Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 37
Object Oriented Databases II

Hello and welcome. In the previous lecture we were looking into new kinds of data management problems where the kind of data that we have may not be easily amenable to reduction to a relational model. We were specifically looking at CAD databases that is databases in computer aided design where the UOD or the universe of discourse is better described by a collection of objects and some kind of relationships between objects rather than a pure relational schema.

(Refer Slide Time: 01:39)

And how is an object different from a tuple in a relational schema. Well, a tuple is part of an object but an object basically a tuple is an abstraction for particular structure of attributes, the way in which attributes are put together to form a tuple. But then an object is an abstraction for not only structure but also behavior. That is an object represents some kind of a complex data entity like say an electronic component like an IC or a transistor or something like that where it's not just the attribute or it's not just the structure that's important but also what kinds of behavior that the object performs.

I mean you can't expect a transistor to work as let us say something else I mean, I don't know, may be some kind of a current source in some other form. That is there is a specific set of behaviors that are associated with an object. And in conjunction with the attributes, it's also the behaviors that the objects provide an abstraction forum. And we also saw different notions that define object oriented database systems for example in an ODBMS it is mandatory or it is essential to have a unique identifier for each object, for

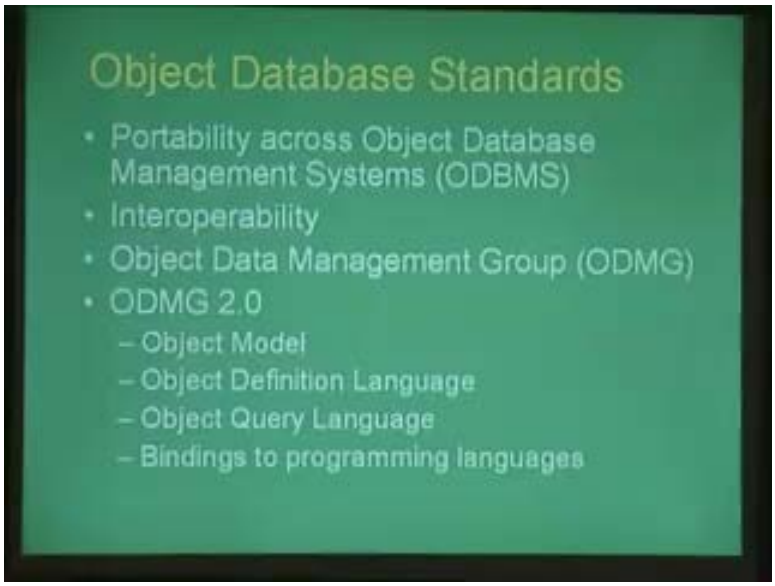
each persistent object in the database and this is exemplified by the OID relationship. And OID is contrasted from a primary key in a RDBMS by the fact that an OID need not be explicitly specified. That is the user need not even be aware that the database system is tracking each object using an OID. And just because the state of two objects are the same that is they have the same set of attributes and the same set of values for each attributes doesn't necessarily mean that they are the same object which is very much unlike the case in relational algebra were two tuples that are the same, represent the same data object essentially or the same data element. But here even if two or more objects have the same state, as long as their OIDs are different they represent different objects.

And then there are other issues like say inheritance and polymorphism and so on where or type hierarchies were an object can actually derive or specialize from a more general object and essentially as correct specialization is one where, wherever in the system you require an object of the general class it should be correct to substitute an object of the specialized class as well.

And there are other issues were the database system can be represented as a graph by associating different objects based on different relationships like containment and inheritance and other kinds of association. And associations are made using OID references that is object A associates with object B if the OID of B is an attribute of object A. And an ODBMS essentially is tightly integrated with OPL or an object oriented programming language where there are some kind of programmatic constructs that are provided to the language where the programmer who is writing the application program can identify certain objects to belong to the database in the sense that these objects can be declared persistent as soon as they are defined that is as soon as they are instantiated.

So as and when they are instantiated, they are also associated with an object in the database rather than just in the programming system, other objects which are not persistent are called transient objects. Now having looked into all those, we started looking into some of, we started getting little bit more concrete and started looking into the object database management group standard, there is the ODMG 2.0 standard for defining object databases.

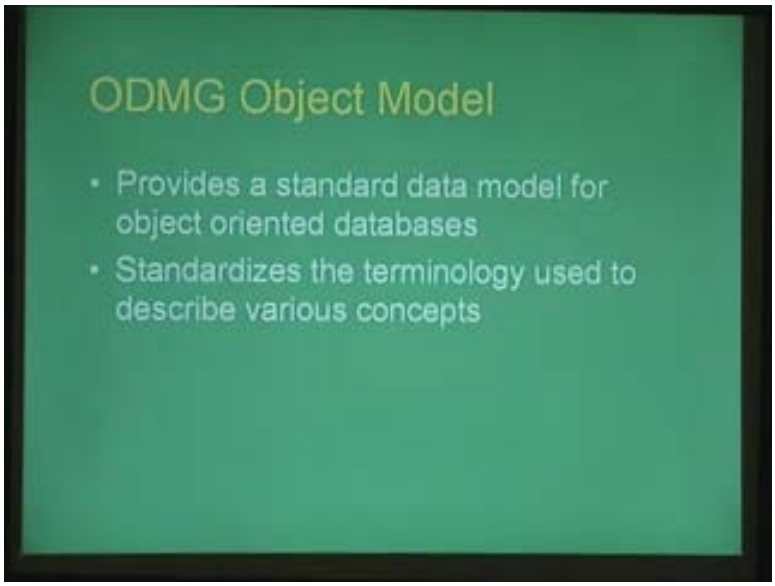
(Refer Slide Time: 07:01)



Let us look at this standard in a little bit more detail today or in this lecture and look at what other options does, does the ODMG standard provide. First of all let us look back at what is the ODMG standard which is slow shown in the slide here. The ODMG standard or the object data management group standard 2.0 model is a model essentially meant to standardize the notion of object databases. And the main idea or the main reasons for this include portability and interoperability between different object databases. And ODMG 2.0 standard defines an object model and a language for defining objects and querying objects.

So object definition language and object query language plus it also defines a number of bindings to existing programming languages like C plus plus and Smalltalk and so on where application programmers programming in any of these languages can directly interface with ODMG objects or in ODMG databases system. So essentially like we said before, the object model is meant for meant to be a standard so that the terminology is standardized and then which in turn help aids in interoperability and portability of database models.

(Refer Slide Time: 07:56)

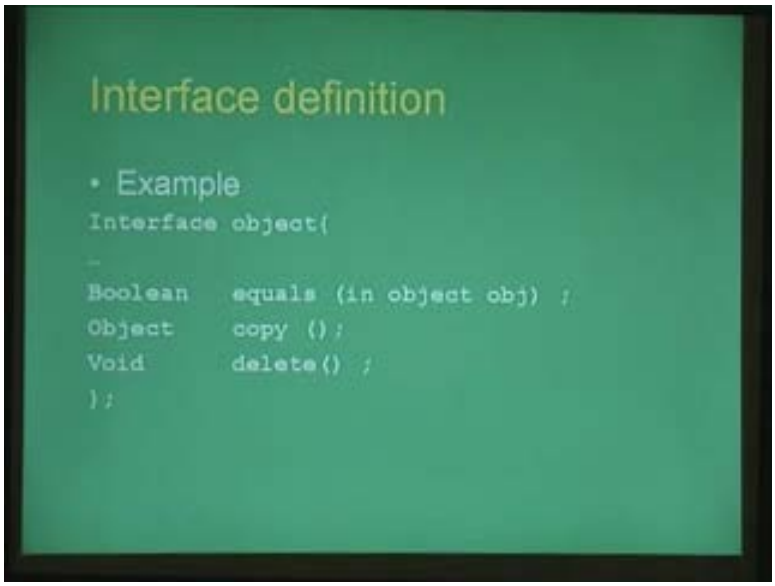


So in the ODMG 2.0 standard, objects are described by a set of different values of course an object has the well-known OID or the identifier and an object has a specific name that is associated with it plus also a life time. Life time in the sense you can either call an object as transient or persistent. So a persistent object is said to exist even after the software or the application program has finished its execution in a sense permanently and of course the structure of the object. And like we saw in the previous session, there are different kinds of data types that are usually supported by an ODBMS.

(Refer Slide Time: 08:34)

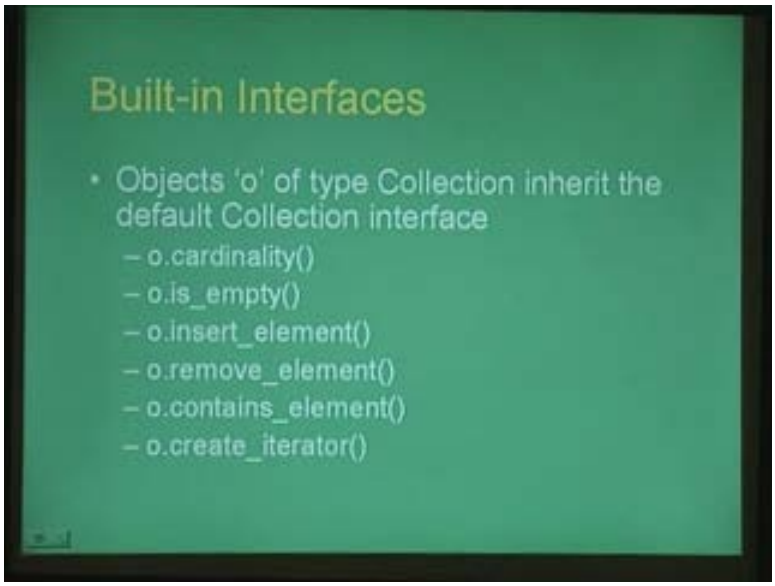
And the ODMG 2.0 standard supports different kinds of object types like, literal types like atomic attributes or atomic objects, collections and structures. This slide shows a typical interface definition of an object in an ODMG.

(Refer Slide Time: 09:22)



Remember that the interface is the signature for a persistent object. That is the interface tells the external world how to interact with the object and the interface for example shows here, the interface does not show type declarations. However it is showing the operation declarations like there is an operation called equals which takes in an object of type object and returns true or false and there is a method called copy and a method called delete and so on. And the ODMG 2.0 standard defines a set of or also proposes a predefined set of classes along with the class hierarchy.

(Refer Slide Time: 10:19)



And one of the classes in the class hierarchy is the class called collection. And as we saw again in the previous session at the top of the class hierarchy is a class called object. So every object in this, in an ODMG database by default belongs to the object class. That is even if you can't resolve in any other way which a class that a particular object belongs to, you can always say that an object of the ODMG database belongs to the root class which is the object class.

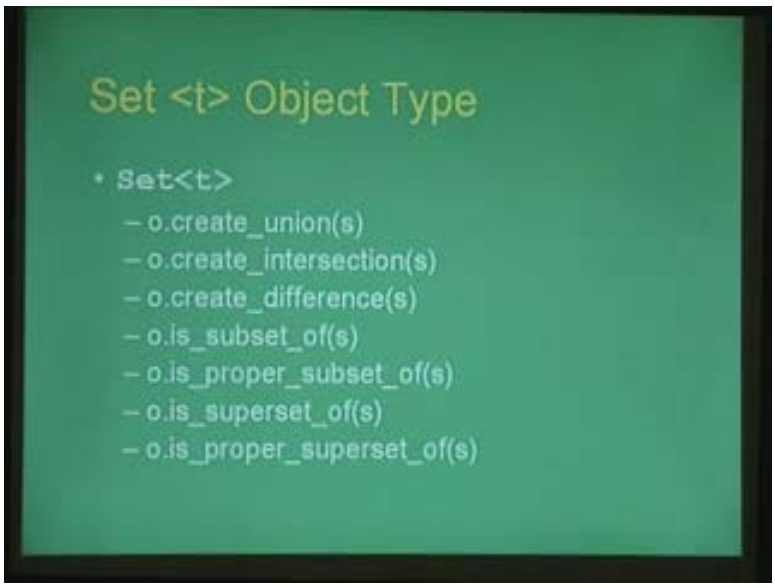
A collection is another derived class or a sub class of the object class which actually defines a collection of different objects that is a collection class can have a set of different objects or objects of the object class. And then the collection class also defines a number of methods like say cardinality of a collection, is_empty which checks for whether the collection is null or is empty then it supports insert_element where you can add an object into a collection, remove _element where you can remove an element from a collection, contains_element the searching element for a particular object and creating_iteratives. We will come to iterative soon, what exactly is meant by an iterator and what are its use.

If you have program with this standard template library in, say in C plus plus for example you probably have guessed what an iterator is where it's a kind of a template using which you can iterate over several different objects of any kind of a collection. There are other kinds of built in interfaces that are also specified by the ODMG standard and collection objects can also be further specialized into different kinds of these classes like say a set of collectors I mean set of collection or list or bag or array or dictionary and so on.

(Refer Slide Time: 12:25)

So, all these are derived from the collection interface collector collection interface that is the collection class interface. We now come to some iterator mechanism that is or rather the requirement for iterators. If you are familiar with C plus plus programming, you would have probably come across the notion of a template wherein a template is in some sense a definition with a hole or a method or a operator definition containing a hole where different other objects can come and fit into the hole or different other objects of other kinds of classes can come and fit into the hole.

(Refer Slide Time: 12:59)

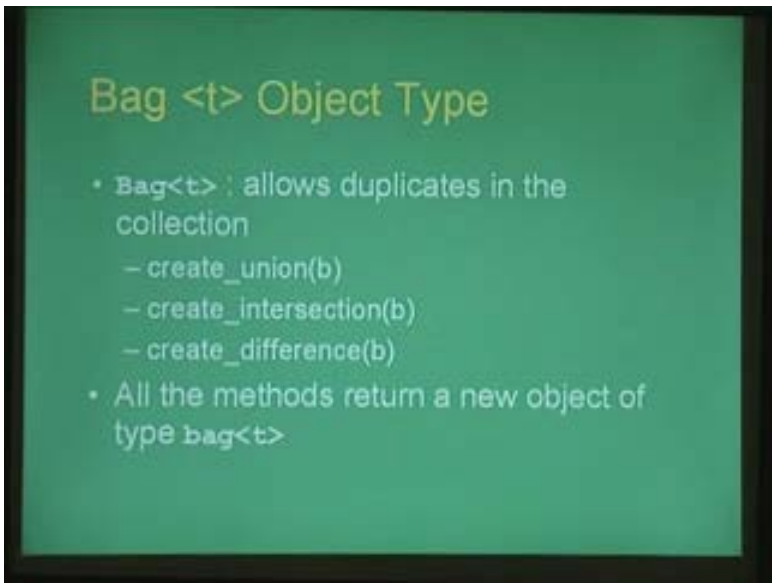


So for example if I define a template called list, I can have a template I can have a list of different objects of class a or objects of class b or any of any kinds of objects. So similarly the ODMG standard defines different template, the first of which that we are going to see is the set template. So set as shown in the slide here (Refer Slide Time: 13:55), set followed by a template mechanism here will create a set of classes of this particular type.

Note the subtle difference between a template like this and a collection. A collection by default is a collection of objects that is it just reads every entity in the every entity in the in the collection to be an instance of type object. However every class or every object in an ODBMS is definitely an instance of type object because object is the root or is the top most class definition in an ODMG database.

Now therefore a collection can be a collection of potentially objects of different classes at lower level. So you can have a collection comprising of one car plus one transistor plus one IC plus one truck which is very well valid. On the other hand a set here is a set of only a specific kind of objects, so you can say that it's a set of cars so you can't really you can't really include an object of class transistor in a car unless of course for some strange reason, transistor is a derived class of car. And the set template also defines different kinds of operations for over which are basically set operations like create_union or create_intersection, create_difference or subset_of or proper_subset_of, superset and proper_superset and so on. Similarly there is just like the set template, we have the bag template where bag is like a set as we have seen earlier that bag or a multi set is a set which can allow duplicates in the collection.

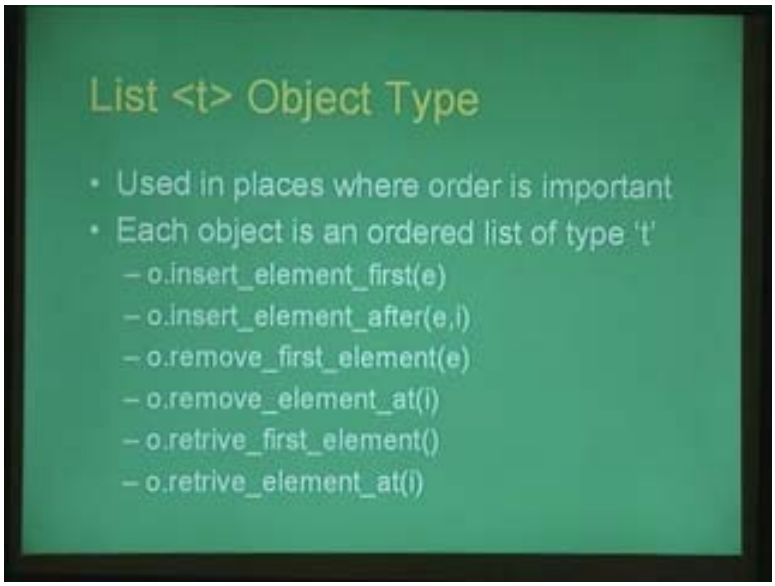
(Refer Slide Time: 15:56)



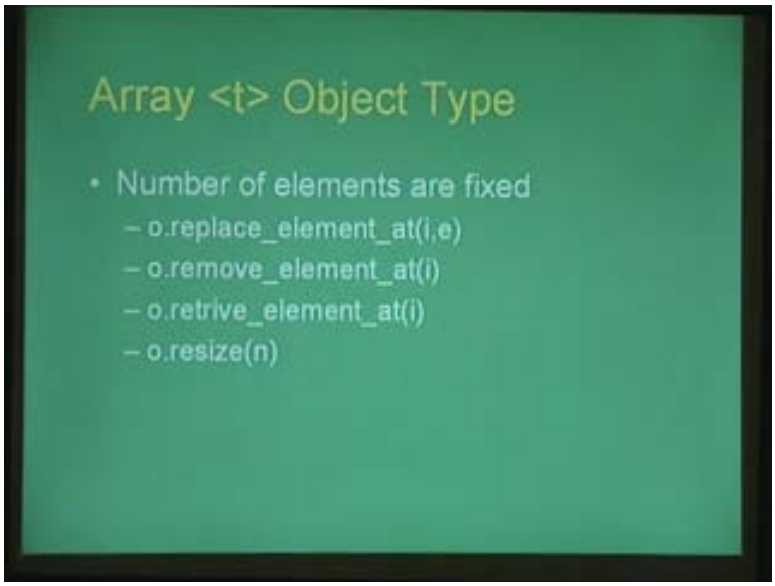
However duplicate should also be objects of the same type or of derived types which is specified in the template, it can't be arbitrary objects. And of course bag also specifies different kinds of methods like create_union and create_intersection, difference and so on. Again like set and bag, we have the list object type where unlike set and bag in a list, list is not just a collection of different objects, it's an ordered collection of different objects that means the ordering between objects is also important.

So you can insert an object at the head of a list or you can insert at the tail of a list, you can insert after an element, after a particular element in a list or you can remove the first element and you can remove a particular element at some particular level in the list or some particular location in the list, position in the list and so on and so forth.

(Refer Slide Time: 16:31)

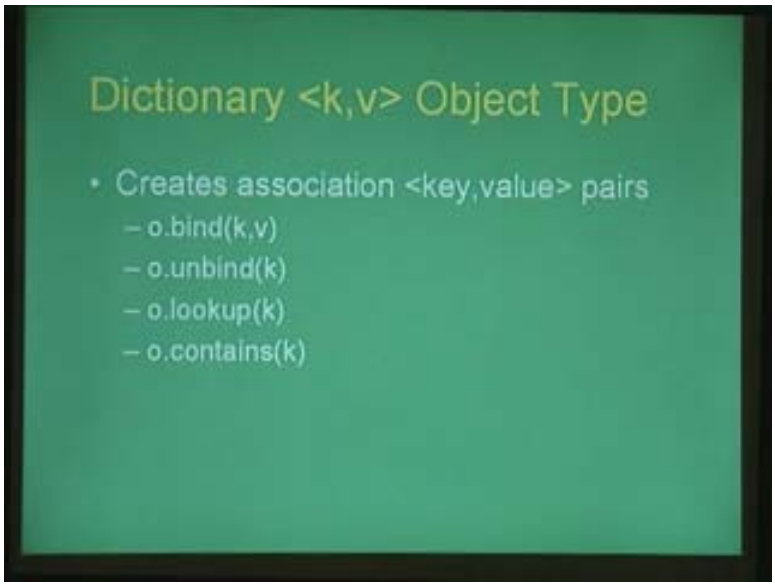


(Refer Slide Time: 17:24)



And of course the array object type which is again array is like a list where the order is important, it's a collection of or it's a set of different objects or of the same type. And where the order is important and usually the number of elements are fixed unlike in a list where the number of elements can vary. There is also a dictionary data type which is derived from the collection data type which is analogous to a hash table implementation. So it basically stores a collection of key value pairs and given a key, the dictionary returns a value and the dictionary object also defines a number of different methods like bind, unbind.

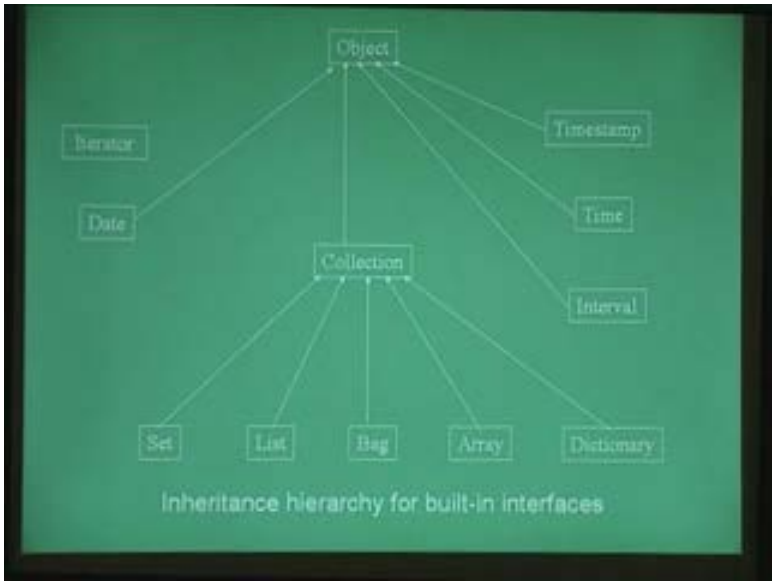
(Refer Slide Time: 17:42)



Bind essentially means that you are associating, you are inserting a value, you are inserting a new key value or rather you are binding a value v with key value k. And you can unbind in a sense delete the value associated with key value k and you can contain, I mean you can look up the value of a key k or you can check to see whether the key, the given key is actually present in the dictionary or not.

And this slide which we also saw in the previous session shows a typical, I mean shows the class hierarchy that is or rather it's more like the interface hierarchy that is the ODMG does not define a complete classes rather it just defines interfaces. And the definition of each methods is the responsibility of the programmer that is it can be the, definition can be written in any high level object oriented language like C plus plus or java or whatever depending on what support is available of course. But this defines an interface hierarchy that is any sub interface in a sense that is any interface somewhere in the hierarchy will inherit all the interface elements like the method interfaces of all interface above in its hierarchy.

(Refer Slide Time: 18:44)

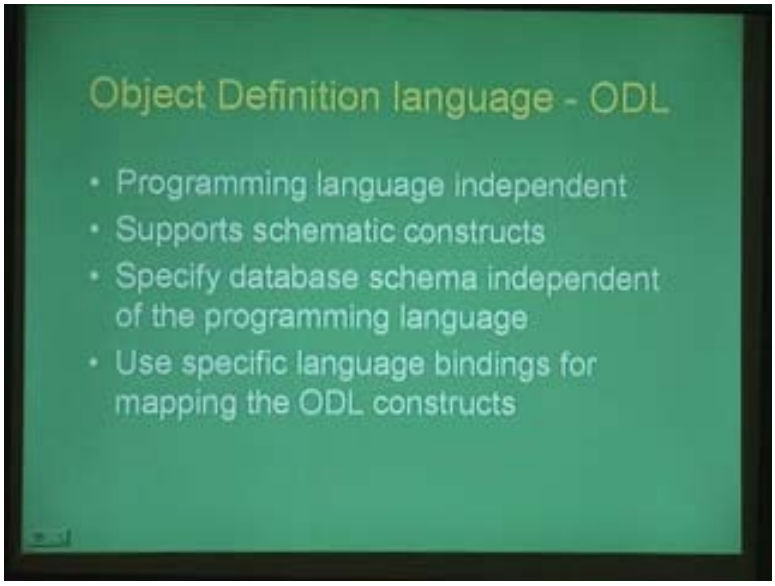


So as seen in this figure here (Refer Slide Time: 19:44), the object interface is the top most interface. So every object here is a kind of a, can cause a little bit of confusion in the sense that there is a class called object. So every object in a database belongs to a class called object by default and a collection is a collection of different objects and sets, lists, bags, arrays and dictionaries are different specific kinds of or kind of templates where they define specific sets or lists over specific types of objects. And then there are several other objects like, classes like timestamp and time and interval and iterator and date and so on.

The ODMG 2.0 standard also defines or also proposes an object definition language plus an object query language as we had mentioned earlier in this session. So the ODL or the object definition language that is defined by the ODMG 2.0 standard is a programming

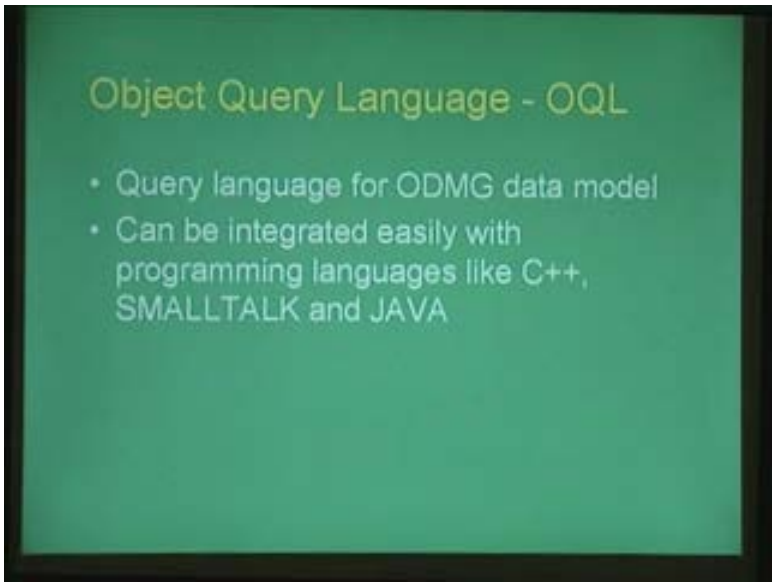
language independent mechanism of defining the structure and behavior of objects. What is meant by programming language independent mechanism? That is you can have the same ODL definition interface with let's say Smalltalk or java or C plus plus or any other programming language. And it supports different kinds of schematic constructs that define structural elements of objects and it can specify the database schema.

(Refer Slide Time: 20:57)



Note here that the database schema is actually a graph of different objects belonging to different classes and their relationships. So it can specify a database schema that is also independent of the programming language. And it has interfaces with specific languages like C plus plus and Smalltalk where you can use language bindings for mapping the ODL constructs.

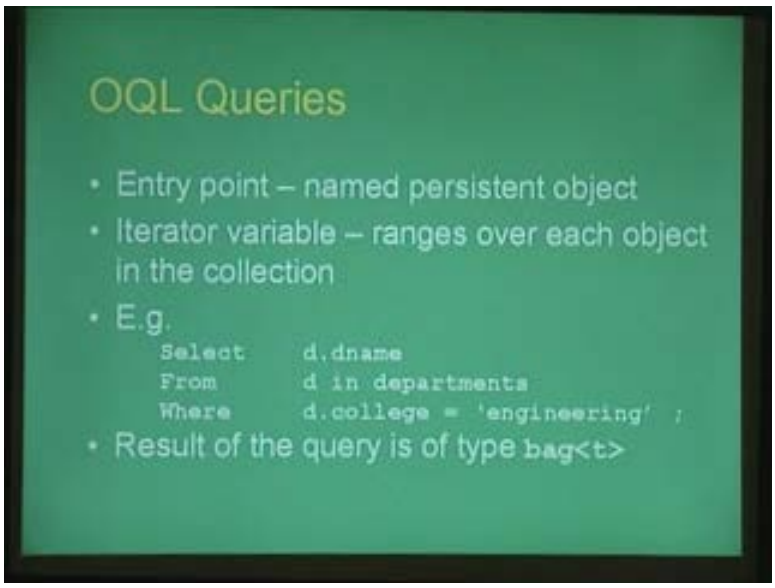
(Refer Slide Time: 22:25)



The ODMG standard in addition to the ODL also defines the OQL or the object query language. Object query language is a mechanism by which you can query on either attributes or behaviors of particular objects, usually for attributes of course. And it's a query language that's specified by the ODMG data model and it can be integrated with existing programming languages like C plus plus and Smalltalk or java and so on.

This slide here shows an example OQL query where note that for the programmer, the programmer is not necessarily aware of the OID mapping or the OID management that is performed by the object database system.

(Refer Slide Time: 22:33)



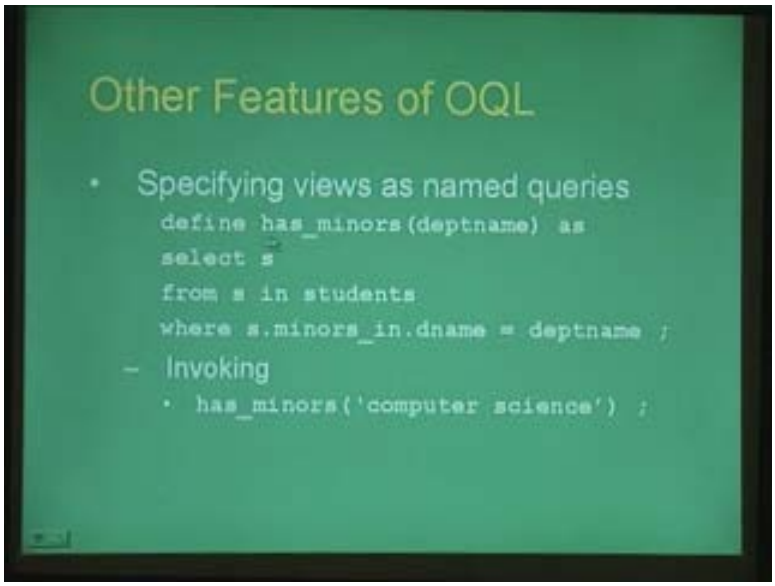
Instead a programmer refers to an object or a unique object in the database using things like an entry point or a reachability condition and so on. So an entry point in an OQL is a named persistent object that is that defines the entry to which the specific object with the particular OID can be accessed. So here in this case entry point is this object called d. So d is the name that uniquely defines an object that is stored, uniquely defines a persistent object that is stored in the database. And there are iterator variables that can iterate over or that can range over each object in the collection.

As you can see here (Refer Slide Time: 23:43) the structure of OQL is very similar to SQL itself where you just say select d.dname from d in departments where d.college equal to engineering except that here, the way it interprets this query is different from the way queries are interpreted in SQL or in the relational model. Here d in departments essentially means that d is the name for all objects that are in the extent called departments. Remember, what is an extent. Extent is analogous to an entity set that is a collection of different objects of the same type or the same class as is, so when I say d in departments, d is a iterator variable that is it iterates over every OID of objects that belong to class departments and then it selects a particular attribute from that.

Note here again that dname should be a visible attribute. If it is an invisible attribute you can access attribute names only through method invocation. So you should be saying something like d.get name, select d.get name from d in departments where d.get college equal to engineering and so on. So however we are assuming that d name and college are visible attributes rather than hidden attributes in the object definition.

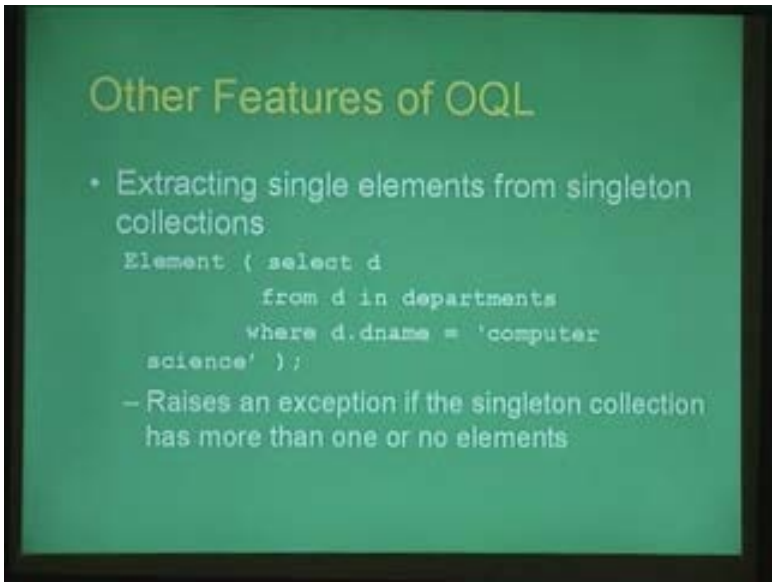
Just as in SQL, one can define views over the over the ODBMS using the OQL and this is quite similar to SQL query except that rather than say in create view, you are saying define.

(Refer Slide Time: 25:58)



And look at how views are essentially created. A view is created in the form of a method that is you are defining a method called has_minors department name as this OQL query. That is has_minors department name is select s where s is an iterator variable from s in students were s.minors in d dname equal to department name and so on. So you are basically defining some kind of a method when you are saying that you are defining a view. So you just invoke this method, you just invoke this view as though you are invoking a particular method. So you just call a function called has_minors like this and the output of this is the set of all students where those students are minors under working in a department that is what is specified by the query here.

(Refer Slide Time: 27:23)

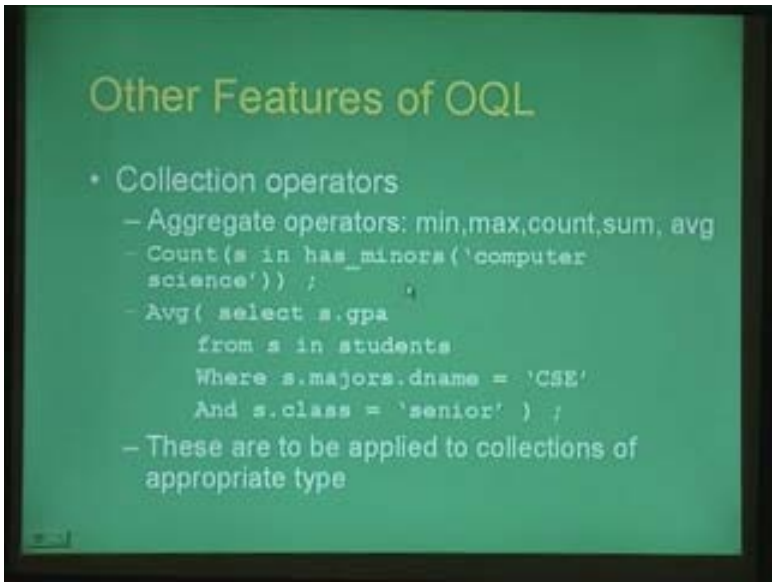


Similarly when a query returns a set of different objects or a collection of different objects, you can specify one single element of this collection and then use it as a separate declaration. This slide shows an example of such a situation where there is a nested query, nested OQL query here which says select d from d in department where d.name equal to computer science d.dname equal to computer science.

Again, so as usual d is a iterator variable that iterates over all objects or all OIDs of objects in the extent called departments. Now suppose there is only one department called computer science, this query returns exactly one object. And you can define this object as a specific element and use it as an attribute of some other object if required and that is the reason or that is the use for such a declaration. And of course such a declaration will raise an exception if this query actually returns more than one elements or on the contrary, it does not return any elements at all. That is if there is no department called computer science or two or more departments have a name called computer science.

Again some more operators that are specified by OQL are the aggregation operator or rather what are called as the collection operators where you can have aggregate operators just like in SQL like min, max, count, sum, average and so on.

(Refer Slide Time: 28:40)

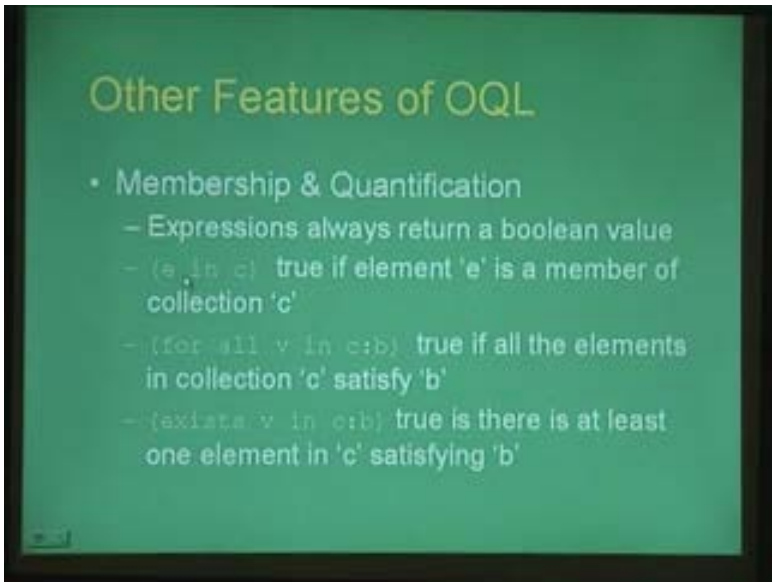


So you can get the minimum of all values of min over certain attributes and max over certain attributes and so on. And similarly count for example, count of something like this where the in command we have not actually see here but in is this operator that test for set membership whether s is a member of has_minors that is the set called has_minors and note that we had defined has_minors as another query and you can as a view defined by a OQL query.

And you can just have one more element like this which just checks whether any given element s is in has_minors and average over this thing. And so you are selecting s.gpa where gpa is a numeric attribute and of course again just like we mentioned earlier gpa is a visible numeric attribute. If it's an invisible numeric attribute then there has to be a associated method that has to exist in order to return the value of gpa. So you should in effect say s.get gpa or something like that and once you have got the set of all gpa values, you can take the average of this.

And of course all this aggregation operators have to be applied to operators of the or collections of the appropriate type. So even though collection can be collection of arbitrary objects, I really can't apply average over a collection of objects comprising of 3 numbers and 4 animals and whatever. So it has to be of the particular type. And so averages are generally are average or min, max and sum and so on are best defined over the set t where t would point to a numerical attribute or numerical object like number and so on.

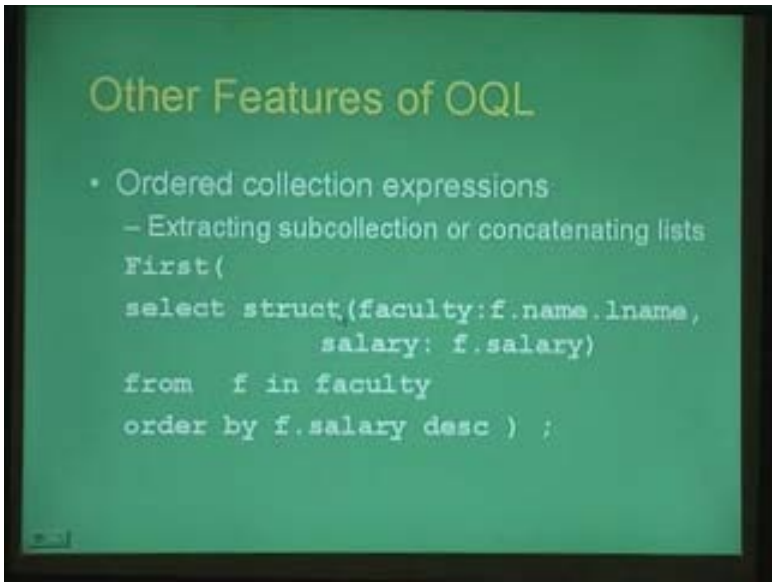
(Refer Slide Time: 31:34)



Similarly you can define memberships and quantifications operators in OQL and for example, you can define something like e in c where c is a collection object and e is an element. And this we saw it in the previous slide where you can actually, this expression actually returns true if e is an element of c. And similarly you can say, for all v in c:b that means in a sense its checking whether for all v in c does the predicate b or does the condition b hold true. So it is true if all elements in collection c satisfy this condition called b.

Now what is this condition? this can be again another expression like this, that is another conditional expression were e in c or e in v and so on where v is used as part of this expression in b. And similarly there exists or exists v in c such that b is true that is there exists at least one element in c satisfying b. Therefore as you can see these are quantification operators that are for example typically used in predicate logic were by you can specify different kinds of queries there exist and for all and that's existential and universal operators that are defined in predicate logic.

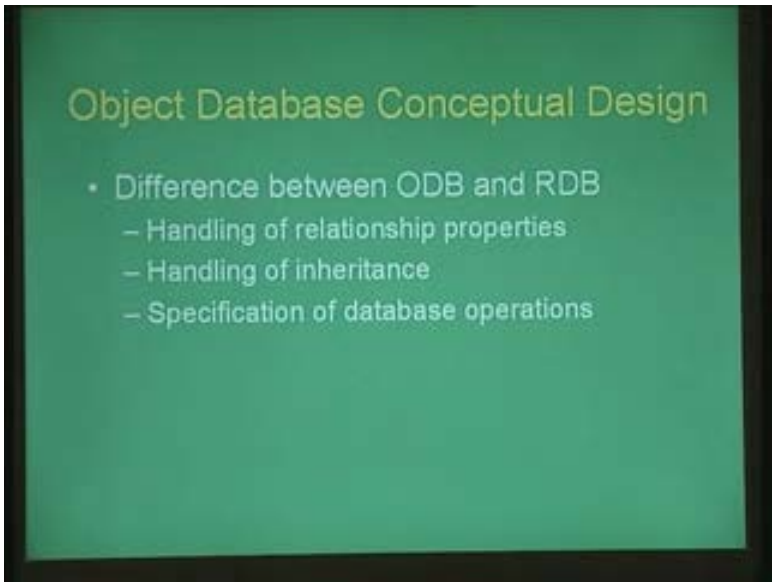
(Refer Slide Time: 33:02)



Then you can also define ordered collection expressions. For example here you are selecting a particular struct that is a particular attribute from an object which is a struct where f is the set of all or f is an iterator over all OIDs in the extent called faculty and you are ordering by faculty.salary and designation or description or whatever. So desc can stand for. And then from out of these you are taking the first element. So essentially out of a sorted list, you can take, you can look at any specific elements like first and second elements and then get or in effect you are seeing which is the faculty member who has the highest salary and so on.

And there are of course, we saw that an OQL is conceptually different from an SQL query. That is an OQL even though it looks very similar to an SQL query, the way it interprets the way an OQL query is interpreted is quite different from the way an SQL query is interpreted. And in OQL we define extents and attributes and iterators and so on whereas in SQL you have a tables and attributes and tuples and so on. And there are other differences between ODBs and RDBMS's especially in how relationship properties are handled, especially foreign key relationships and other kinds of relationships and in a sense an OBD database or ODBMS is an un normalized database in the sense that it is not even in first normal form.

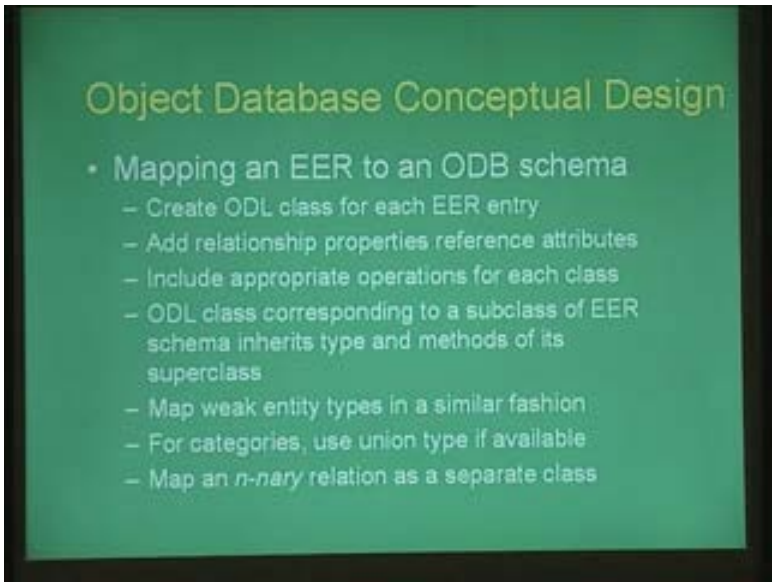
(Refer Slide Time: 34:31)



Remember that a relation is in first normal from if it is not a nested relation. That is a tuple does not contain nested tuples. However an object can contain tuples which can contain other tuples and so on to any levels of nesting. So it's not in first normal form and the way relations are handled in ODBMS's are mainly through OID references and OID references have little, if any to do with normalization in an ODBMS.

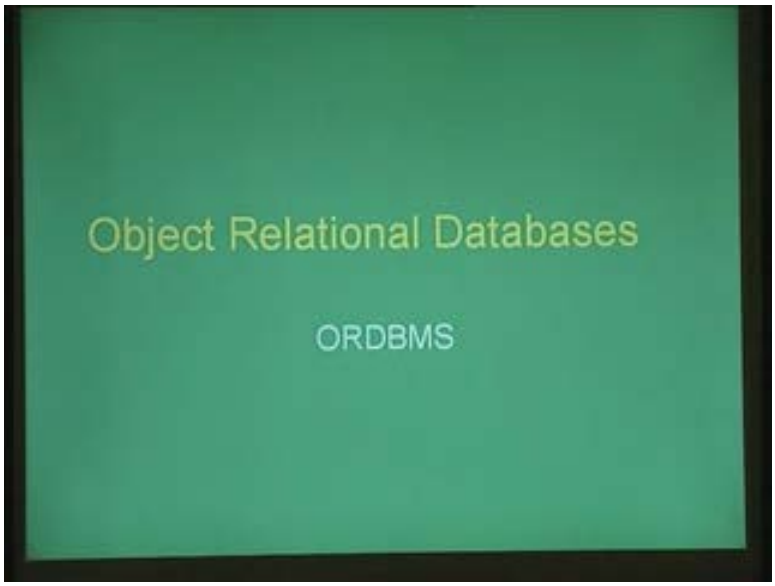
And then the handling of inheritance; inheritance is an integral part of ODBMS's whereas inheritance is an alien concept in relational database. So you don't really inherit, you don't really see types and classes inheriting or tables and tuples inheriting from one another and of course the specification of database operations. And there are also certain techniques which talk about how you can map a conceptual schema written in the ER model or the extended entity, EER model extended entity relationship model to an ODB schema rather than a relational schema.

(Refer Slide Time: 36:18)



We shall not be going through this slide in great detail though except to note that just like each entity look at the first step here where each entity in an ER schema corresponded to one table in a sense in the relational schema. Here it roughly corresponds to one class for each EER entity in a sense and relationships are referenced, relationships are essentially shown using associations rather than say foreign keys or having the same values and so on. And several different steps which are or several different thumb rules which specify, which tell a designer how to map from an extended entity relationship model to a ODB schema. And we shall not be going, looking at this in great detail.

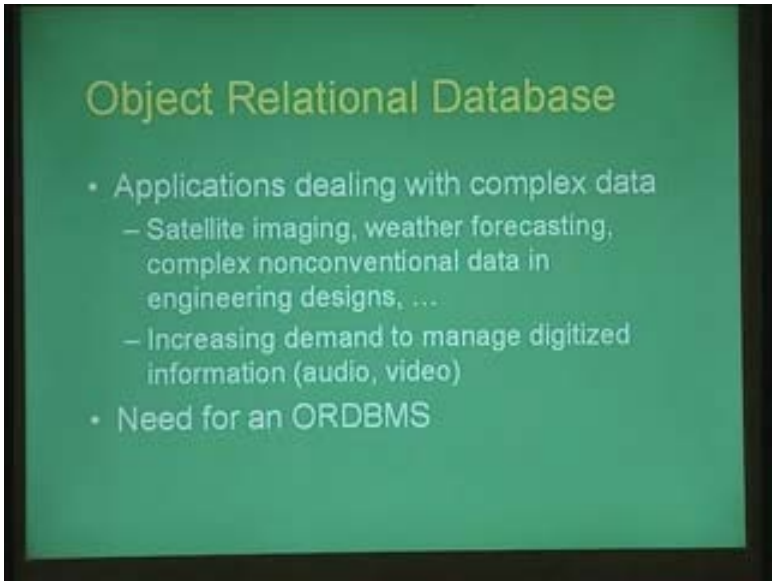
(Refer Slide Time: 37:10)



The next topic that we are going to see which is again of quite a bit of importance is the object relational databases or also what are called as ORDBMS's. And ORDBMS's are in some sense a middle path between RDBMS's and ODBMS's and both of them have their own pros and cons. ODBMS's are better suited for handling different kinds of complex objects like BLOBS and say and also in providing say behavioral abstraction where transistor is defined by a particular kind of methods and so on which are not present in the RDBMS space. However an RDBMS by itself has its own advantages which cannot be matched by a specific, by a pure ODBMS database. For example concepts like normalization and query handling and indexing, storage structure and so on where lot of work has gone into specifying not just the relational data model but also building an implementational scheme around the relational data model.

And all of these would have to be reinvented for at least a large part in the object database realm, if pure object orientation is being considered. Now the middle path is to use object relational database that is can we use both objects within relations and vice versa. So, an object relational database again are meant for applications dealing with complex data like satellite imaging and weather forecasting and so on and where the complex data is in turn associated with number of traditional data like which can be mapped on to tuples.

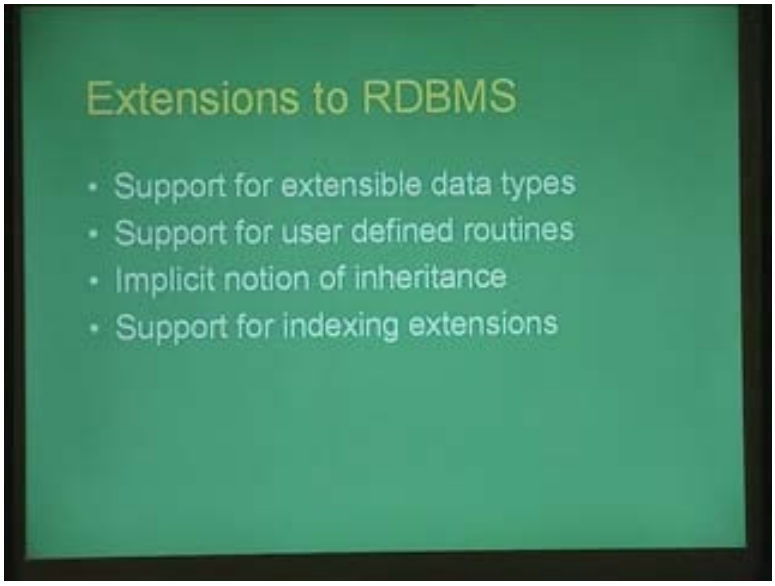
(Refer Slide Time: 38:57)



However it's not just the traditional data that is of interest here, it's the complex data that is how these complex data can be manipulated. That is what kinds of methods are there and how do we check the state of this data and so on. So there are several extension that have been proposed for RDBMS systems to make them compatible, to make them also to also support objects. For example there are support for extensible data types, remember that inheritance is not an integral part of the relational model and there have been some efforts to introduce inheritances or data type extensions into the relational model. And

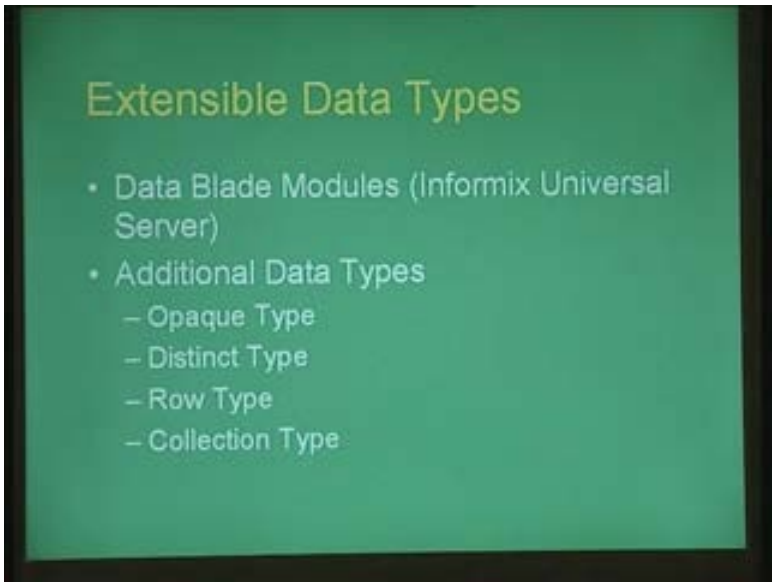
then support for user defined routines where the user can define certain methods by which data can be manipulated.

(Refer Slide Time: 39:50)



And implicit notion of inheritance that is you can, inheritance as an integral part of the DBMS system and some other extensions to indexing the traditional indexing of RDBMS systems.

(Refer Slide Time: 41:01)



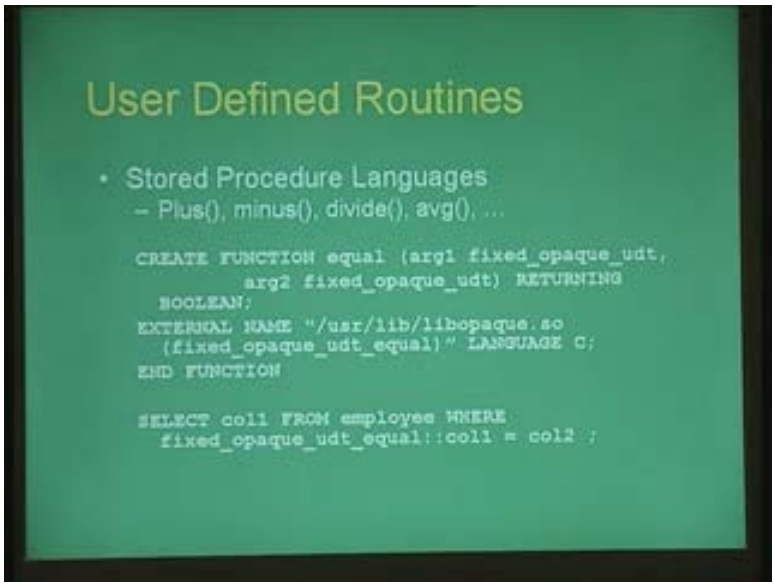
So we shall be looking at one specific kind of ORDBMS as kind of a mini case study in a in a sense namely the informix universal server. That's from informix which provides

extensions to the traditional relational data model by what are called as data blade modules. The idea behind data blade is as though that blade actually means a racer in this context, that is the actual meaning of blade.

The concept is as though a new data types or new extensions can be cut through existing definitions as though like a blade cutting through a fabric. So you can actually cut through something and then introduce new data types. And there are several kinds of additional data types that are proposed for example the opaque type or rather the opaque table in a sense an opaque table is one where the contents of the table are hidden just like in an object, if variables are hidden or invisible then it becomes an opaque table.

Now you can actually in a sense cut through an existing database system and introduce a new type called opaque tables and so that's the idea of a data blade module. And then there are distinct types and you have the notion of a row type that is you can define a tuple as a type and a collection type which is specified in the ODMG standard as well. Then user defined functions are provided in the form of stored procedures.

(Refer Slide Time: 42:11)

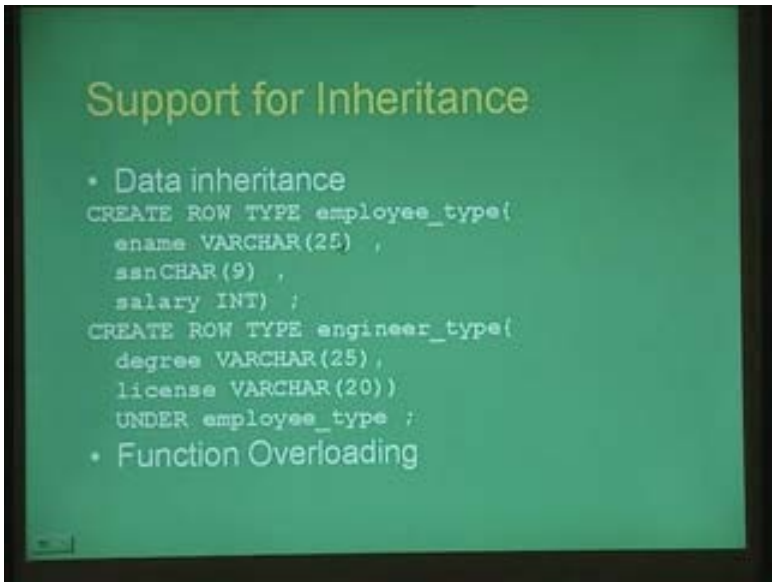


We already saw stored procedures when we are looking into case studies in RDBMS's. So in fact many of today's RDBMS actually supports stored procedures which are in some sense, which in some sense can be argued to be some kind of support for object databases. That is stored procedures in effect define some kind of a method that works on or that is applicable to tuples of specific kinds.

So here there is an example of creation of a stored procedure something like create function equal where you provide different arguments arg 1 and arg 2 which returns Boolean and then you define your function and you also define the language in which the function is written where you say it is written in language C and then end function.

So note here that the stored procedure in informix universal server doesn't really require the definition of the procedure to be specified here. The definition can be specified in a third party language like language C, like C language and then it's only the interface or the stub for the stored procedure which is available as part of the database itself.

(Refer Slide Time: 43:58)

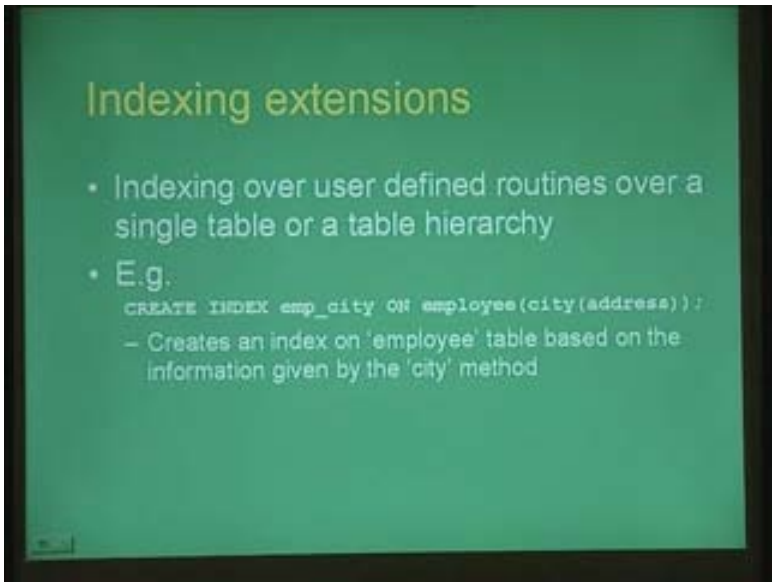


And the informix server also supports data inheritance as an integral part of its ORDBMS data model. So for example here you are saying, you first create a row type. Row type is again a tuple type which is created like this. So you create a row type as employee type which contains employee name and social security number and salary and so on. And you are creating another row type, engineer type which says which has two other variables degree and license and then there is this key word called UNDER employee_type.

That means, what this means is that the row type called engineer type is a sub type of employee type or it is a specialization over employee type. That is in addition to the fields that exist in the employee type, engineer type has two extra fields like degree and license. Similarly you can also define function over loading as well.

Then there are several kinds of indexing extensions that are defined and where you can create index not just on data values but also on user defined routines or user defined methods. For example this slide here shows a declaration, shows an invocation where it says a create index emp_city on this function called employee city address.

(Refer Slide Time: 45:07)

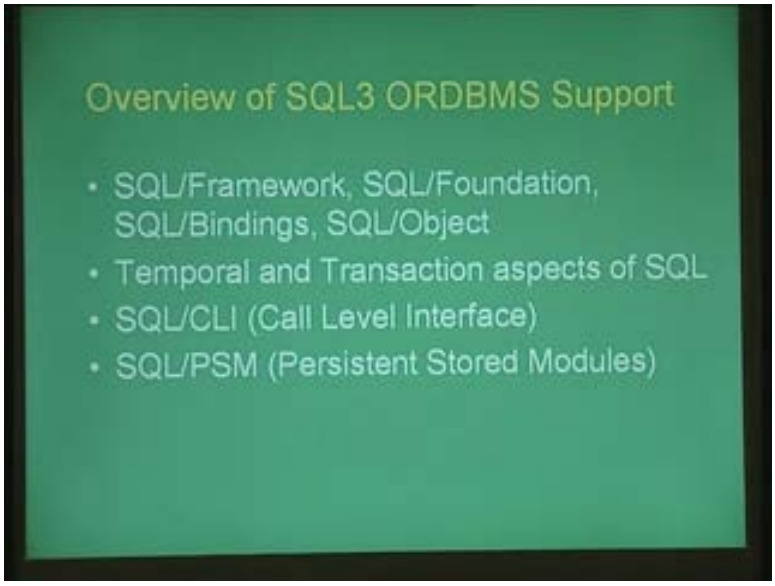


So this is actually a function over which the emp_city is being defined. So it creates an index on the employee table based on the information given by the city method. That is it first executes the city method on all tuples of this employee table and based on this information, you create an index that is some kind of a b plus tree index or whatever. And there are several kinds of APIs that the data blades provide and where you can something like two dimensional data types and image data types and so on. There is also construct for abstract data types where you can create, where you can define an abstract data type along with specific methods to manipulate the variables or manipulate the elements of this abstract data type.

(Refer Slide Time: 46:02)

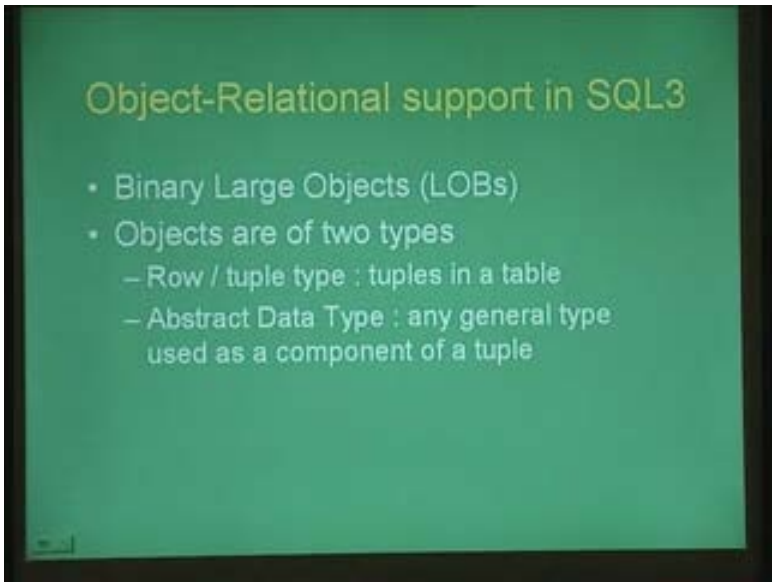
The next kind of ORDBMS support going from informix universal server, let us move on to another kind of ORDBMS support namely in SQL 3 or SQL 3 standard. SQL 3 standard has a number of extensions from the earlier SQL standard which provide some kind of constructs which are in some sense in line with ORDBMS requirements. And for example SQL 3 is divided into different parts like SQL framework, SQL foundation, SQL bindings and SQL object and it's in the SQL object part where number of ORDBMS support is provided as part of SQL 3.

(Refer Slide Time: 47:12)



And of course there are several other extensions that provides like temporal and transaction aspects of SQL and persistent stored modules and call level interfacing and so on. So as far as object relational support goes in SQL 3, SQL 3 supports a new data type called LOB or binary large objects or a large object and so on where you can define some kind of a binary dump or binary data object to be a large object or a binary large object.

(Refer Slide Time: 47:48)

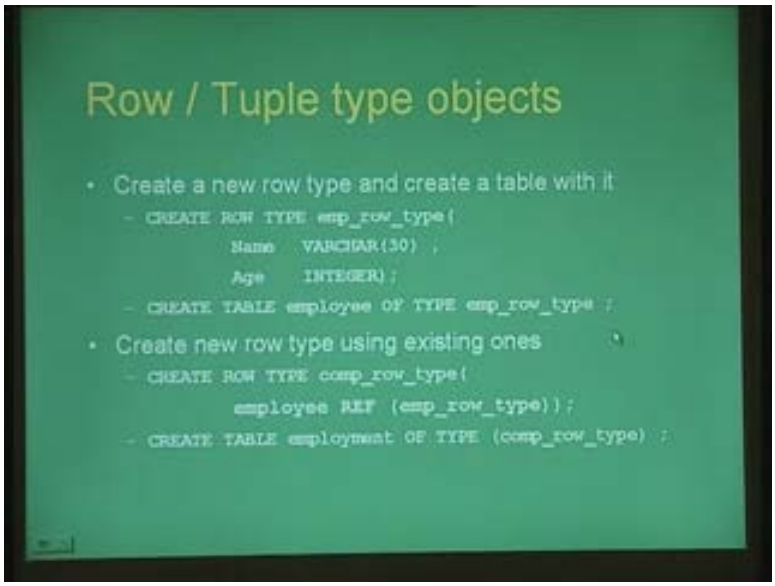


So for example you can have a media file like a video or a music file and as part of an attribute type and reference it and call it a binary large object. And there is also support for objects in the following ways where the first thing is the creation of row or tuple types that is just like in informix server, you can create a type out of a tuple. So once you create a type out of a tuple, you can create abstract data types like a struct which contains different elements of type tuples. So you can have different tuples that make up an ADT and then in addition certain kinds of methods that manipulate this tuples.

So here are certain examples that are shown in this slide. For example you can say this is an SQL 3 by the way. So you can say something like create row type where basically this is the tuple that is name is a varchar of 30 characters and age is an integer. Now this tuple is called emp_row_type that is this is a new type which creates emp_row_type and then you can simply say CREATE TABLE employee of TYPE emp_row _type.

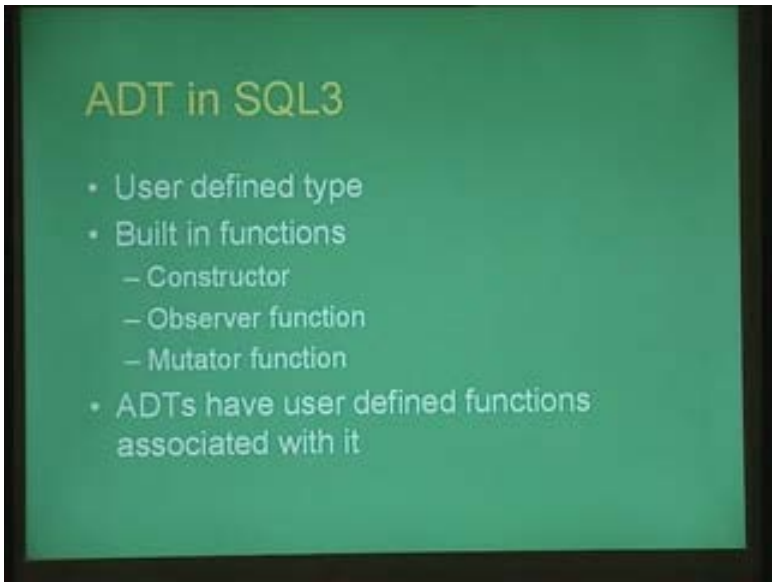
So it will just create a table of different tuples of this type. And you can extend existing rows, again the notion of type hierarchies comes into play here where you say CREATE ROW TYPE comp_row_type and then you say employee REF employee row type. That means you basically create another element and then just refer to the previous row type that you have defined. And all the attributes that have been defined there automatically come into this definition here and again you can say create table employment of type comp_row_type or so with the new type that you have created.

(Refer Slide Time: 49:34)



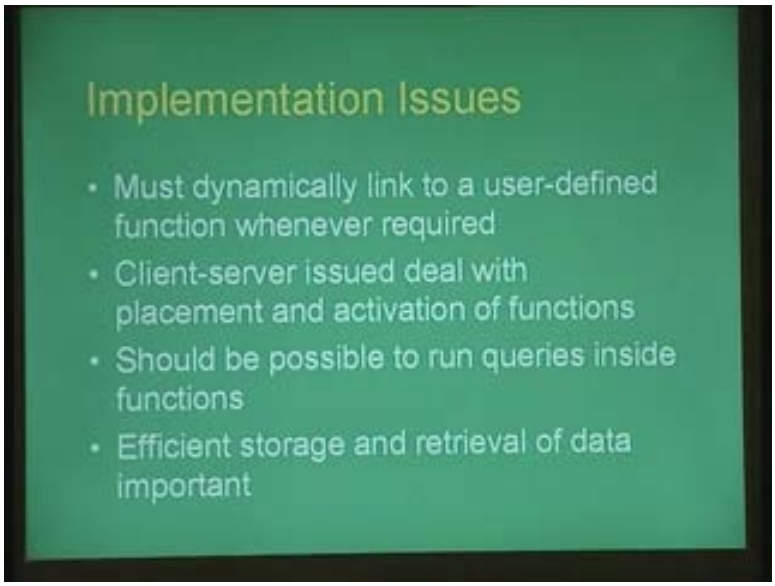
And in addition to row types, you can also define the notion of an ADT or an abstract data type in SQL 2. So ADT essentially is a user defined type for a particular variable, for a particular attribute. that is you can actually define a struct comprising of different tuples and different ADTs, nested ADTs in itself and then and then use that as a composite type, composite data object representing a type comprising of different attribute and so on. And then an ADT comes with built in functions like the constructor function wherein you can initialize and instantiate the ADT then there is an observer function which can, which is basically like the get and put operation.

(Refer Slide Time: 51:20)



So you can get the values of different hidden variables that the ADT defines and mutator function which is the put function where you can actually change this thing. And ADTs can also have, may have user functions associated within.

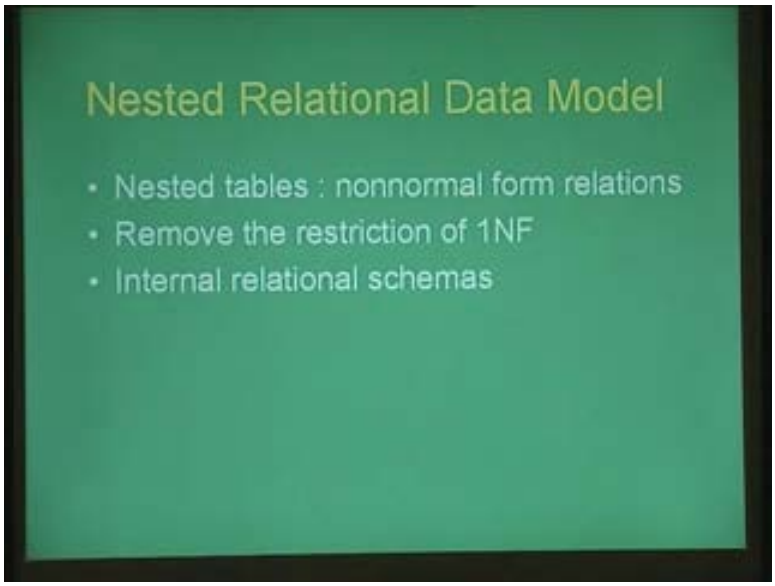
(Refer Slide Time: 51:38)



And there are several kinds of implementation issues in the SQL 3 specifications like suppose a user specifies a function or user defines a function then the implementation system, the DBMS implementing SQL 3 should dynamically link to a user defined function whenever required at run time.

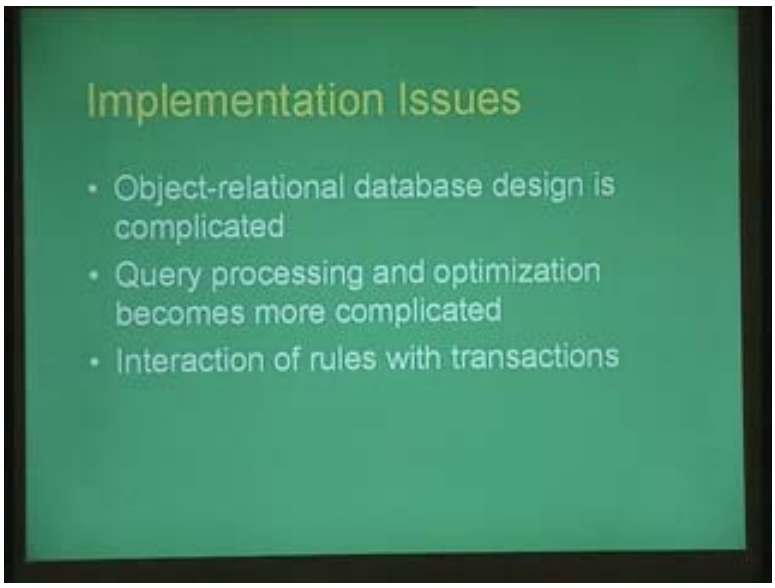
And there are other kinds of client server issues as to where the function is defined, if the function is defined on the client and it has to be executed on the server or whether the function should be defined and stored on the server beforehand and so on. And it's not clear whether one can run queries within functions and of course efficient storage and retrieval of data is also important.

(Refer Slide Time: 52:31)



So let us in a sense summarize the ORDBMS's issues, one is of course that object relational database design is much more complicated than relational database design. You might have already guessed why? That is creating indexes over functions or defining inheritances and so on can hinder or hamper with the traditional relational database storage and access functions. And query processing and optimization becomes more complicated when it comes to object databases. especially because when user defined functions have to be run, stored procedures have to be run there is no way to further optimizer to know how much time or how efficient is a, what is the behavioral characteristics of the user defined function.

(Refer Slide Time: 52:49)

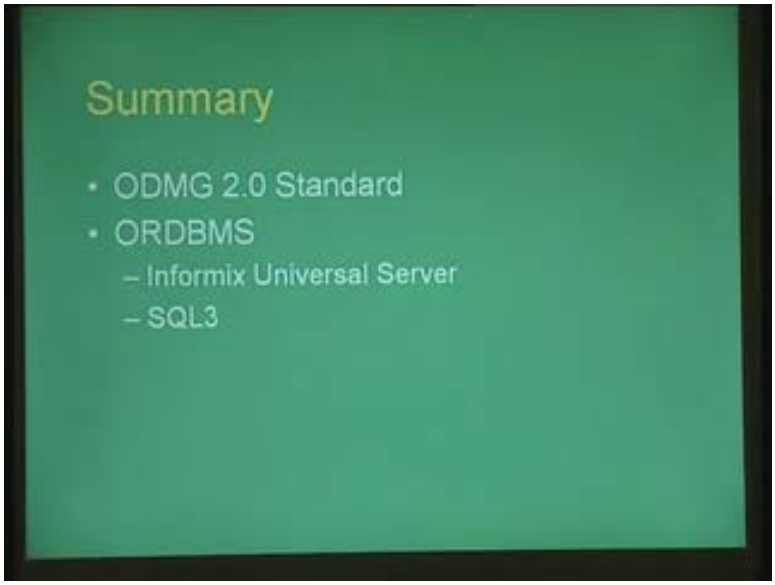


So that's the large unknown which the optimizer has to take care of when optimizing a query. And of course there are the notions of triggers and transactions and integrity constraints and so on which can further complicate the matter of maintaining an object relational database system.

Similarly because ADTs can define user defined types and an ADT might may contain a tuple which in turn can contain another tuples and so on. It basically an ORDBMS can become un normalized or de normalized because it's not even in first normal form. A first normal form each tuple has to contain atomic elements. So it is necessary to remove the restriction of first normal form on object relational databases. But internally even though that's the abstraction that is provided to the user, internally perhaps it is still more efficient to map the user abstraction that is provided to an internal pure relational database schema.

So let us summarize what we studied in this lecture or in this class. Starting from the previous lecture on object oriented database systems, we looked into the ODMG 2.0 standard and what kinds of object definition language and object query language constructs that it provides. And we saw how OQL queries although they look very similar to SQL queries are interpreted in a very different fashion in an ODBMS.

(Refer Slide Time: 54:53)



And the relative short comings and advantages of ODBMS, we saw RDBMS prompted the emerges of ORDBMS or object relational database systems. And we essentially saw two kinds of specifications, the informix universal server and SQL 3 support for ORDBMS. And of course ORDBMS themselves are by no means as elegant as the pure relational database. That means they don't have a nice specification and there are several issues like query optimization which is a major issue plus storage and retrieval and so on which still they have to contend with. So with that we come to the end of this session.