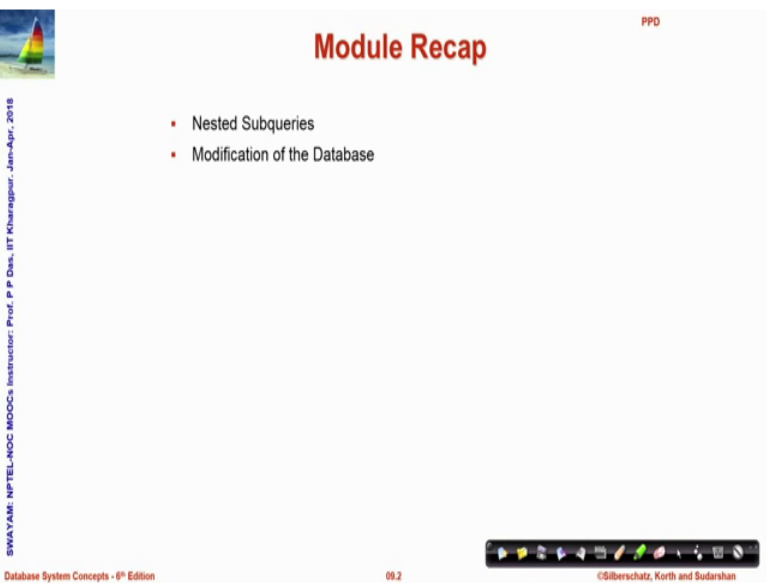
Database Management System
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 09
Intermediate SQL/1

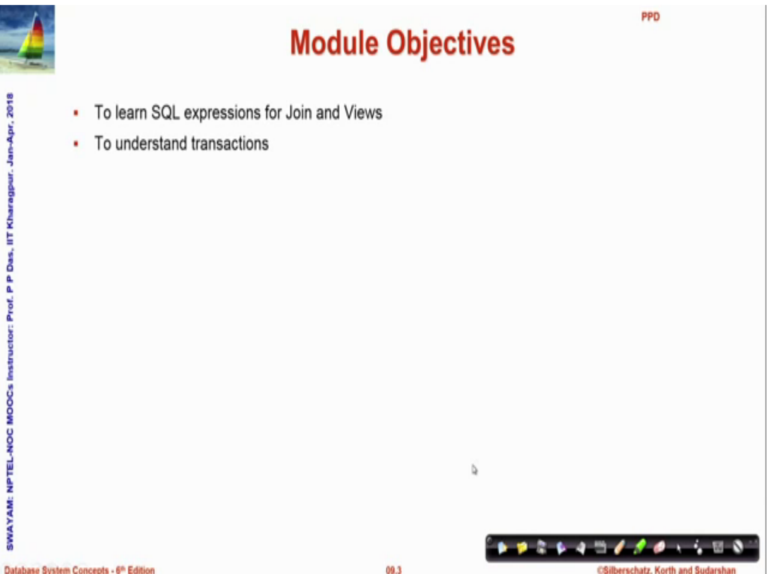
Welcome to module 9 of database management systems. We have discussed about the introductory level of SQL the structured query language in this module and the next we will take up some more intermediate level features of SQL. So, these modules are called intermediate SQL.

(Refer Slide Time: 00:42)



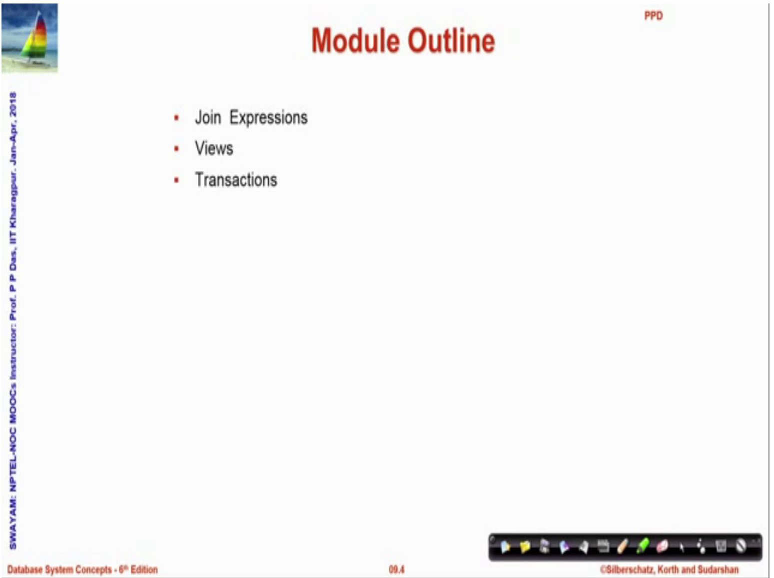
So, in the last module this is what we have done which was part of the closing part of the introductory SQL the nested sub queries and modifications to the database.

(Refer Slide Time: 00:55)



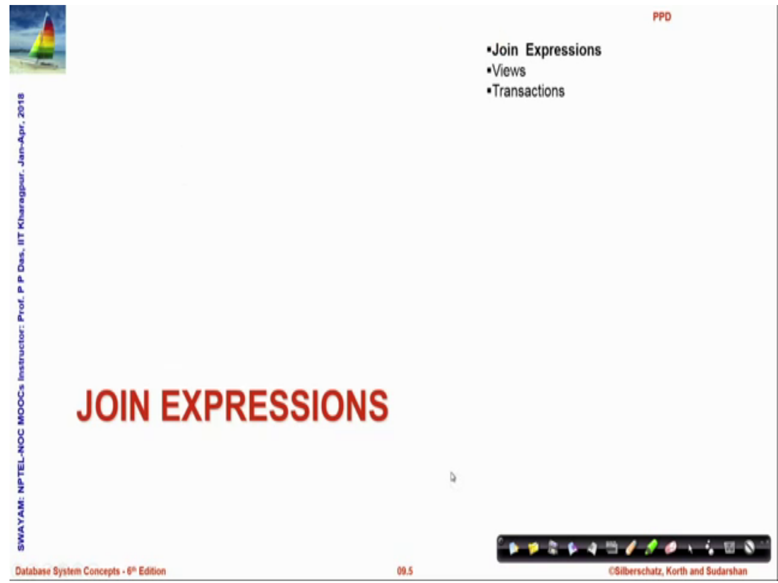
Today, we will, in this module learn about SQL expressions for join and views and we will take a quick look into understanding the transaction.

(Refer Slide Time: 01:11)



This is the module outline as it will span.

(Refer Slide Time: 01:15)



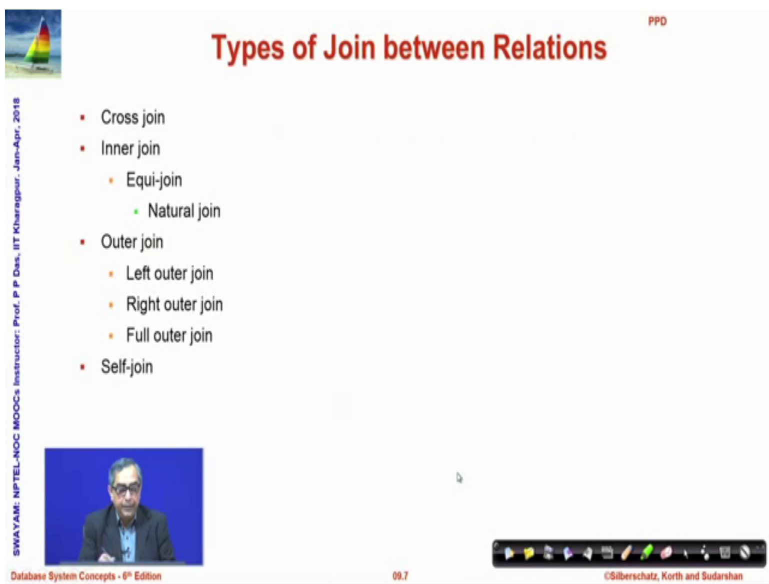
So, we start with the join expressions in SQL join as we have already introduced.

(Refer Slide Time: 01:21)

Takes two relations and returns a result as another relation. So, it is two different instances of two schemas and we try to connect them according to certain properties..

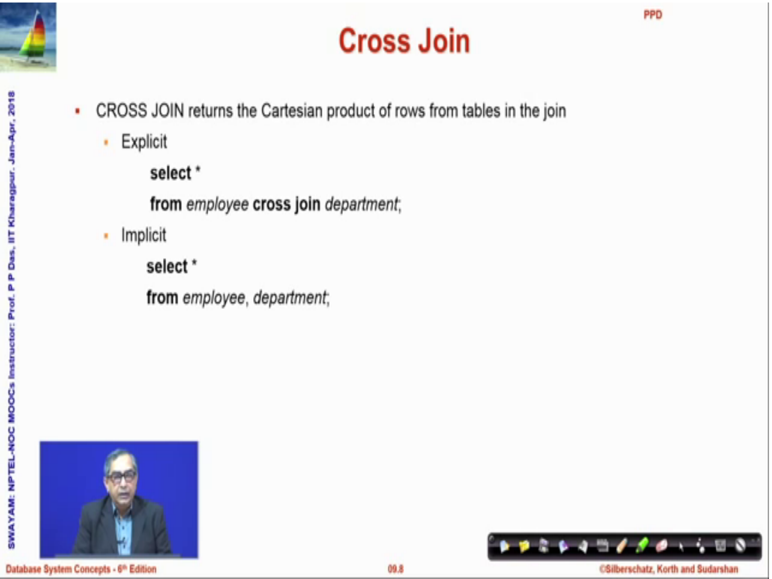
So, a join operation is primarily a Cartesian product, which relates tuples in two relations under certain conditions of match. It also specifies that after the joining has been done, what are the tuples, which will be present in the output join? So, join operation typically uses sub query, is used in a sub query, in the from clause we will see those uses later.

(Refer Slide Time: 02:19)



So, if we look into the different types of join that SQL support, these are the different classifications. So, we have cross join, we have inner join, which specifically could be equi join and even more specifically natural join and we will see there are variety of outer join, that are possible. There could be a self-join also, where one relation is joined with itself.

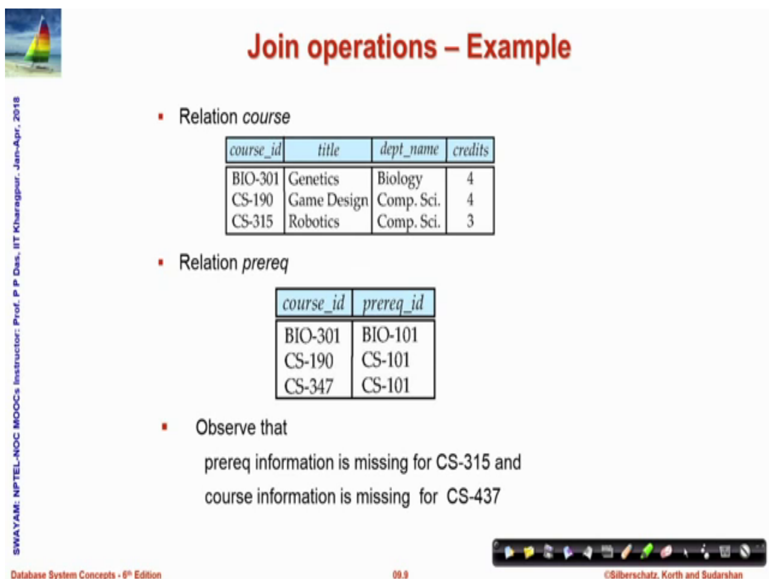
(Refer Slide Time: 02:51)



Cross join is just a formal join based, name for Cartesian product of two rows. So, you could explicitly do a cross join, which you can see here or you can implicitly also do a

cross join, but by specifying two or more relations in from clause, and taking all the attributes from there, we have seen these kind of Cartesian products earlier. So, cross join here is more as placeholder, in the context of the join semantics that pure Cartesian product is a cross joins, but what would be more interesting is, when we take different kinds of inner and outer joins.

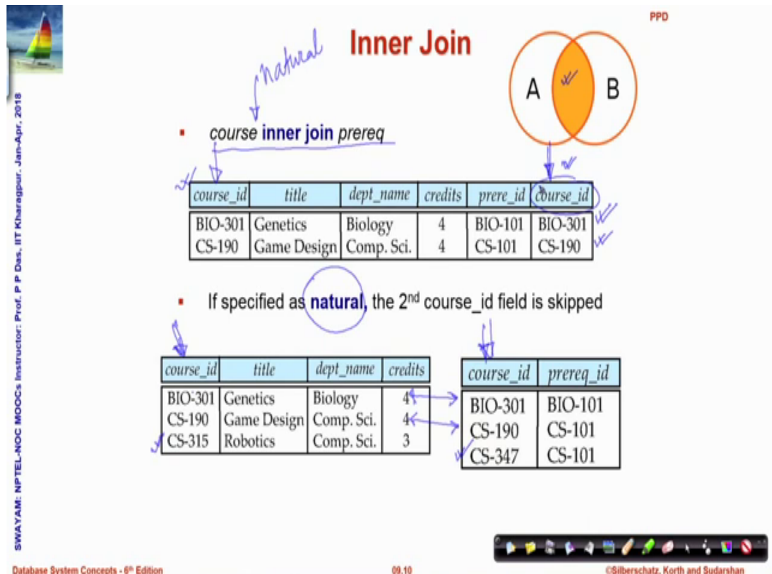
(Refer Slide Time: 03:30)



So, let us start with a simple example to understand the issues. So, there is a relation course, which has four attributes and in this particular instance, it has three tuples, three rows and there is another relation prereq, which specifies the prerequisite for every course.

So, it has two attributes; the course id and the corresponding prerequisite course id. It also has three rows, three tuples here, and if you look at the instances carefully, you will find that the three courses that are specified in the course relation all are not specified in the prereq relation. Bio 301 and C S 190 is present in prereq, but C S 315 is not present in at the same time, the prereq has one particular tuple, specifying the prerequisite of C S 347, which in turn is not present in the course relation. So, with this observation, let us start trying to see what different joint mean.

(Refer Slide Time: 04:54)



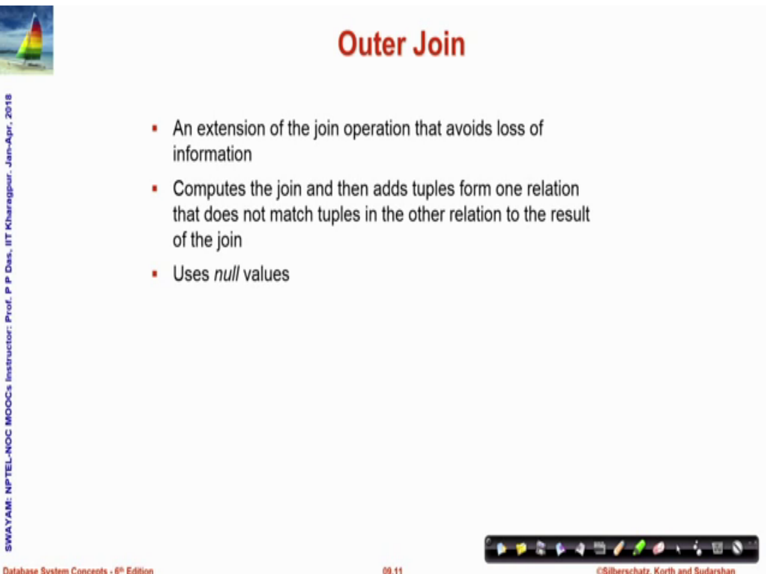
So, in a join is computed, then in terms of the two relations that we have, there is an attribute course id, which is common. So, once we have taken the cross product, we will from the cross product, only retain those rows, where the course id in relation course and the course id in relation prec prerequisite are same. So, when we do this particular record, when it gets mapped with this corresponding record, it will generate the corresponding output record. Similarly, C S 190, when it is mapped to the C S 190, in the prereq, it will generate the second record, we have already understood this, the third record in the courses C S 315 has no match here in prereq..

So, that will not feature in the output. Similarly, in the prereqs C S 347 that exist has no match in courses. So, that also will not appear in the output and also in the output you find that the course id has actually featured twice. This is the first column course id comes from course. So, it should more formally be called course dot course id whereas, the second one comes from prereq. So, it should that should formally be called prereq dot course id.

Now, if in addition to saying that, this is an inner join, if I also specify the word natural, I can say natural here, if say natural, then this second duplicate attribute course id will be dropped from the output that becomes a natural join, inner join as the name suggests, finds out the inner part of the two relations..

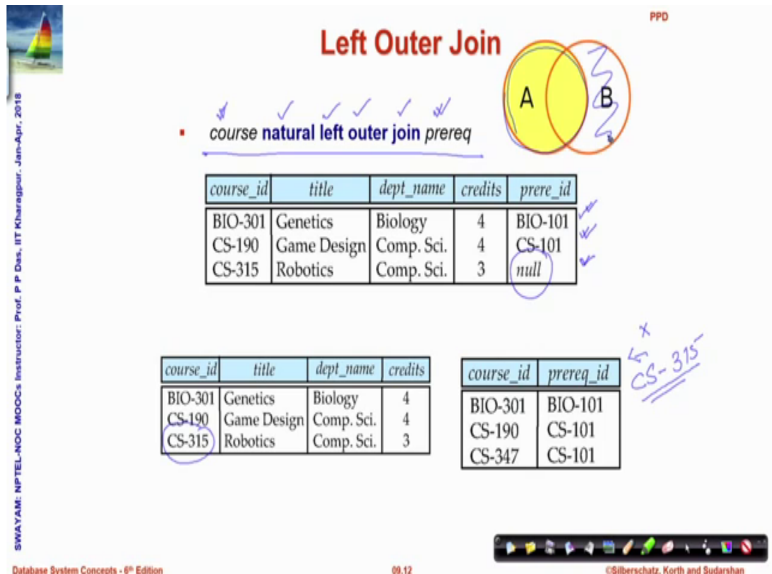
So, if we look at the two relations as A and B only those records, which are both have instance in A as well as B, in terms of equality of this course id attribute will come in the output. So, this is the basic type of join, inner join which is most commonly used.

(Refer Slide Time: 07:34)



Now, we can extend this into a different kind of join, known as outer join in the inner join. As you have seen that courses that exist in the course relation, but are not there in the prereq or the ones that exist in the prereq and is not there in the course are not featuring in the final inner join output. So, there is some loss of information in terms of this. So, why we are doing this we can compute and add tuples from one relation that may not match with the with any tuple in the other relation and if we want to do that then naturally for the other attributes of that tuple in the target relation we will not know the values. So, we will use null values this is the basic idea of outer join..

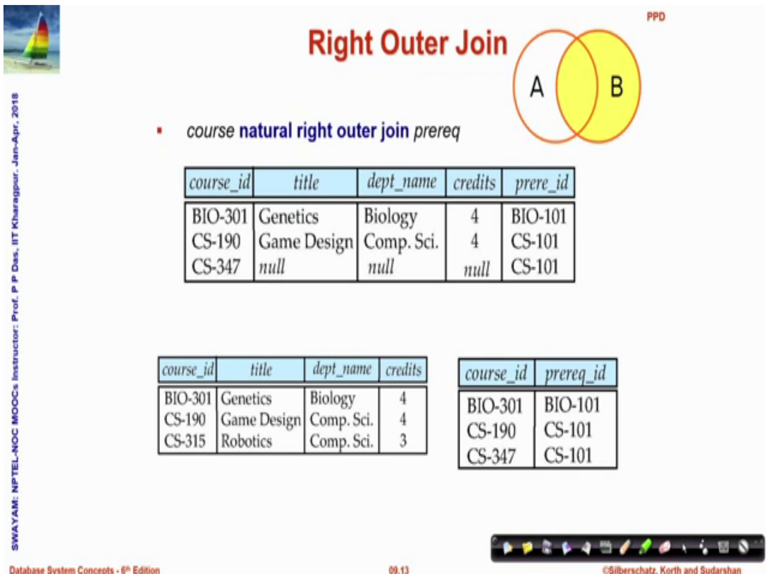
(Refer Slide Time: 08:34)



So, let us see what it specifically means? We first talked about left outer join, left in the sense that we have, this is how it is written. Left outer join is a sequence of commands that you give, you are also saying it is natural, which means that the common attribute will not feature twice in the output and this is the left relation and this is the right relation. So, left outer join specifies that in the output all records of the left relation, in this case the course relation must feature..

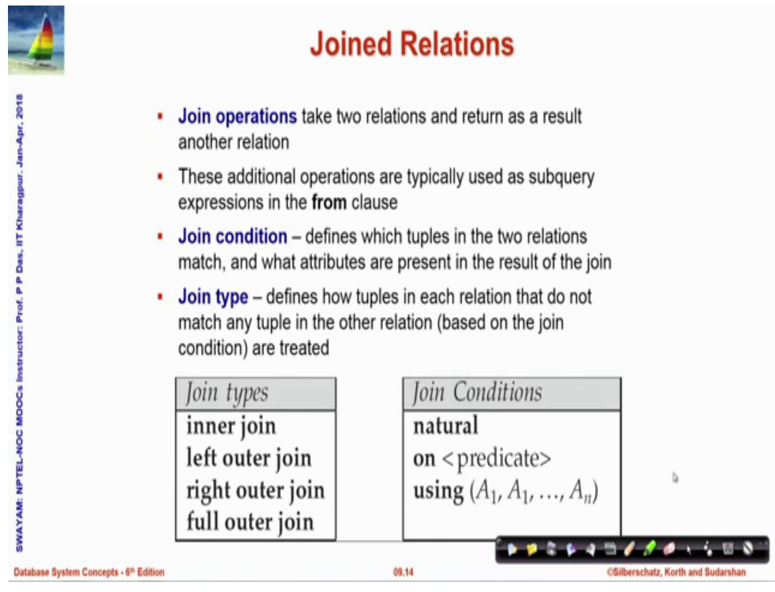
So, naturally when we do the join, we will get these two records as we have got in terms of the inner join, in terms of course 315 the C S, C S 315, the third course there is no instance in the prereq, we will still have that in the output, but since the prereq value for that, the prerequisite value is not known, the prerequisite id will be set to null here. So, left outer join ensure that, all relations of the left relation, all tuples of the left relation will necessarily feature in the output and that is a reason. If you see in the Venn diagram, the whole of this set, A is shown whereas, this part certainly will not feature..

(Refer Slide Time: 10:05)



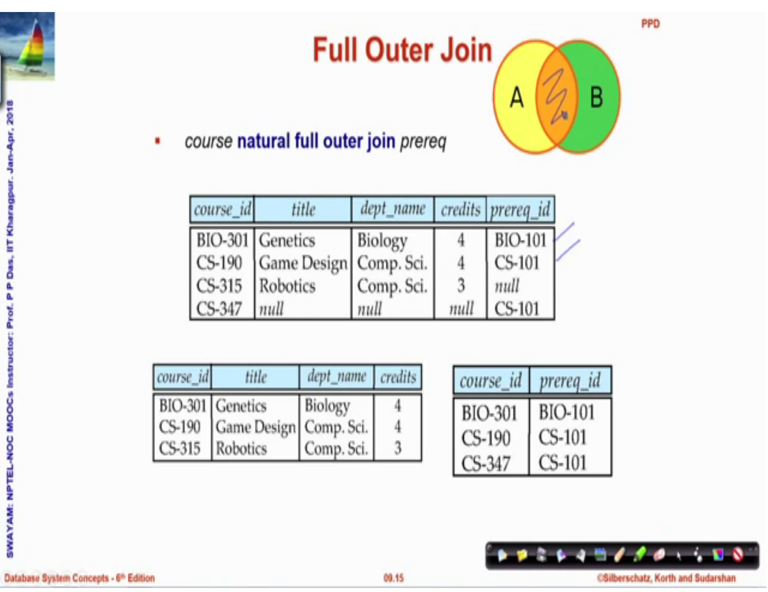
Now, similarly we can have a right outer join, where the concept is the same except that. Now, we ensure that all records of the right relation, in this case the prereq relation will feature and therefore, C S 2347 for which there is no entry in the course relation will also come as a record and since we do not know the title department name and credits for these fields, we will put them as null and this again is a natural one. So, course id is featuring only once. So, you will understand that since, we have a left version and we have a right version.

(Refer Slide Time: 10:47)



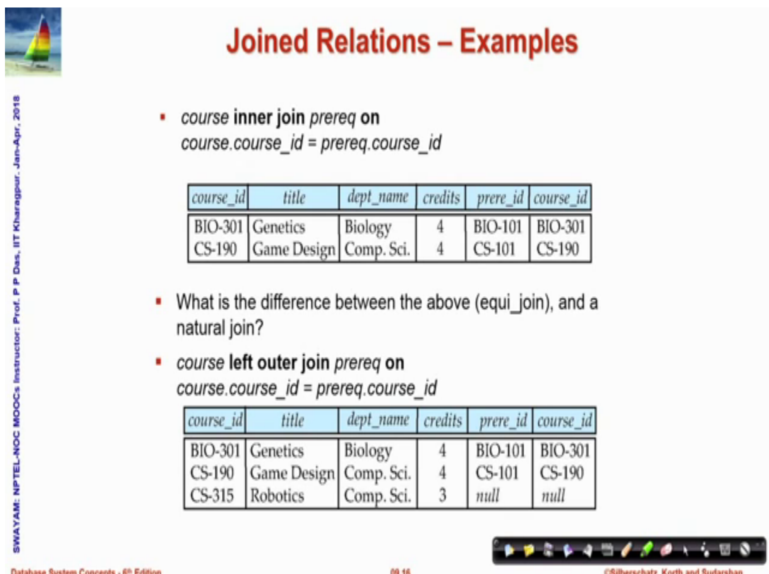
We can actually have a full version as well. So, if we look into the join relations in general, it takes two relations and returns a result and those additional operations are used in the sub query in from and there is a set of join conditions. So, these are the join conditions that we are specifying, whether it is natural and we will soon see that we can actually not depend only on the attributes that are common, we can actually specify that which attributes should be used in computing the joint. So, those are on condition and the using clause, we will just illustrate them soon and finally, there are four types of join that can happen, that is the inner join. We have seen the left outer join, right outer join and we will soon see the full outer join..

(Refer Slide Time: 11:52)



So, full outer join as you must have guessed will ensure that you get certainly the tuples from the inner join, which is here, you will get the tuple from the left outer join that is here, that is a tuple which exists in course and there is no corresponding matching tuple in the prereq and you will also get the tuple from the right outer join, that is for tuple, which exists in the prereq relation, but there is no corresponding tuple in the course relation and corresponding missing values are all set to null. So, these three kinds of outer join are possible.

(Refer Slide Time: 12:38)

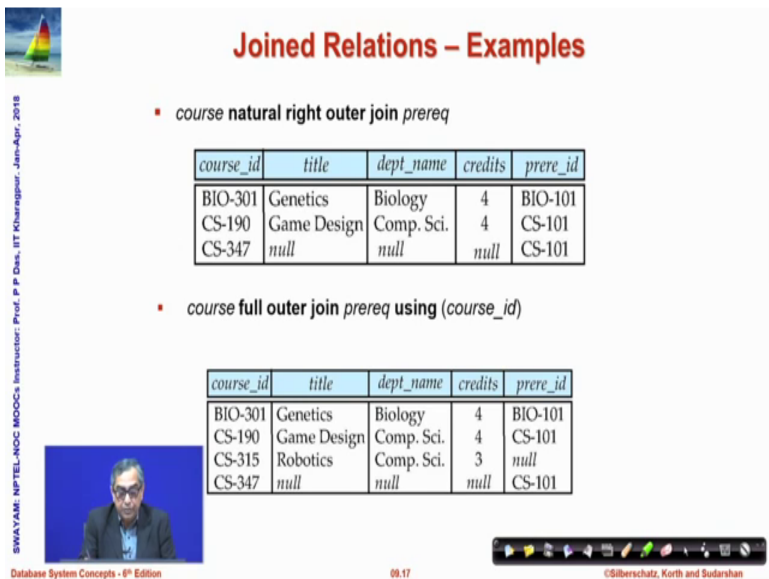


So, you can also specify join by saying that explicitly saying what attribute we want to join on and if you specify that, then you are saying it is a course inner join prereq. This part was same then you are putting an on clause, saying in the on clause, you will have to provide a predicate that is, which field should equate or match with what field. So, you are saying course dot course id is equal to prereq dot course id.

So, this result incidentally happens to be same as just doing the inner join, but we are illustrating that, on clause can explicitly use. For example, between the two relations, we have more than one common attribute, but we may want to actually do the inner join based on only one of them or equality on two of them and so on..

So, this kind of a join, where inner join, where you set two fields to be equal or two or more fields to be equal is also known as equi join and since we have not specified natural, you can again observe, then the course id attribute has occurred twice. If it was said natural then the second course id attribute would not have come in the result, this is a showing. The left outer join in terms of on clause and we have seen similar results. And now, this can be seen in terms of the on clause as well, and you can see in that second course id field. This entry is null, because actually you do not have that in the prerequisite set and; obviously, this set will be null, this field will be null..

(Refer Slide Time: 14:35)

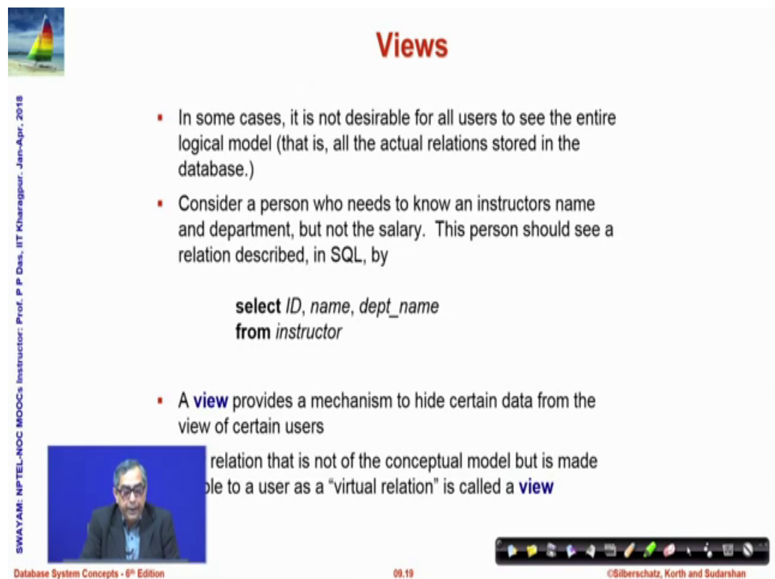


So, this is another example, showing you the natural right outer join, this is you, showing you full outer join and we are showing the use of the using clause. We can say using and put a set of attributes and the meaning is the join will be performed, based on those attributes. So, here in this case again the join will be based on course id.

(Refer Slide Time: 15:01)

So, that was about different kinds of join that we can do, which we going forward. We will see that form, a very critical has a very critical place, in terms of query formulation. Now, we take you to a different concept known as views now.

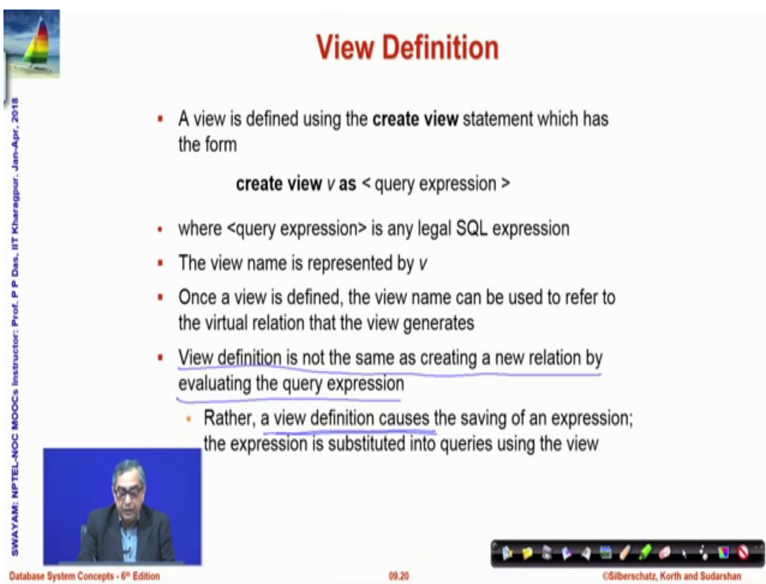
(Refer Slide Time: 15:20)



We have seen that. So, far we have been computing certain query results, based on one or more relations one or more instances. Now, in some cases, we may want the results to be restrictive in terms of based on the user or based on the context, in which the result should be used. So, we may not want all fields of a result to be visible to all the users or to the application. So, we may not expose the whole logical model and in those cases, we introduce a view. So, here we are showing one, where, from the instructor relation, we are only picking up three fields and we are not picking up the salary field..

Now, you would think that, well this is what we can do in terms of the normal query and certainly then, what is the point of using this? Now, what we can do is, we can create this not adjust as a query, but as a view one, once we create this as a view, it actually this query expression is treated as what is known as a view expression and every time you want to use that view. The actual tuples in that view are computed, but this is not actually a relation that exists in the database. So, it is a kind of, can be thought of as a kind of virtual relation, which exists, which can be seen only when you use that..

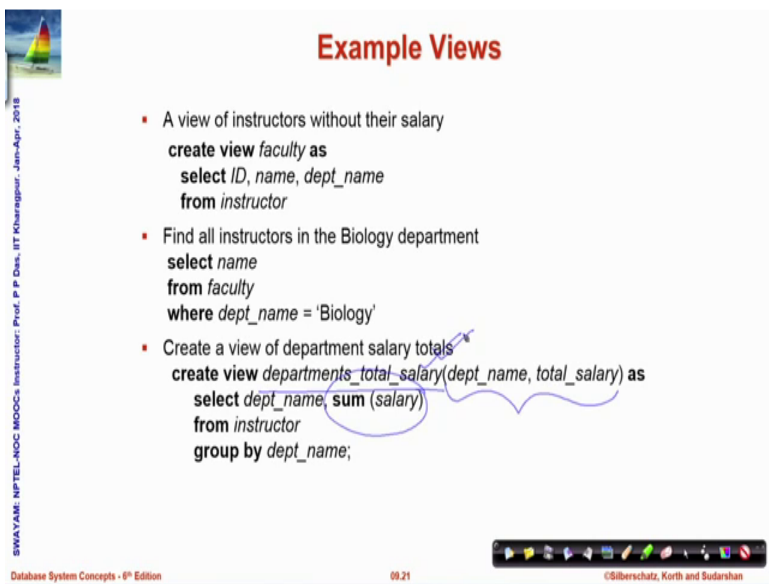
(Refer Slide Time: 17:22)



So, there is a subtle, but very strong difference between, actually computing a result, through a select query and defining a view, based on a select query and then making use of the view, as if it were actually a relation that existed. So, to do this, this is how we go about. It is the send text, is very similar to the create table. So, you do a create view give a name and then you specify as is the connective and specify the query expression, which is an SQL query, which will let you compute the view, every time you actually need it..

So, this is a view name, once a view is defined, the view name can be used as a virtual relation, it can be used exactly as we use any of the really existing relation, the conceptual relations that we have created, through create table. So, it is the difference, this is what needs to be understood very well, the view definition is not the same as creating a new relation, once you create the new relation, the time you have created it, you get the result and that result is explicitly available as a set of tuples as a table rather a view is a definition, which you stored in the database as an expression. So, every time you make use of that view, at that time the set of tuples are computed. It is not existing in the database as stored like the real relations and based on that computation, all the rest of the query will actually be executed..

(Refer Slide Time: 18:58)



So, let us take a quick look, this is a create view, we have created the view of a, of view called faculty, from instructor relation. Instructor relation is the real one, I existing one and faculty is a view expression being created and in that, what we have done? Simply, we have taken a done, a projection, we have lived left out the salary field. Now, we can make use of that view, you can see that we are doing from faculty..

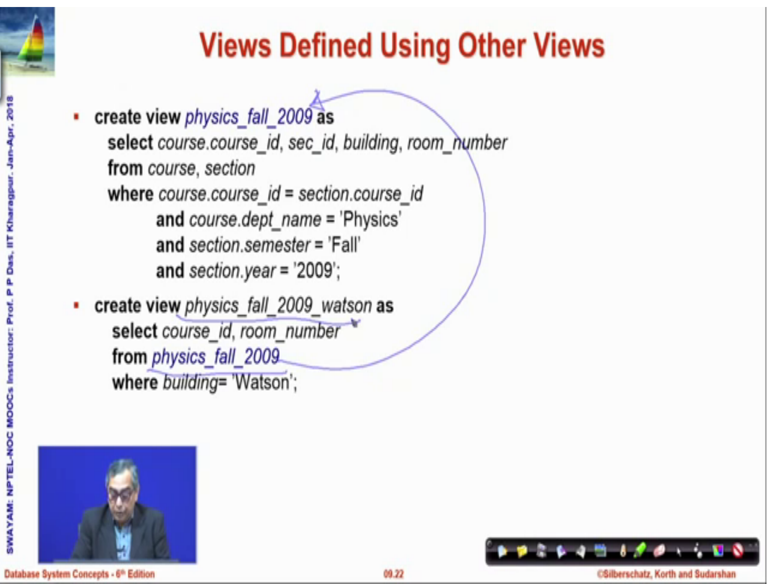
So, this actually is a view, but this behaves as if this is varying relation. So, from faculty we are trying to find out the name of all those faculty; who belong to the biology department. So, what will happen, when they want to execute this query? This will refer to this view. So, to execute this query, it will have to first execute this query, get the temporary virtual instance of the virtual relation, created and based on that, this query will be computed and the results will be given accordingly.

So, that is the basic purpose of the view that the whole thing, the whole view expression remains as an abstraction in the database and computed whenever it is used. So, this is showing you another view, which shows certain computed information. For example, we are creating a view for departmental total salary, which will show as two fields department name and total salary, which has been created by aggregation..

So, anytime we make use of this view in a from clause, we will get, we will feel as if such a relation really exists where the department name and the total salary of the instructors in that department are stored, but it really does not exist. It is computed

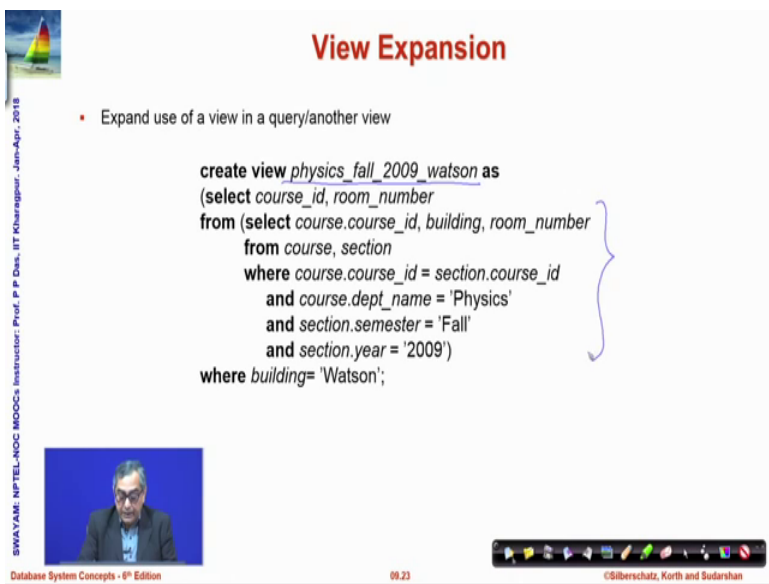
whenever it is needed, whenever it is used, you can actually use views to create other views.

(Refer Slide Time: 20:52)



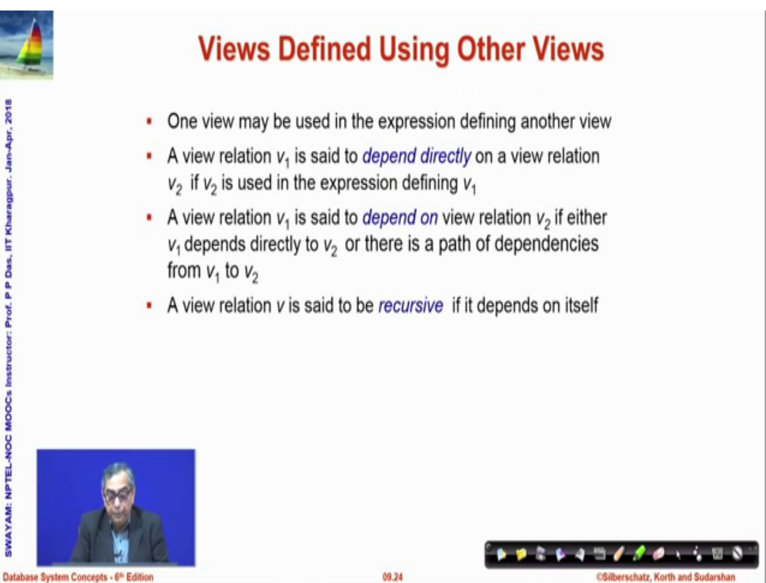
For example, this is one view which is the view of physics fall 2009, which are all courses that are offered in physics, from the physics department in the semester fall of year 2009 and using that we can create another view. See here again, we are in the, from clause we are using this view. So, creating this using this view, we are creating yet another view, which shows the courses that ran in the Watson building. So, views can be used as I have already said as any other.

(Refer Slide Time: 21:35)



Actual relation, but they do not really exist. So, if you expand out, if you just put the physics fall 2009 expression, within the view definition of physics for 2009 Watson, this is your earlier view relation. So, this is known as view expansion. So, this is actually the query that, you are executing and that has a lot of value.

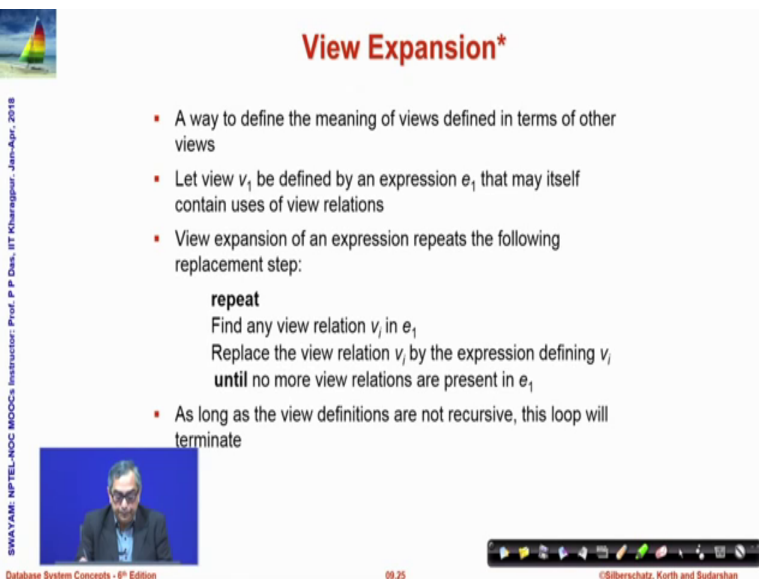
(Refer Slide Time: 22:04)



So, as we have said views can be defined indirectly from one relation. So, these are called direct dependence or they could be defined in terms of a chain of relations v one in

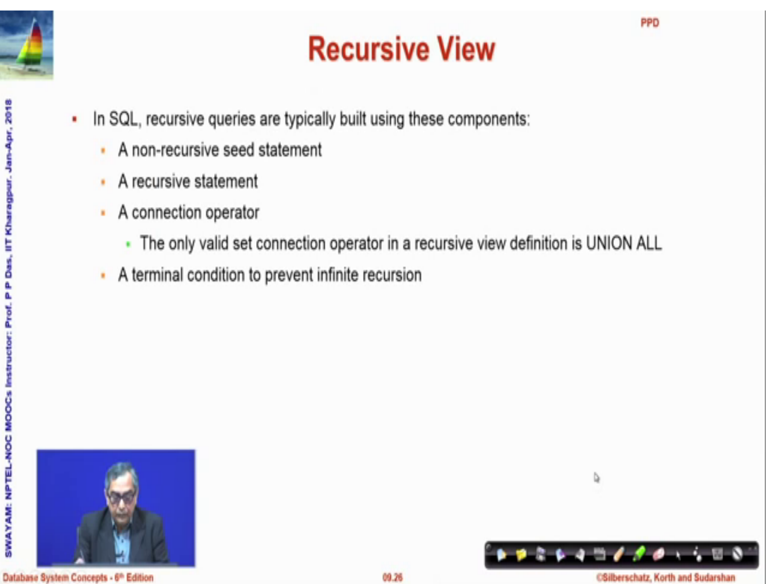
terms of v 2 v 2 in terms of v 3 and. So, on and a view relation can be recursive also that a view could be in terms of itself.

(Refer Slide Time: 22:29)



A lot of power view expansion is the process that SQL uses to evaluate a view. So, I would request you to study this and understand that this process works. This is pretty much like pseudo code C program..

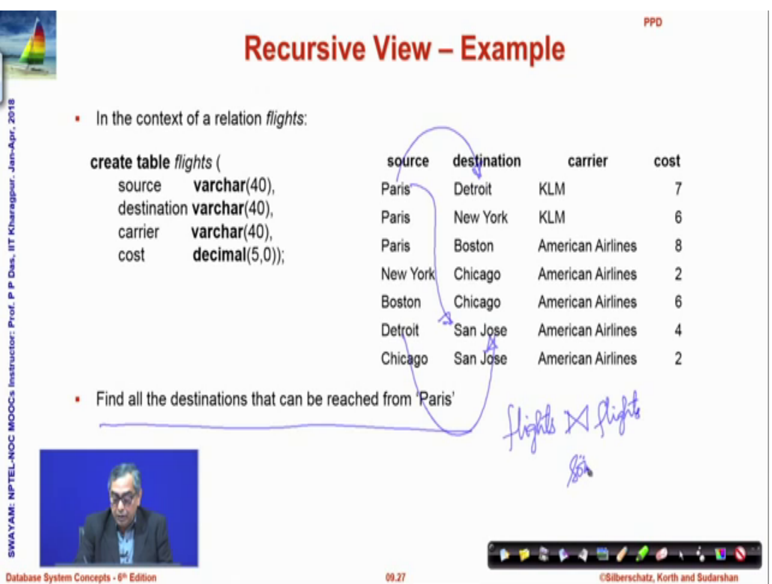
(Refer Slide Time: 22:47)



Now, moving to recursive views, the views where the same relation can be used in the view to define another view, we need like every other recursive structure. We need first a

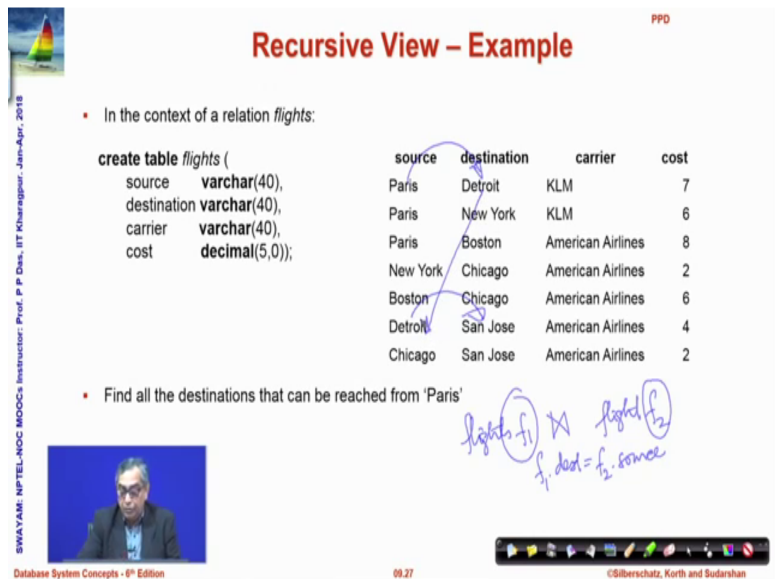
non-recursive statement, which is called the seed statement. we need a recursive statement, which can recur, we need a connection operator, which can connect the non-recursive and the recursive results together put them together, the only connective that is valid is union, all that is multi set union and we also need some kind of a terminal condition to guarantee that the recursion really a terminus, it does not go on forever.

(Refer Slide Time: 23:36)



So, let us take an example. So, this is in context of a relation flights, where the four fields are as specified and there is an instance shown, which show that different source destination of different carriers, carrying people from one source to the other destination and what do you want to find is all destinations that can be reached from Paris. Now, you can see that from Paris, if I can reach Detroit and from Detroit I can reach San Jose, then I can actually reach San Jose from Paris. So, that is the basic reach ability. So, that will necessarily, if I want to compute that, then I will be able to compute this very easily by doing natural join of flights with flights provided, I take say source..

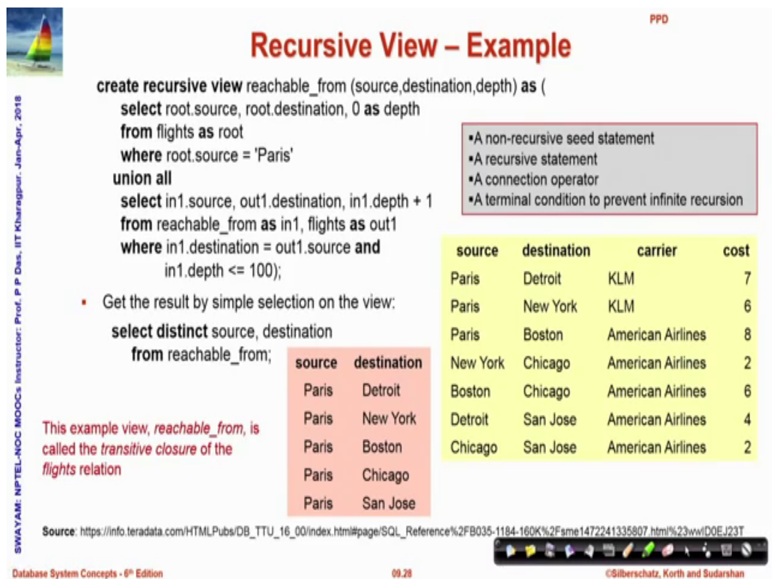
(Refer Slide Time: 24:38)



Let us compute it like this, flights f 1 join, flights f 2 and I will have f 1 dot destination equal to f 2 dot sources. So, the idea is if something goes from Paris to Detroit that is in f 1 and if some flight goes from Detroit to San Jose that is in f 2, then the destination in f 1 and the source in f 2 have to be equated. So, if we do this kind of a self equi join, then we will be able to find out all flights that go from Paris to San Jose or all places that you can reach from Paris in one hop.

Naturally, once you reach, once you do that then you may be able to go to another destination in two hops and once you do that then, you may be able to reach another, yet another destination in three hops and so on. So, we do not really know how many hops, maximum would be required to compute this reachability information. So, that is the reason we need to make use of the recursion and so, this is how we express it.

(Refer Slide Time: 25:54)



So, if we look into this, we are specifying that is a recursive view. It will happen with itself, this is the name and this is what we want to compute source destination and we take another dummy attribute kind of which specify the depth of recursion. So, the present instance of the relation is at depth 0.

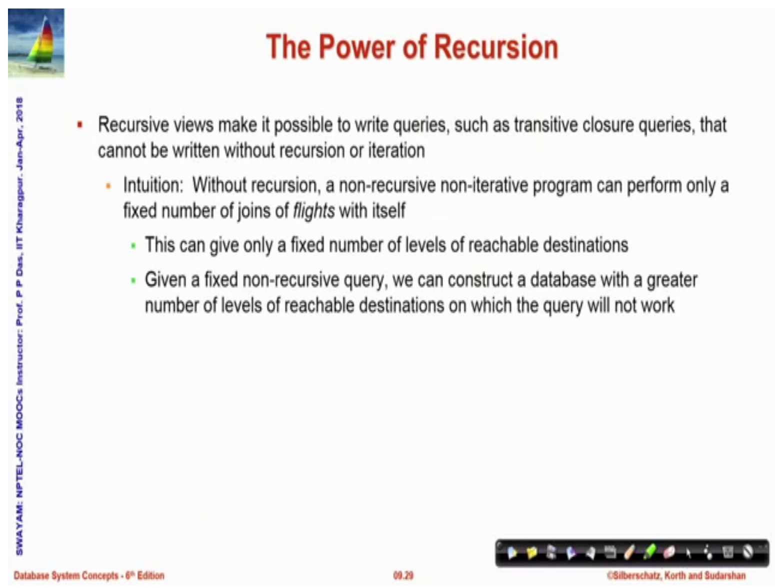
So, which defines your non recursive seat part so, it says select. So, you have renamed is at flights as route, you are specified that it has to start from Paris and you can find out the source destination pair at depth 0, then you specify the recursive part that is the second hop has to be defined. So, we are saying that, if you had the reachability then call, let us call it in one. This reachability maybe in one hop that is a depth 0 maybe in 2 hops that is a depth 1 maybe at 3 hops; that is a depth 2 and you take another instance of flight as out 1 and what you need is the destination in the first in one has to be same as the source in the other..

So, that they get connected and then you output the source from the first one destination from the second one and naturally the depth has got incremented, because you have done added one more hop and so, this is the and you add another condition saying that in one dot depth should be less than equal to 100. This is as I mentioned is a terminal condition which makes sure that, you do not get into infinite recursion. So, this view, recursive view cannot be used to compute any reachability, which has more than 101 hops.

So, that is to be noted and finally, we need to connect these two results, which is the initial start seed and the recursive one. So, this is the connection operator.

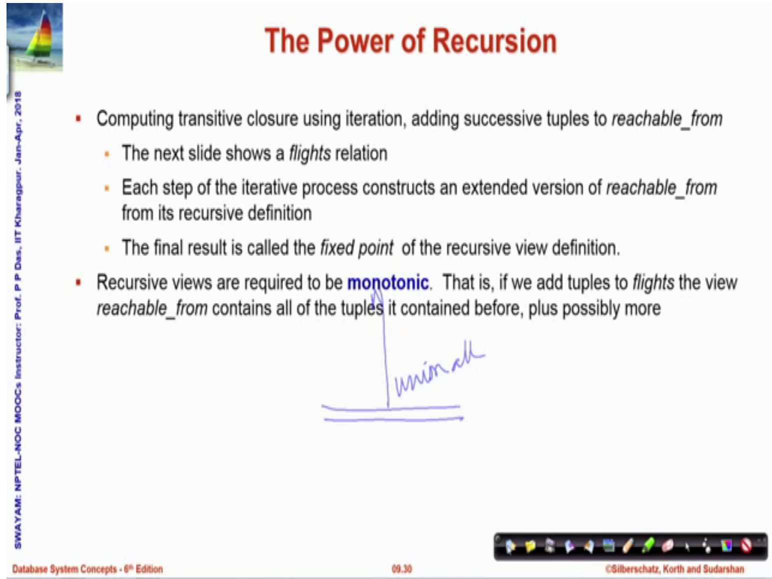
So, this is basically the idea of the recursive view, those of you who are more familiar with discrete structure, would have known among relations in some more depth, you would know that, we can define a transitive closure of a binary relation. So, this recursive view is necessarily computing the transitive closure from the flights relation. So, this is the instance of the flights and on the final computation, this is what you get. This gives you all the destinations that can eventually be reached from the source Paris.

(Refer Slide Time: 28:41)



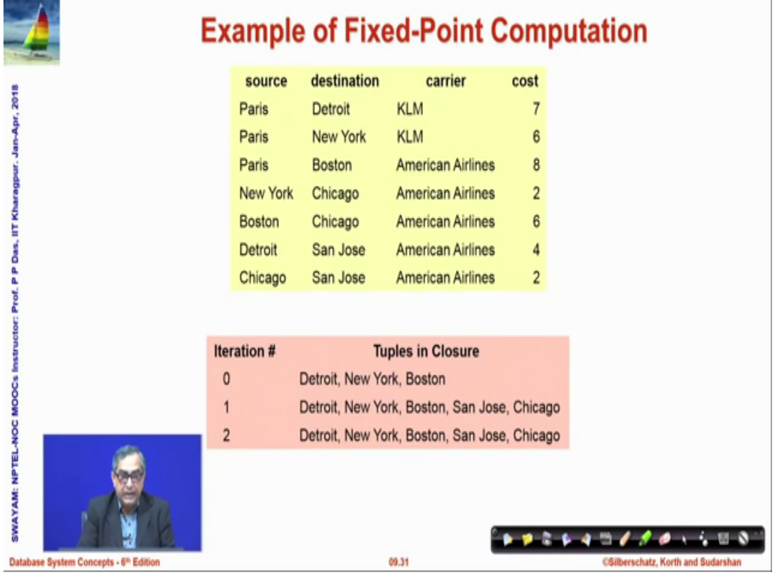
So, the recursive is very powerful in the sense that without recursion a non-recursive version can only find flights up to a certain number of hops and whatever ma query you write it is always possible to write out a database instance which will have more hops and your query will necessarily fail.

(Refer Slide Time: 29:05)



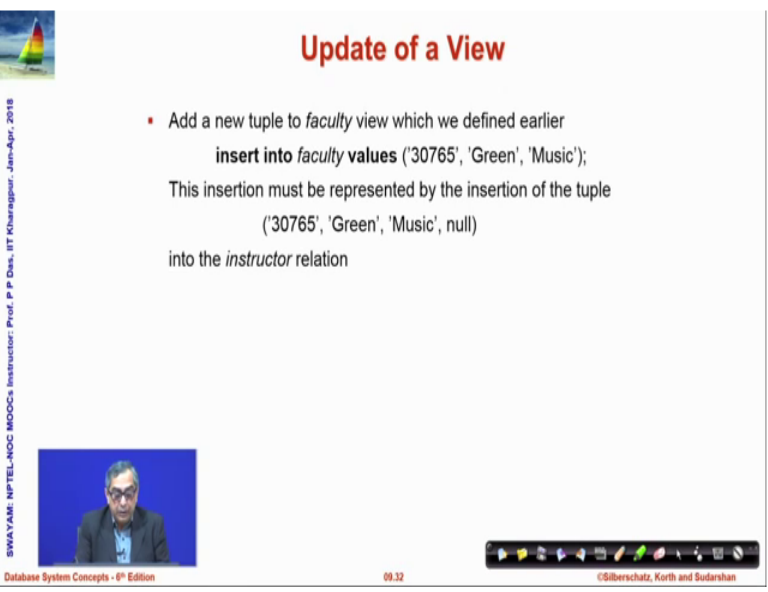
So, we make use of the recursion here to make sure that you can actually extend this to whatever depth you want and to compute this we keep on computing till no changes are possible and in that sense this recursive views are said to be monotonic in that every time you compute your result necessarily becomes larger and that is the reason you for the purpose of being monotonic you are actually making use of the union all. So, that makes it all inclusive

(Refer Slide Time: 29:49)



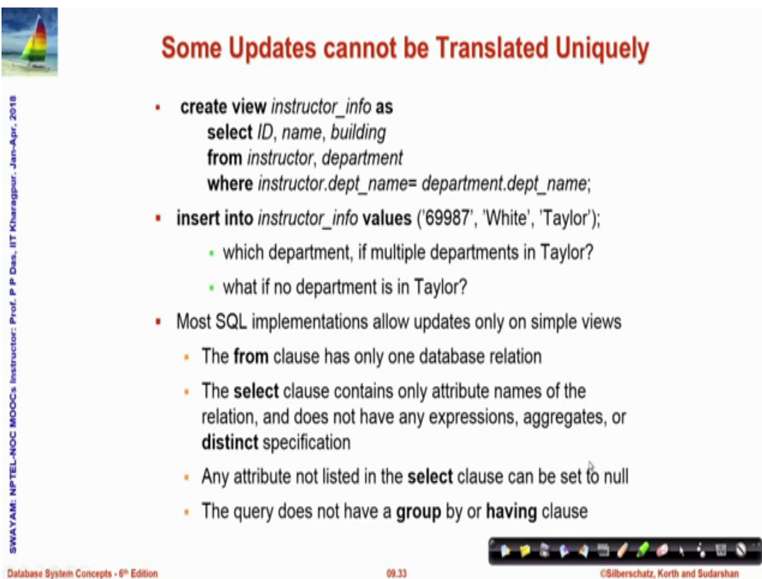
So, now, if we go and this is the instance and this you can. Here, I have shown that how the iteration actually happens in the iteration 0, in the flights itself, you had three destinations, then you add two more in iteration 1, in iteration 2, you do not add anything else. So, your result henceforth will not change. So, you have reached a fixed point and you have computations, are over.

(Refer Slide Time: 30:10)



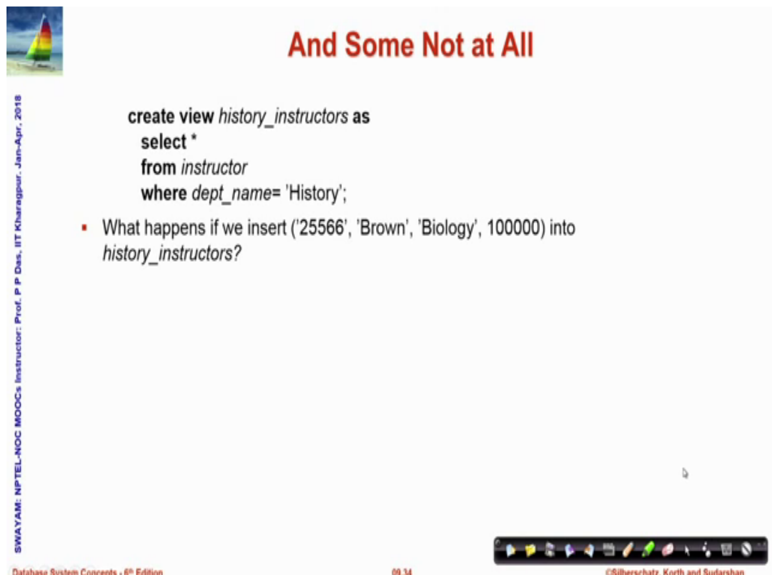
You can also update a view you can insert a record directly into a view, but since view only is partial information on the relation, when you insert into a view since, view is virtual. There will have to be an insertion, in the real relation and in the real relation you may not know certain fields. So, if you are doing this insertion into faculty, which is a view of instructor, then the salary field is not known. So, in the actual instructor a null will have to get inserted in the salary field. So, the salary field needs to be null able kind of field so updates on views have certain restrictions.

(Refer Slide Time: 30:47)

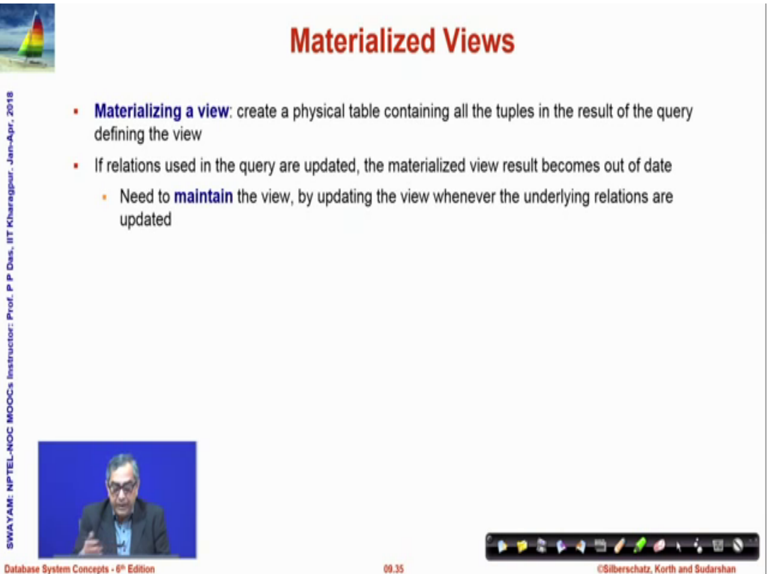


So, there are some more instances that I have given, which you can study and try to understand that what are the difficulties of updating on the view. So, it can be done, but it has to be done in a restrictive sense. So, these are the different conditions that has to happen for views to be updated.

(Refer Slide Time: 31:08)

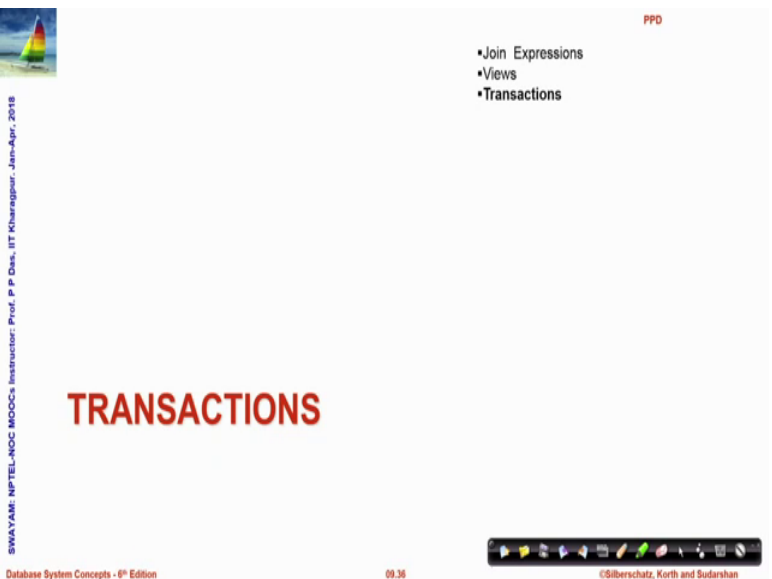


(Refer Slide Time: 31:11)

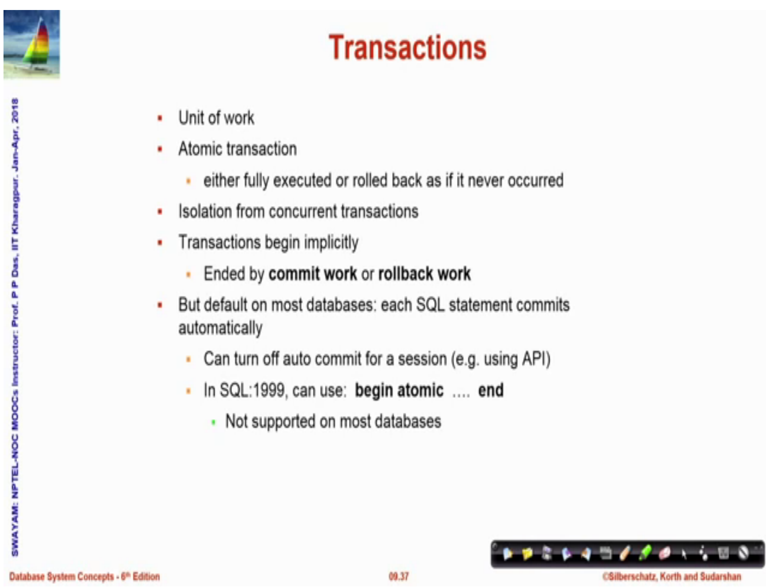


So, please go through these slides to understand what are there in terms of the views? Finally, view is a virtual relation, but it can be materialized also, that is materializing is basically computing a physical relation at the at the instance of the view, but naturally if you materialized then there is a certain point of time, where you have materialized where you have made it into a physical relation and hence, if your original source data in the view changes in future the materialized view also need to be updated otherwise, your data will get bad.

(Refer Slide Time: 31:43)



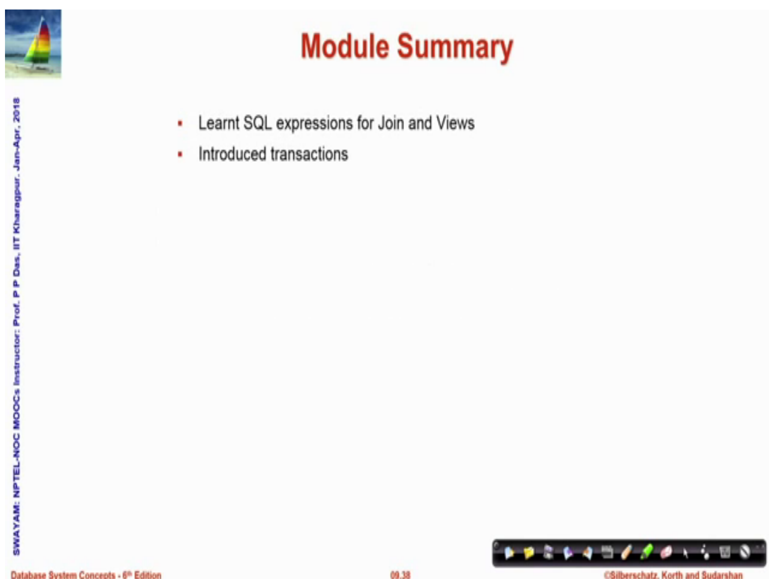
(Refer Slide Time: 31:49)



Finally, in this module, we mentioned that, there is something called transactions, which we will take up at a later stage, in much depth. This is just to get you familiar with atom a transaction is a unit of work, which is usually atomic, which is either fully executed or if it fails, it will be rolled back as if it never occurred and this is required for isolation in concurrent transactions. So, we will talk about this lot more when we take up concurrency and related issues..

So, come transactions implicitly begin and they end by either committing the work that they have successfully finished or rolling back that, this cannot be done. So, there are some features in the SQL for doing transactions and, but usually you can transactions, commit by default and the only it is exceptions, when the rollback is happening and we will see more of that later.

(Refer Slide Time: 32:49)



So, to summarize in this module, we have learnt about two important SQL features in terms of join and views and we just introduced the basic notion of committing transactions.