Storage and Relation Databases

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

(Refer Slide Time: 00:17)

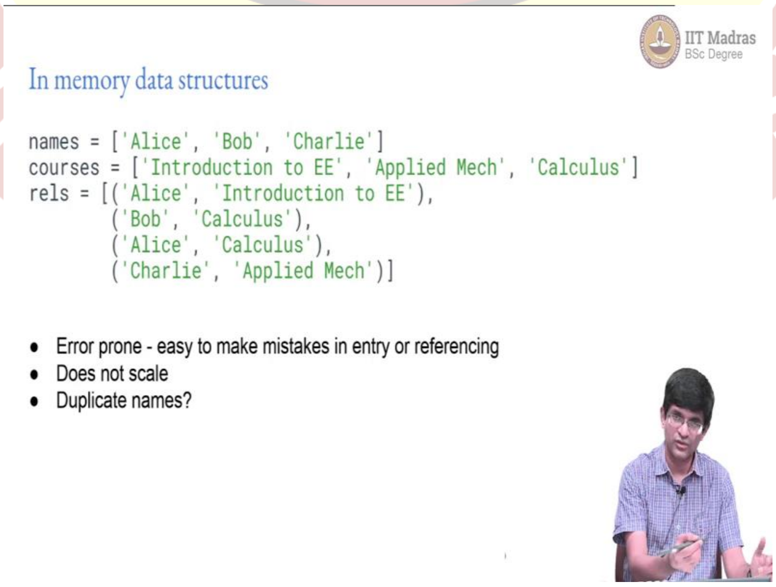


Storage

Mechanisms for persistent storage

So, first and foremost, let us look at mechanisms for persistent storage or how we might be able to store the information corresponding to relationships between different data in such a way that it can then be retrieved or loaded back later.

(Refer Slide Time: 00:32)



In memory data structures

```
names = ['Alice', 'Bob', 'Charlie']
courses = ['Introduction to EE', 'Applied Mech', 'Calculus']
rels = [('Alice', 'Introduction to EE'),
        ('Bob', 'Calculus'),
        ('Alice', 'Calculus'),
        ('Charlie', 'Applied Mech')]
```

- Error prone - easy to make mistakes in entry or referencing
- Does not scale
- Duplicate names?

Now, one way of thinking about it, before we even get to the question of saving it in a file or on disk somewhere, let us think in terms of what is actually being saved. Now, this is potentially something which just looks like you know, some Python data types. Each of these data types is basically a list as you can see by these square brackets that are applied around them and the entries over here essentially correspond to some names, I have chosen different names just to keep it simple.

Similarly, courses is also a list. In this case, it so happens that I have three courses and three students. But obviously, there is no requirement that you know, it is not a one-to-one mapping and then I have another entry, which basically gives the relations between the two and one possibility that I could use over here is to make use of Pythons data type called a tuple.

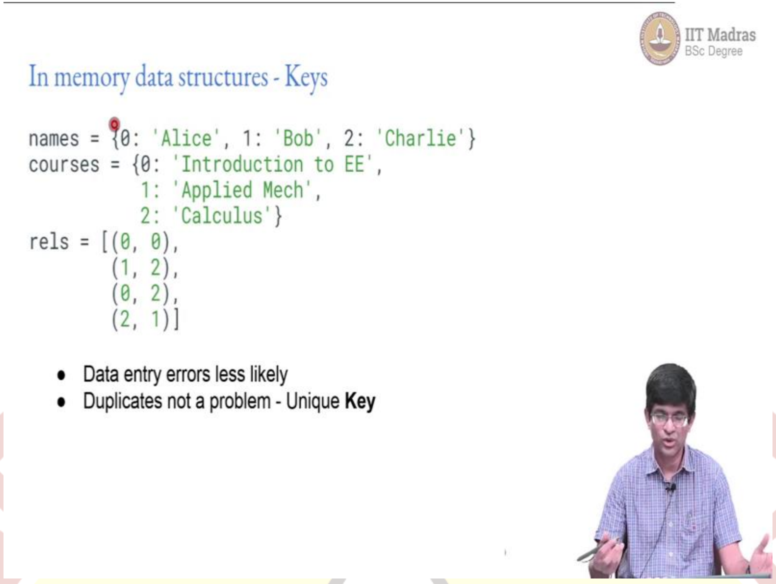
So, a tuple is essentially an ordered set. In this case, an ordered pair of elements. There are two elements. So, it is encapsulated within the parenthesis, there are two entries, the first one is a name and the second one is the name of a course. So, this gives the relationship saying that Alice has taken the course introduction to EE and similarly, Bob has taken the course calculus, Alice has also taken calculus, in addition to the introduction to EE and finally, Charlie has taken applied mech.

So, this is just one way by which it could be represented and as you can sort of make out from what we have over here, if I directly tried storing the relationships in this way, it is sort of error prone, it makes it easy to make mistakes when I am entering data or referencing. What I mean by that is each of these tuples. So, for example, calculus appears twice, Alice appears twice. In each of those cases, I need the full name of the person or of the course and it needs to be typed in.

Something like introduction to EE is a long name, it contains spaces. I mean, it is quite easy to make a mistake, it might be as simple as getting a capital letter instead of a small letter. The point is, this is not easy for a person to interact with, I do not want to introduce this possibility of error coming in here.

It also does not scale very easily, because I am storing quite a lot of information just to indicate these relationships and of course, it has this problem that what if I had more than one, Alice? Would this correspond to the same person or two different people? We already sort of saw how to solve parts of this problem.

(Refer Slide Time: 03:13)



In memory data structures - Keys

```
names = {0: 'Alice', 1: 'Bob', 2: 'Charlie'}
courses = {0: 'Introduction to EE',
           1: 'Applied Mech',
           2: 'Calculus'}
rels = [(0, 0),
        (1, 2),
        (0, 2),
        (2, 1)]
```

- Data entry errors less likely
- Duplicates not a problem - Unique **Key**

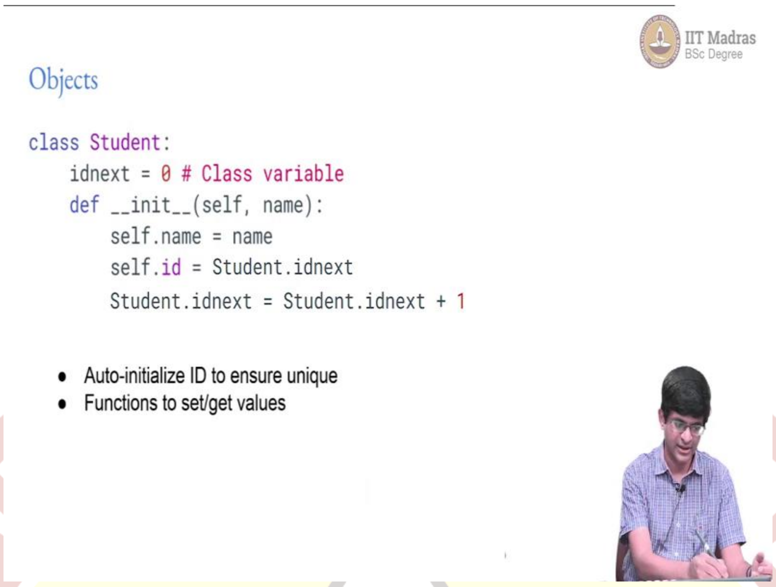
One way to do it would be for the names, I actually give a key or an ID. So, id number 0 corresponds to the name Alice id number 1 corresponds to the name Bob, and id number 2 corresponds to the name Charlie. Similarly, 0 corresponds to Introduction to EE, 1 corresponds to applied mechanics and 2 corresponds to Calculus.

Now, the same relations that I had earlier are represented in a very similar fashion once again, using tuples. But it is much more compact and the chance of making a typo by typing in the wrong information has been reduced. So, in other words, this has brought in this notion of something called a key. So, what is the key over here, it is basically these values, 0, 1, 2 and so on.

All of these keys are just some unique identifiers, that help us to point to a given person or a course or some object in general and combinations of those keys can be used in order to represent relationships between different types of objects. This makes data entry errors less likely and of course, we do not have problems with duplicates, because keys can be made unique. Even if I had two people with the same name, I will give them different keys.

Now, how do we take this forward over here, as you can see, I still have you know, I have converted from a list to a dictionary, but it is essentially still a list of keys. Each of those keys corresponds to a value, a dictionary after all can be thought of as a list of a linked list of keys and each of those keys essentially corresponds to some value, which I can look up directly, linked lists to or array, list in general, a set of keys. So ultimately, all this sort of boils down to a bag of data, I mean, I have some number of keys, each of those keys corresponds to a value I have that are stored somewhere.

(Refer Slide Time: 05:10)



Objects

```
class Student:
    idnext = 0 # Class variable
    def __init__(self, name):
        self.name = name
        self.id = Student.idnext
        Student.idnext = Student.idnext + 1
```

- Auto-initialize ID to ensure unique
- Functions to set/get values

If I wanted to give it a little bit of structure, one thing that I could use, making yourself Python once again, is maybe define a new class and one thing that I could do in the class is I basically say, I have an initialization function corresponding to the class and every time so for example, if I say that, a is equal to new student, a student and I have to give it a name, because the initializer, the constructor used for this class requires a name as input.

What does it do with that name? It just assigns it to the class variable, not the class variable, the instance variable over here, self.name is equal to name, which means that when I create a new student with a given name, that name gets assigned to the self.name or the object variable corresponding to that particular instance.

Now, one possible trick that you could use over here is that I could do something like this, the id, I do not take it as coming in from outside, the identifier. So, what do I do instead, I maintain something out here, which is a class variable, I have given idnext equal to 0. Now, the way that this is written, essentially what it means is that idnext is a property of the class itself, not of any instance of that class.

So, idnext is not a property of a given student, idnext is a property of the entire class student and what that means is that if I, when I create one student, if I give self.id=idnext and then do idnext=idnext+1 and then when I create another student, this idnext, which belongs to the class would have got the value 1 and the next second student that I create will get the self.id =1, third will get the value 2, fourth will get the value 3 and so on.

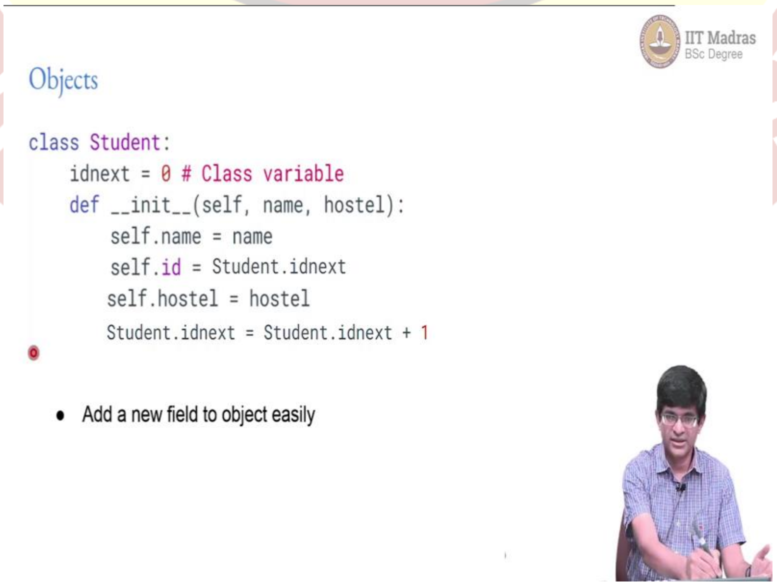
This is just one way by which you could do it, there might be other ways, typically what is done is not like this, you do not try and initialise inside a class, initialise ids inside a class, what you would do is you allow the database, the underlying system that is going to do your storage for you to assign an id.

The reason for that is because that way, even under heavy load, when many different people are trying to interact with the system at the same time, maybe create multiple users all at once, even then the database can ensure that unique ids are created, which may not exactly happen if you have multiple different people running the same programme in parallel.

So, this auto initialization of the id is one possibility, this what I have indicated over here is one trick or one way by which you could do it, not necessarily the best or even the preferred way to do it. Now, the interesting thing as what this also means, once I have defined a class like this, is that I can also define functions to set and get values.

What do I mean by that, I could now define, you know, maybe some function which assigns a photograph corresponding to the user and saves that in a variable corresponding to that user somewhere and if I want to get, let us say, the full name of the user, I might have some way by which I can automatically take a first name, a last name and a function which puts them together and returns the full name, even though I never entered the full name as a separate entity. So, all of those sort of fancier functions can be implemented neatly with this kind of class-based technique.

(Refer Slide Time: 08:41)



Objects

```
class Student:
    idnext = 0 # Class variable
    def __init__(self, name, hostel):
        self.name = name
        self.id = Student.idnext
        self.hostel = hostel
        Student.idnext = Student.idnext + 1
```

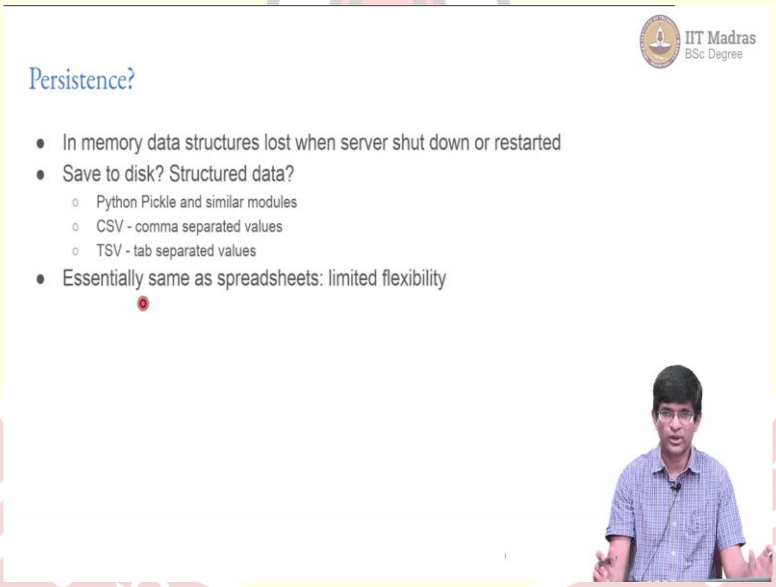
- Add a new field to object easily

The other interesting thing over here is let us say that I now introduce a new parameter. A new field that needs to be stored, I want to know in which hostel a given student is saying, what happens is I just make that one of the parameters to the constructor and I add one extra line out here.

Everything that worked before continues to work. All that now happens is the constructor, whenever it is executed from now onwards will expect a new value, the hostel and if it does not find the hostel basically in this case, since I have not given a default value, it will actually show an error.

Now, how will this deal with previously created users for whom the hostel was not specified? There is no unique answer to that, that would depend on the implementation you have to, you as the designer have to decide on how you want to handle such a situation. But the point is, having this kind of a class, like structure allows us to add new fields to the objects easily.

(Refer Slide Time: 09:44)



Persistence?

- In memory data structures lost when server shut down or restarted
- Save to disk? Structured data?
  - Python Pickle and similar modules
  - CSV - comma separated values
  - TSV - tab separated values
- Essentially same as spreadsheets: limited flexibility

Now, all very good. We could either use lists or dictionaries or classes in Python in order to store. In other words, those have data structures or data types that we could use internally in order to actually store the information that we need. The problem is all of memory structures are lost, either when the programme is killed, in fact or if the server is shut down or restarted.

How do you, well you can prevent that from happening. But at least how do you recover from something like that. Ideally, what I would like is in case I do have a situation where the server crashed, then when it restarts, it should be able to load back the last state it was in and there are multiple ways by which I can take data types or data structures in Python and store them on to disk.

Python has a module called pickle, it has other similar kinds of modules, basically revolving around the idea of serialising objects. So even an object like a dictionary, which has like keys values and could be like some slightly complicated data structure can be saved in such a way that it can be right back in exactly the same way that it was saved to disk.

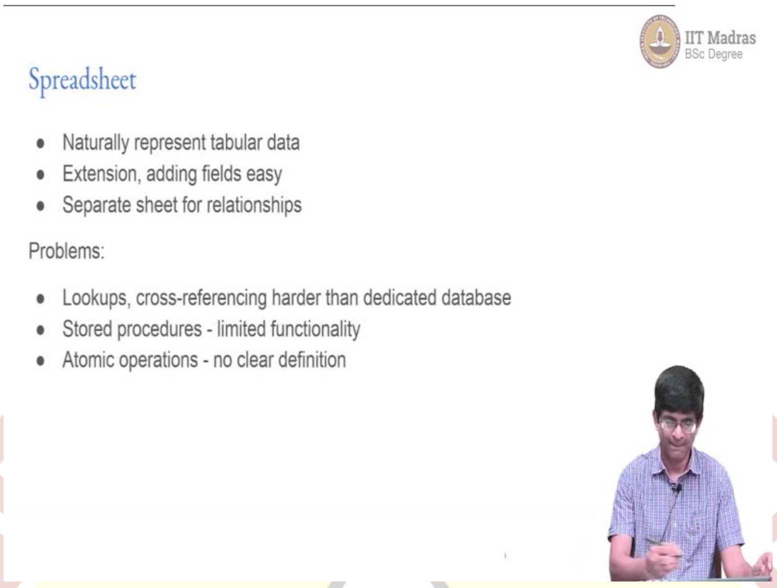
Other formats that are actually very common for storing tabular data after over all here we are looking at tabular data, are the so-called CSV and TSV, the Comma Separated and Tab Separated Value files. Both of these are sort of file formats, which are sufficiently well defined that you can use it for storing significant amounts of useful data.

There are some problems with CSV, which is why TSV is used. I mean, in particular, if you have a comma somewhere in the middle of your text handling that requires some special escape characters, which is not always standardised. But what this tells you is, if all I had was table data and I needed to store it somewhere, then yeah, maybe there is some reasonable way by which I can save it.

Now, if you think about it, essentially what it says is, every sheet in a spreadsheet, is a table, which means that every sheet over there could be saved to disk using this kind of format and in fact, you might notice that, spreadsheets can be exported in CSV format. The reason it says exported rather than saved is because CSV is a limited format, it has very limited flexibility, it does not allow you to sort of have links between different parts of a spreadsheet and lots of things sort of go away when you try to use a CSV or TSV kind of format, that could be stored in other ways if you had a better way of storing the spreadsheet.

But the point is, there are ways of storing spreadsheets, there are multiple, independently designed ways in fact. They cover most of our requirements, but as we will see later, there are cases where you have limited flexibility in terms of what you can do.

(Refer Slide Time: 12:44)



Spreadsheet

- Naturally represent tabular data
- Extension, adding fields easy
- Separate sheet for relationships

Problems:

- Lookups, cross-referencing harder than dedicated database
- Stored procedures - limited functionality
- Atomic operations - no clear definition

So, a spreadsheet in other words is naturally a very good choice for representing tabular data and extending it or adding new fields to a table for example, is very easy. I mean, it has such a large number of columns that you just add one more column, you give it a heading and you start entering data out there and spreadsheets sort of make it easy to sort of populate columns easily with data, copy paste, lots of things of that sort happened quite nicely. Now, as we saw in the case of the students and courses, we might need to use a separate sheet a separate table, in other words, which indicates relationships between other tables, but yeah, all this makes sense.

There are however, a few problems with using spreadsheets, one of them is that in general, doing sort of cross referencing or lookups between different sheets can be complicated in a spreadsheet, what I mean by that is, you have students in one sheet courses in another sheet students courses in a third sheet and sort of doing a cross referencing or finding out some information from the students courses sheet and trying to figure out the corresponding entry in the students sheet and so on.

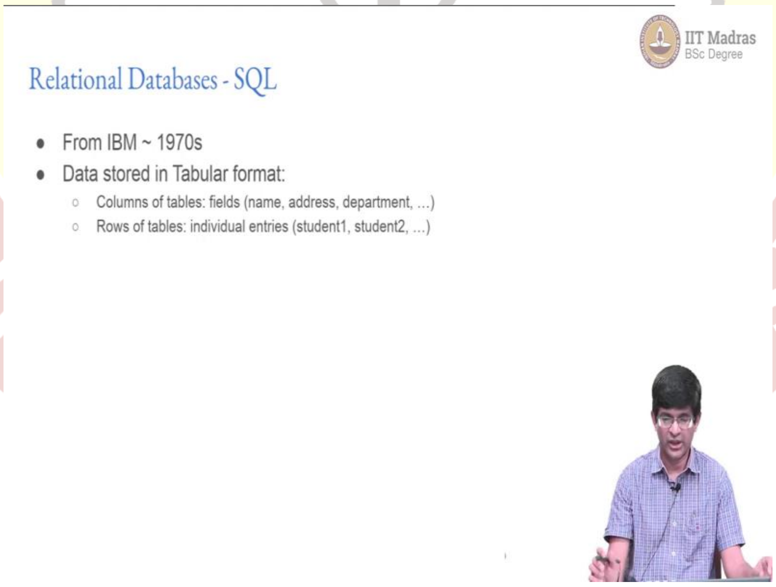
While it can be done, can be a little cumbersome or non-intuitive to write code for and the point is, there are better solutions, which is where dedicated databases come into the picture. There is also one further thing which you will probably look at or understand more the other, the next two problems with spreadsheets are things which you would understand better, when you do a full-blown course on database systems and even though as far as this particular course application development is concerned, we do not expect that you are an expert on databases.

Sort of basic working knowledge of databases is what is required over here, some of the concepts. So, for example, something like a stored procedure, which is there in a database, which makes certain kinds of activities easier to do is possible, but some-what difficult to implement in the context of a spreadsheet and most importantly, databases also have a concept of something called an atomic operation.

An atomic operation essentially says that a certain set of changes to the database are either guaranteed to completely happen or not happen at all. In other words, I can never end up with a situation where, let us say I am trying to delete one particular student and midway through while deleting some of the entries in the row, something else changed and I ended up deleting some other users information instead or I did not delete part of it from some other part of the database.

Spreadsheets do not really have any checks and balances for that. It is very difficult to code them in, if you can do it using various kinds of spreadsheets, scripting languages and so on. But if you actually tried that, you will probably end up with a database of your own. So, why reinvent the wheel, you are probably better off actually using something that is specialised for this purpose.

(Refer Slide Time: 15:53)



Relational Databases - SQL

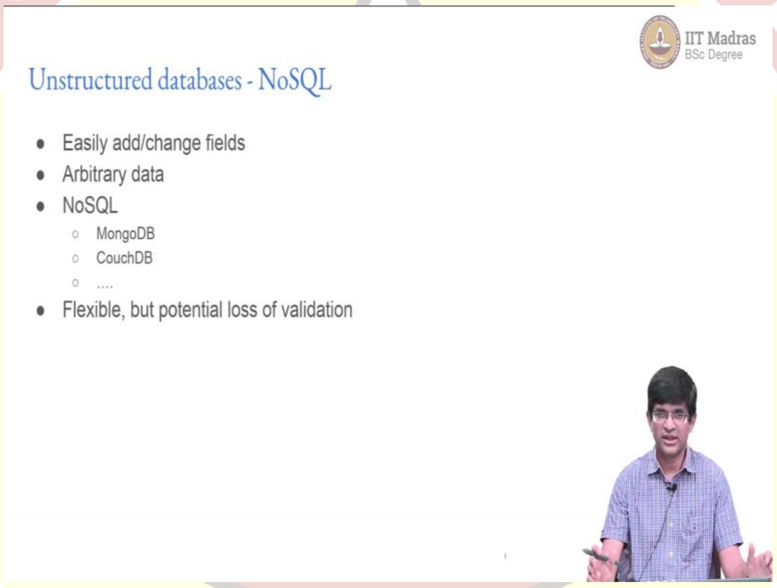
- From IBM ~ 1970s
- Data stored in Tabular format:
  - Columns of tables: fields (name, address, department, ...)
  - Rows of tables: individual entries (student1, student2, ...)

And that is essentially where relational databases and SQL, which is something you will probably hear a lot about as we move forward as well come into the picture. So, SQL is something which was sort of developed initially with at IBM in the 1970s. Primarily, it is, it was in the 1970s.

So, in some sense, it actually predates the idea of spreadsheets, data is just stored in tabular format, but it has a very sort of rigorous mathematical theory around it, which basically talks about how the data should be stored, what kind of keys should be used, how you can express relationships, and so on and it uses the basic idea exactly like we did in the spreadsheet, columns correspond to fields in a table and rows correspond to individual entries.

Now, SQL given the fact that in this course, we will primarily be building around SQL or some format of a database that uses SQL, we are, I will look at that a little bit more detail in at a later point. For now, I just want to mention that this is definitely something that is superior to spreadsheets, at least in the context that we are looking at.

(Refer Slide Time: 17:06)



IIT Madras
BSc Degree

Unstructured databases - NoSQL

- Easily add/change fields
- Arbitrary data
- NoSQL
  - MongoDB
  - CouchDB
  - ....
- Flexible, but potential loss of validation

Now, nowadays there is also another option, which is usually called the no SQL storage and rather than worrying about what no SQL means or why it is called that, a better way of thinking about it, is that we are now looking at sort of so-called unstructured databases. So, what is an unstructured database. So, what is the first thing, what is a structured database?

In a structured database, the schema of every table, that is to say, the set of columns that is present in each table is predetermined, fixed and is very clearly defined, every row in that table has to have the same set of entries. Now, it is potentially possible to think of a scenario where let us say, I have one student, for whom I need to store the hostel information. There is another student, let us say who is a day scholar. The third student who is actually maybe spending a semester abroad.

And although I could sort of think of one column, which sort of indicates their status hostel or a day scholar, non-resident or whatever it is, I might find that corresponding to each of these things, I have different entries that need to be stored. So, for a hosteller, I would need to have things like, what are the which mess do they go to which, what are their payment schedules for the different history.

If it is a day scholar, I might need to know what is their actual permanent address and their local address outside of the campus? For a non-resident person, it might be okay, which university are they in? What course are they doing, lots of additional information, which has no relevance to the other people.

So, when you start looking at it that way, you realise that even though SQL would allow you to just start creating columns and putting entries, so you could have one column, which basically talks about the mess facilities and other column corresponding to the local address and other corresponding to the foreign university that they are attending. But for a person who is in the hostel, the foreign University column is irrelevant and so is the local address and for a person who is actually on a semester abroad, their local address or their mess facility is again, irrelevant.

How do we handle this? So, this is essentially where the no SQL concept gain strength and the idea is that you can have semi structured or even unstructured data. What I mean by semi structured is, there is some common data to all students, their roll number and their name. But the rest of it could be just fields that exist for some users, but do not exist for other users and the no SQL database manages to efficiently store all of them without having to create columns for each and every user.

So, if it turns out that a student is spending a semester abroad, then I can go in and actually start querying that student's entry for the university at which they are spending time what the course translation is and so on and if they are not doing so and if they are still staying in a hostel, I might find that I have additional queries that I can use to find out, which mess they are going to answer.

So, obviously, this is a better fit for real life, because in practice, what we find is that, you know, if we start putting entries corresponding to what each and every different person can go through, our database is likely to become our tables that we need, can get too complicated. So, this adding or changing fields, in a relational database is a complex task because it affects the entire database, all entries. But in no SQL, you could have one field just added for one row in the database with very little performance impact on the rest of the system.

Arbitrary data can be stored, so I can actually sort of, attach binary data even as some entries for some people, rather than worrying about how exactly to expand the database and have it in regular structured sizes. There are several examples of this MongoDB is one of them, CouchDB, there are a whole lot, so I am not getting into the listing out names over here.

They provide a lot of flexibility, potentially at the cost of some validation, because after all, the structure in a structured database allows you to validate things are all entries, are all columns present. Does this column have data of a specific type and if it does not, it probably means there is something wrong with this particular row.

Of course, there is a reason why the NoSQL was brought in because you did not perhaps have that kind of structure to your data, but you need to know when to use structured databases and when to use NoSQL. Because NoSQL also has the drawback that because of the fact that it gives you this kind of flexibility, it comes at a cost, your searches or your access to the end to the database may not be as fast as in a structured database. There are of course, various kinds of tuning and so on that are done. But at the end of the day, the fact that is structured, does make it simpler to access certain elements in the database.