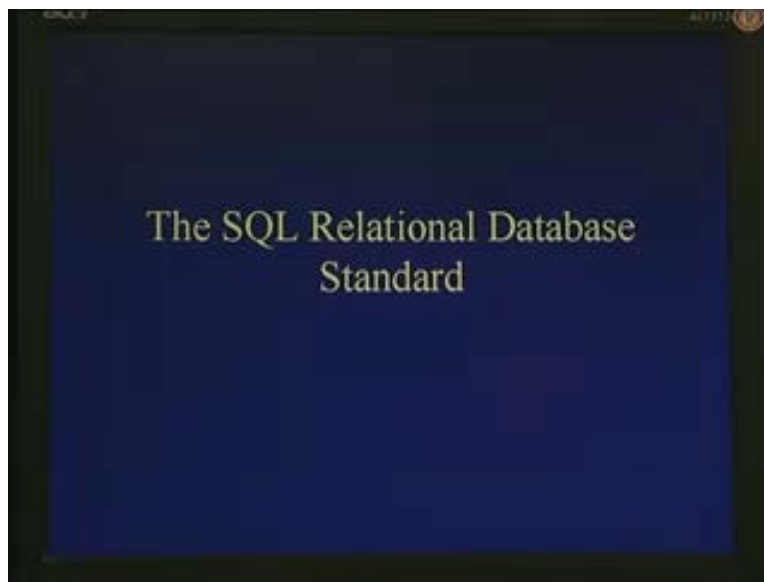


Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 5

Structured Query Language

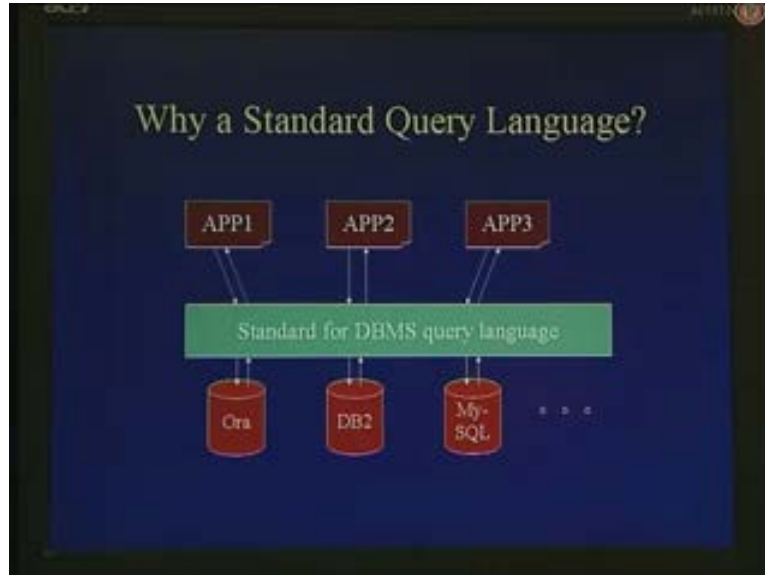
Hello and greetings. In the ongoing saga of management of data, we have covered the aspects of high level schema design using entity relationship modeling and one kind of a data model called the relational model which is mainly used for designing of low level schemas. Today we are going to look at something more concrete that is something which you are likely to be using on a day today basis, if ever you are going to be working in databases. Especially if you are a database user or database administrator, this is the SQL language or the structured query language.

(Refer Slide Time: 00:01:54)



The structured query language is the standard query language that is used by most database management systems that are available today and this is a language that is definitely required to be known by anybody who is going to be using a database management system. So let us move into structured query language today.

(Refer Slide Time: 00:2:19)



Now what is meant by a standard for query languages? Why do we need a standard for query languages? I just mentioned that the structured query language is the de facto standard or it's rather these standard for relational databases that all databases or most of the database users today use SQL in some form or the other. Why do we a standard query language for database management? Now today at this point in time, in the database arena there are large number of database management systems that are currently available.

Some of them are commercially available, some of them are freely available and some of them have been implemented in academic institution, some of them have been implemented in companies, industries and so on. We have oracle, we have IBM db two, we have Sybase, Microsoft sequel server, MySQL, Postgres and what not. There are so many different varieties of databases.

Now suppose we were to have a different query language being adopted by each different database management system, it becomes extremely difficult if not impossible to be able to port application program from one database to the other. We will have to write software that specifically uses let us say MySQL or software that specifically uses oracle and so on. However with a common query language like SQL, all that the application programs need to know is SQL. They should be able to speak SQL and using SQL we should be able to connect or use any database management system. So this is what is depicted in this in this slide here.

Suppose you have several different database systems, in fact in some application context you may have several DBMS within the same application context. A huge company for example may use several different kinds of DBMS systems in their different branches, while all of them have to deal with the same application, same work flow procedures or same business logic or business processes and so on. So suppose we have several

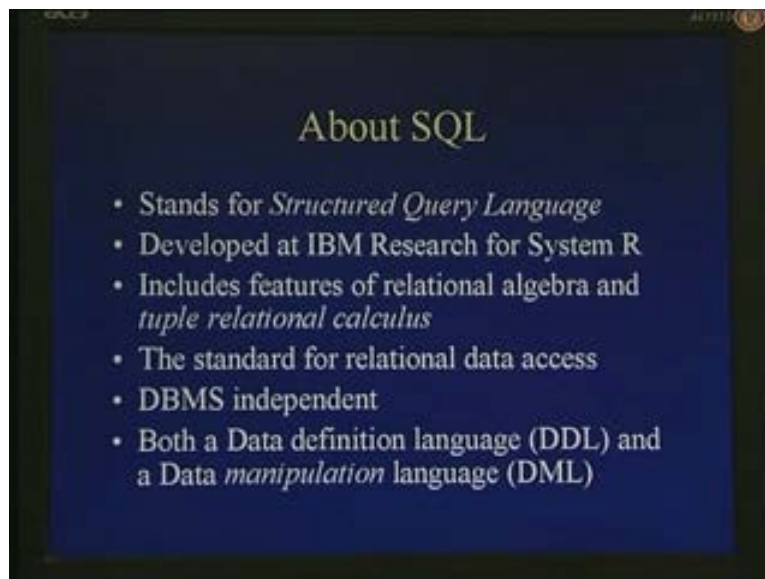
different databases or DBMS as shown in the slide here and suppose all of them are able to speak a query language like SQL, all that the applications need to do when there are different application says that they should be able to speak SQL.

And they should, it doesn't matter which DBMS they are going to be working. But reality obviously is much more complex than these and while this was the reason why a standard for database query was introduced, it is nowhere near to achieving its objectives. that is in reality however we still see applications that say this application is meant for MySQL, this is meant for oracle database, this is meant for db two and so on because there are number of additions, there are number of other features in addition to SQL that are unique to each different applications and so federating or working across different, connecting different database system is a completely different question all together now.

And data integration is a completely different question which we would be exploring in much more detail in a later session. So coming back to SQL, SQL stands for structured query language and this was introduced the genesis of SQL was at IBM research for database system that they had built called system R. Just like seminal paper for cord, you can give an internet search for a system R and there are of course some very good papers that are been, that are available on the net that talks about earlier days of SQL.

As you will see today, SQL contains a lot of constructs like select and set operators and Cartesian products which are quite similar to relational algebra that we saw in an earlier session and SQL also works on a relational data model very similar to relational algebra.

(Refer Slide Time: 00:05:57)

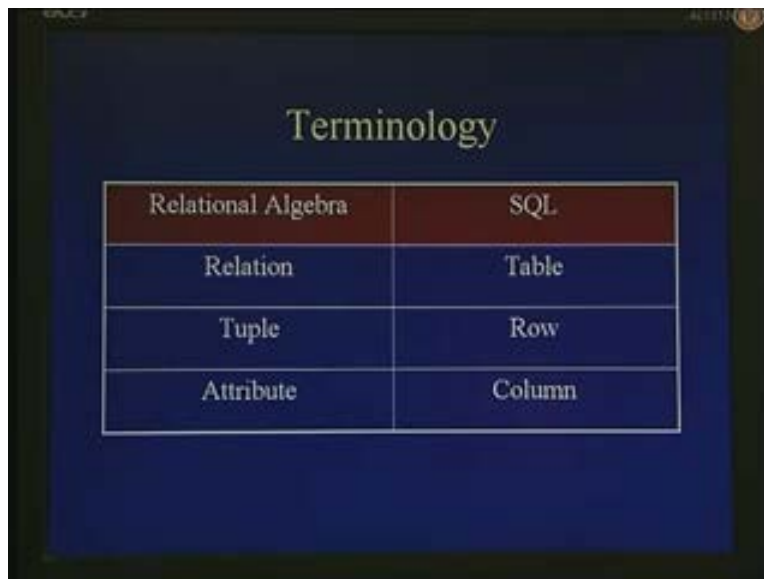


However SQL has no relation to relational algebra or very little relation to relational algebra in itself. The mathematical foundations for SQL is another data model called the tuple relational calculus which is also another data model based on the relational data modal that is the notion of mathematical relations. But there are some similarities in

terms of terminology between what we have seen in relational algebra and SQL. And SQL is a database independent language and in this session, we would be looking at construct from the SQL two standard and SQL three standard has some more facilities that are provided. And SQL is both a data definition language and a data manipulation language. What is it mean? A data definition language or a DDL essentially defines data elements. That means what are the data elements in a relational model? Of course a relation that is relation, attributes, domains, values, constraints, keys and so on.

So all of these can be defined using SQL. Using SQL you can define a relation, you can define its attributes, you can define its domains, you can define constraints across attribute, relationships, key constraints and entity constraints and so on. And it's also data manipulation language in the sense that you should be able to add more data into a database, you should be search for some data, you should be able to retrieve some data, you should be able to modify data elements and so on. So it's both DDL and a DML.

(Refer Slide Time: 00:8:30)

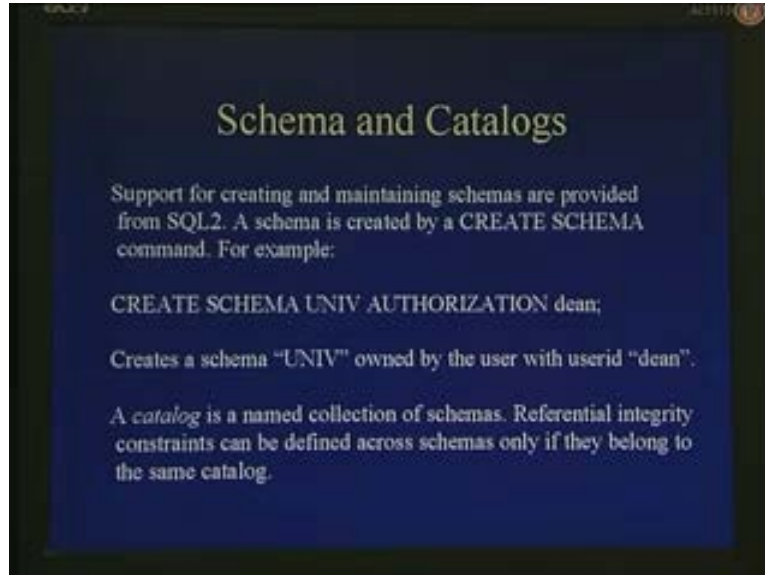


Relational Algebra	SQL
Relation	Table
Tuple	Row
Attribute	Column

So, in terms of terminology while we use the terms relation, tuple and attribute in relational algebra usually the terms table, rows and columns are used for the corresponding terms. We shall be using either of these terms interchangeably as they mean the same thing whether it is a relation or a table, we saw in the session on relational algebra that relation can be represented in the form of a table where each attribute of the relation is a column which is what is returned by the project operator. And each tuple in the relation is actually a row which is what is returned by the select operator.

The SQL two standard defines methodologies for introducing schematic structures into our databases. A schema is created by what is shown here as the create schema command. So the slide here shows a small example which says create schema univ authorization dean, so it creates a schema called univ which is owned by the user with a user id called dean.

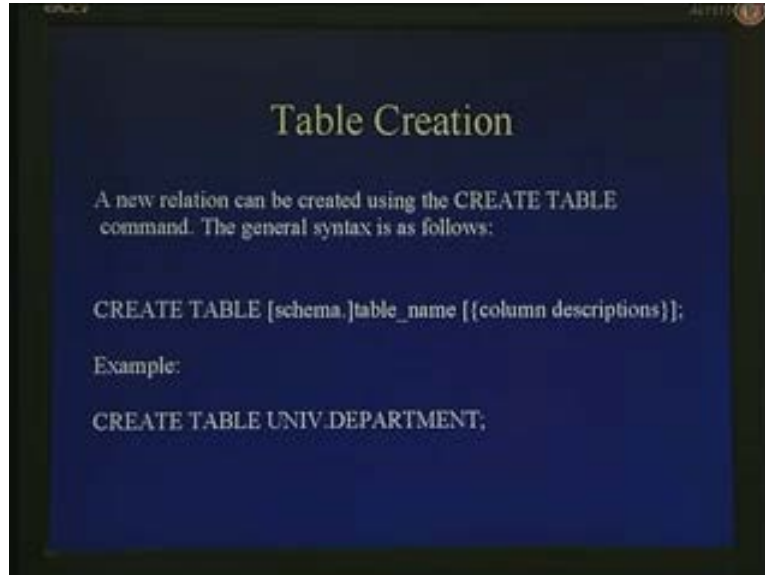
(Refer Slide Time: 09:21)



We shall be looking into this authorization in much more detail when we are talking about security and authorization in databases where different users of database management system are usually given certain privileges and certain authorizations which authorizes them to do certain kinds of data manipulation operations on the database. So hence if the owner of this schema is this user id called dean then a dean is given certain kinds of authorization that is defined by some default values which can also change here, which would typically include adding a table, deleting a table, adding rows deleting and so on. I mean any kind of activities that a typical owner of a database would do.

A catalog in SQL terminology is a named collection of schemas. So suppose I have a collection of different relations and combine them within particular name, this is called a catalog. And a catalog is required or it is required for different tables or different relations to be within the same catalog if I have to be able to enforce referential integrity on my schema. So using a scale you are able to enforce referential integrity only within tables that lie within the same catalog.

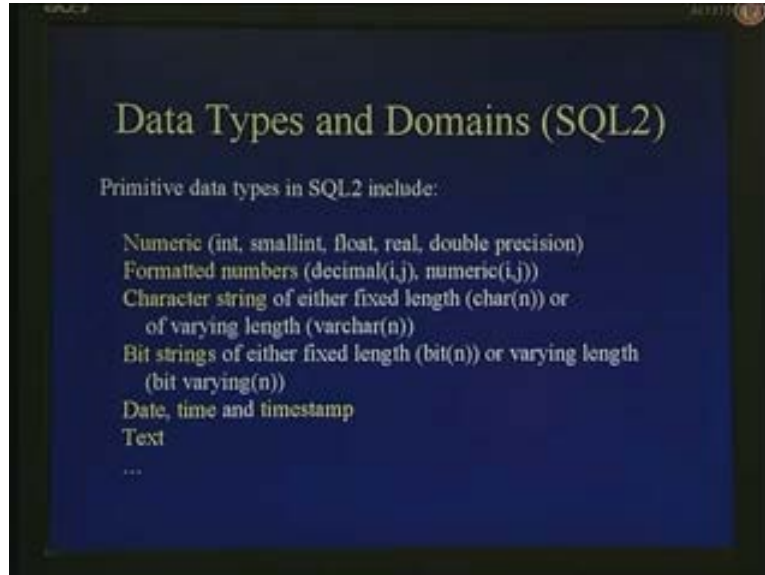
(Refer Slide Time: 00:11:19)



Creation of a table: As you know table is another term for relation and a table can be created with the command create table command, just like create schema for schema. So the slide here shows an example create table command of the form create table name and some kind of column descriptions. The parts of the syntax enclosed within box braces are optional structures that means you can refer to the table name when the context is clear or otherwise you need to identify the schema within which the table belongs. You can say something like create table univ.department that means create table called department under the schema called univ.

There are also certain column descriptions which we are going to see shortly which can say what are the different attributes that form the table and what are the domains of each of these attributes and what kinds of constraints that each of these attributes have. So before we look into column descriptions, let us say what kinds of data types are also domains that SQL two supports. There are several different kinds of domains that SQL two supports and some of the most commonly used domains are as shown here. There is the numeric domain which is identified by different data types called int, small int, float, real and double precision.

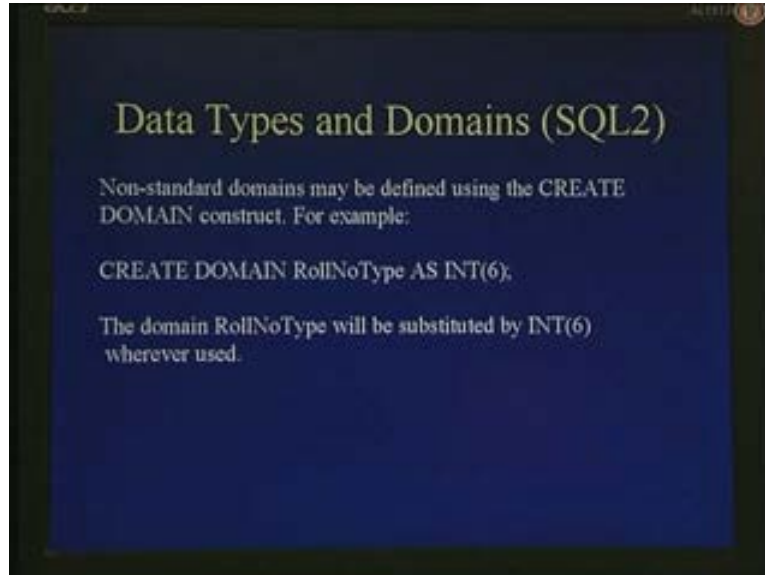
(Refer Slide Time: 00:12:30)



Each of these refer to different sizes that are of the data word that is being stored as part of this domain. There are formatted numbers like decimal I, j which basically means that this number has i number of digits of which j number of digits occur after the decimal point. Hence if I say a decimal 8, 2 it means that there are a total of 8 digit in this decimal of which two digits, the last two digits occur after the decimal point. You can also define character strings of either fixed length which is shown by char of n or of varying length which is defined by varchar of n.

So I can say that this attribute name is a varchar of say 60, so that means that this attribute can store a string whose length can vary anywhere up to a maximum of 60 characters. Then there are also big strings that you can store either a fixed length or varying length and similarly a special data types like date, time and time stamp and text and several other data types like binary objects and so on. But typically we would generally be working with numeric or character string in a typical transactional database system like employee records or railway reservation or whatever the standard transactional databases that we would be considering here.

(Refer Slide Time: 00:14:34)

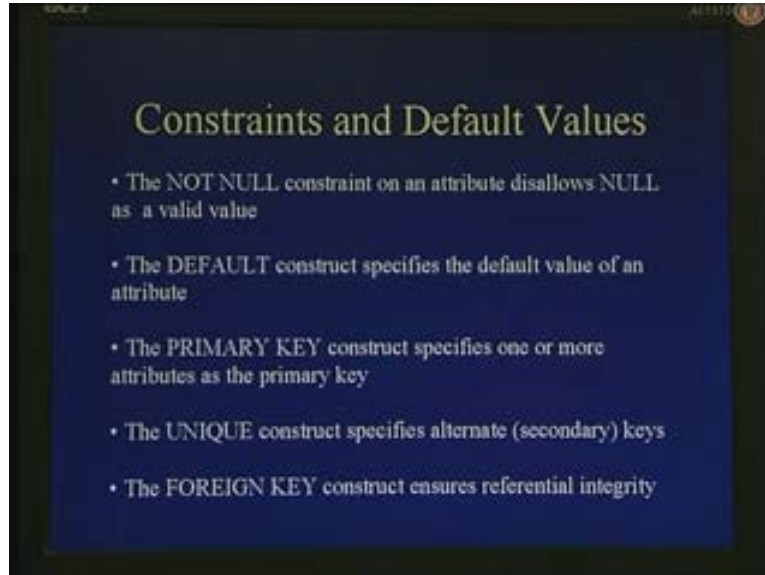


You can also create your own domain, name domain by using the create domain construct. The slide here shows an example which says create domain roll number type as int of 6. If I know that the roll numbers that I give to students in our institute would be an integer having 6 digits, I can as well use roll number type instead of int of 6 wherever I need to store student roll numbers.

The advantage with this is that tomorrow if this domain definition is changed, I need to change it at only one place where I am defining the domain called roll number type. There are also certain constraints and default values that you can specify using the create table command which you can specify on an attributes. for example the not null constraint, the first constraint is that we are going to take up is a not null constraint which says that this attribute name for which I am placing this constraint can never have a value called null.

So it disallows null as a valid value for this constraint. For example I can specify that the age of an employee may never be null that means I need to always have a valid age for an employee record that is entered in this data value, if I have to insert a particular row into the database.

(Refer Slide Time: 00:15:17)

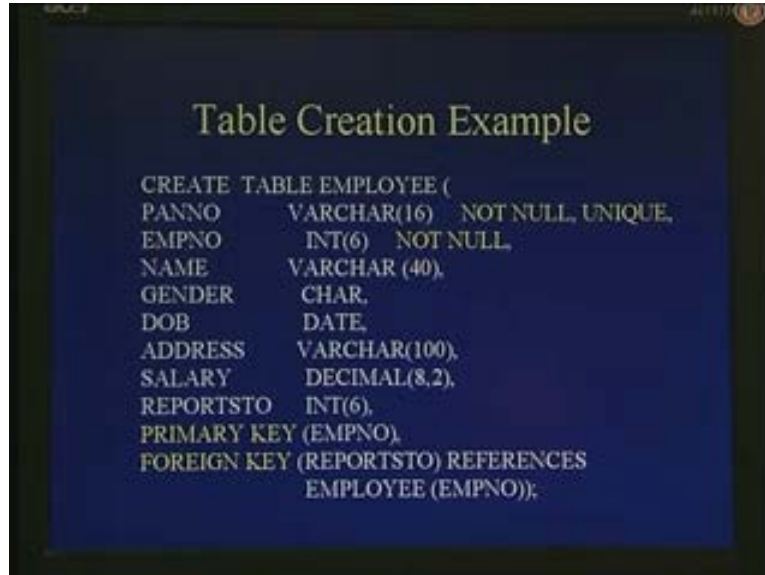


Similarly I can use the default construct, the default construct shows what would be the default value for a particular data item. so if the not null is specified and I also specify a default value, whenever let us say I specify that age of an employee not null and default 18. So whenever the age of an employee is not known, the default value namely 18 in this example is put in its place. Then there is the primary key constructs, the third constraint in this slide which specifies one or more attributes of this tables as the primary key of this record. Similarly there is the constraint called unique, I can say for one or more attributes I can say unique which essentially says that this attribute has to have unique values or distinct values for each row of this table.

In other words it means that it is an alternate key or a secondary key. Note that a key is something which can uniquely identify every tuple. Hence the key has to have a unique value for each tuple in the relation similarly I can also specify a foreign key construct that ensures referential integrity especially when some part of a table is actually referring to some other parts of another table, I can specify them as foreign key in order to ensure that they refer to existing aspects of the other table.

So just to brush up what is referential integrity, if I have some aspects of my table referring to another table it should either refer to an existing tuple or existing row in the other table or it should be null that is I should not refer to any row in the other table or I should refer to an existing row in formal terms this is what is meant by referential integrity.

(Refer Slide Time: 00:18:16)



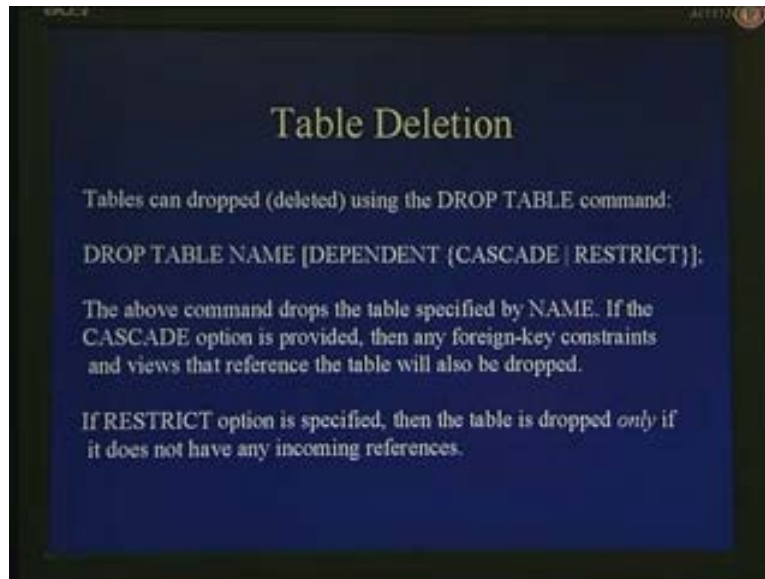
So here is an example of table creation. This slide shows an example table which says a create table employee, the name of the table or the name of the relation is called employee and then there is a set of attribute descriptions. The first attribute called pan number, the permanent account number of an employee is given a varchar that is variable character string of 16 letters and it is shown as not null and it is shown as unique. That is this pan number may never be null and it has to be unique across each employee that is each employee has to have a distinct pan number. If you go down the slide you can see that there is another construct called primary key and emp number that is the second attribute employee number which is also shown as not null is termed as a primary key.

Hence the pan number in this case because it's unique can form a secondary key or an alternate key of this relation. Then there are other attributes like name, gender, date of birth, address, salary and reports to which basically shows the employee number of the manager or the person to which those employee reports to. So you also say another constraint called foreign key which says that reports to, the field called reports to is actually a reference to another record belonging to the same relation called employee and referring to the employee number in this record.

Therefore whenever I enter a table and I enter a particular employee number for reports to, suppose I enter details of an employee and enter the details of the manager by specifying the employee number of the manager to whom these employee reports to then because of the foreign key constraint that is specified here the database management system will verify whether a record for the manager already exists. If the manager record does not exist that is the employee with that reports to employee number does not exist then addition of this employee record will fail as part of the database management.

Deletion of tables: Tables can be deleted, the terminology used for table deletion is called drop. So tables can be dropped using the drop table command, this is shown in this slide here.

(Refer Slide Time: 00:21:05)



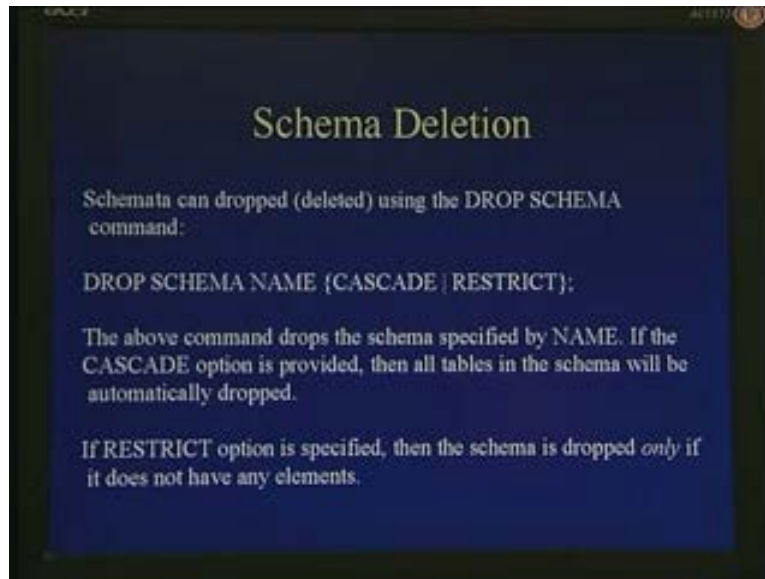
You can use, the syntax is something like this drop table and name of the table and there are certain optional attributes which says dependent cascade or restrict. So the first three terms are obvious, drop table name that is the table name by the given name should be drop. What is the dependent clause or what is it do? So if dependent is termed as cascade if I say drop table employee dependent cascade, then any foreign key constraints that the table holds or views that reference the table, well we have not come to views as yet but for the moment let us just not consider that. But when I drop a table let us say employee, any foreign key constrains will also be dropped.

So the dropping is in some way a cascading process because of table employee, any foreign keys that the table references to or all going to be dropped in a cascading fashion. on the other hand if the restrict option is specified then a table is dropped only if it does not have any references, incoming references that is only if nobody references the table only then will a table be allowed to drop. Just like table deletion, you can also delete an entire schema using the drop schema command. This is shown in the slide here. The drop schema command also has a very similar syntax, it is like drop schema name and either cascade or restrict.

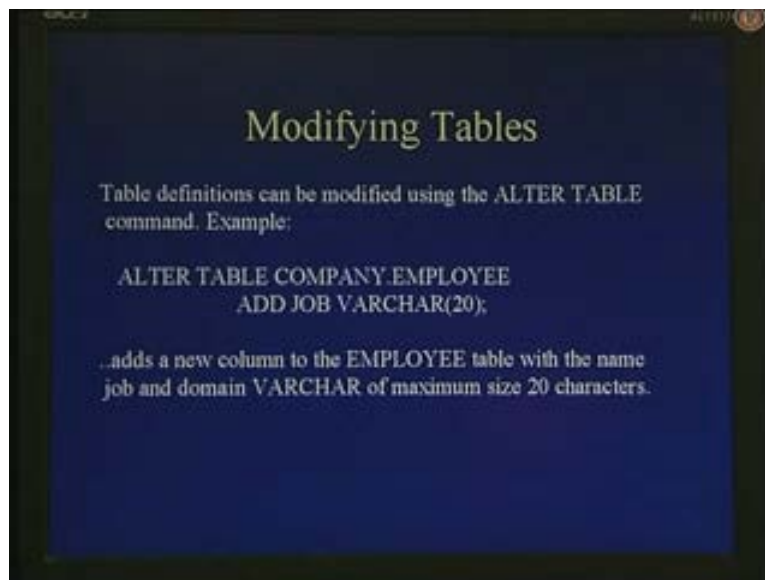
So drop schema and name is obvious that is you have to drop a schema by the given name and if the cascade option is provided then all tables that are there in the schema will be automatically dropped. On the other hand if restrict option is specified, then a schema is dropped only if it does not have any table or if it does not have any elements. How do we modify tables? How do we alter an existing table definition? So table definitions can be modified using what is called as a alter table command. Note that alter table or

modification of a table is modification of the schema of the table not the data in the table. That is we are not modifying existing data elements or adding or deleting data elements from and to the table, we are actually modifying the table definition or the table schema.

(Refer Slide Time: 00:23:04)



(Refer Slide Time: 00:23:12)

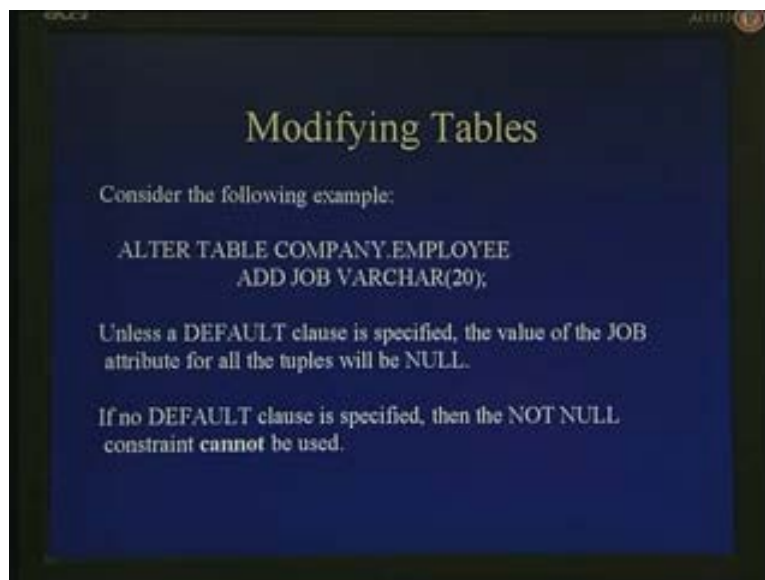


The table schema is simply a definition of the set of all attributes that form the table and their domains. so the slide here shows an example which says alter table company dot employee add job varchar 20, so it essentially adds a new column to the employee table with the name called job and domain varchar that is a variable character string with a maximum size of 20 characters.

Now suppose the table that I am altering **has already containing** is already containing some data that is I have created a table, I have added certain data elements and I have used the database for sometime and then suddenly I give an alter table command and say add a new column like job. Now what happens to, what value should job get in all of these tables? Because I have not specified any value as part of the alter table command.

You might have guessed it that it is going to be given a default value of null. So a new column called job is going to be created with a value of null but what happens if I specify a constraint called not null? Suppose the slide here reads alter table company dot employee add job varchar of 20 not null. Now what should be the, what should be the value that has to be filled in for this new column?

(Refer Slide Time: 00:25:28)



The answer to this is in the next slide here. Now unless a default condition is specified, unless a default value is specified you cannot use the not null constraint, as simple as that. That is if I don't use a not null constraint and the query is just like this that is which says alter table company dot employee add job varchar 20, it just adds this new column with all null attributes. On the other hand if I wanted to specify not null then I should also use a default value which is what is going to be filled for all the data elements in this new column.

So I can say default employee or default shop floor or something like this. So the default value called the shop floor is going to be filled for all of the elements in this new column which should later be change for specific rows using some other command which we are going to see later on.

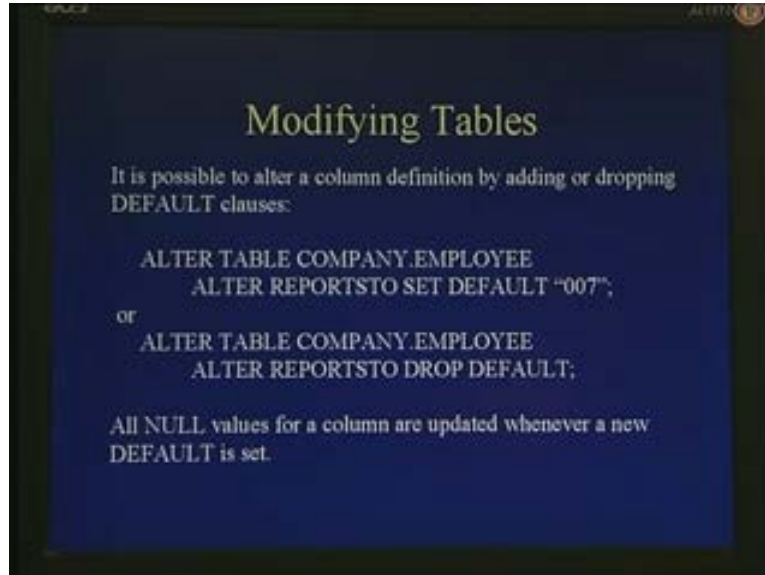
(Refer Slide Time: 00:26:31)



Now we just saw how to modify a table by adding a column. What if we need to delete a specific column? The syntax is again quiet similar to that of deleting tables. We use the key word called drop, so this table here shows an example alter table company dot employee and as it to drop the column called pan number and there is an option called cascade.

Now drop pan number is obvious that is the column called pan number is going to be dropped. If the cascade option is used then all constraints that refer to this column are also dropped automatically. That is if some other column refers to this column as in the form of a foreign key or a view or so on then they are all going to be dropped automatically, it's a cascading process. Similarly if restrict is used then a column is dropped only if there is no incoming references to this particular column.

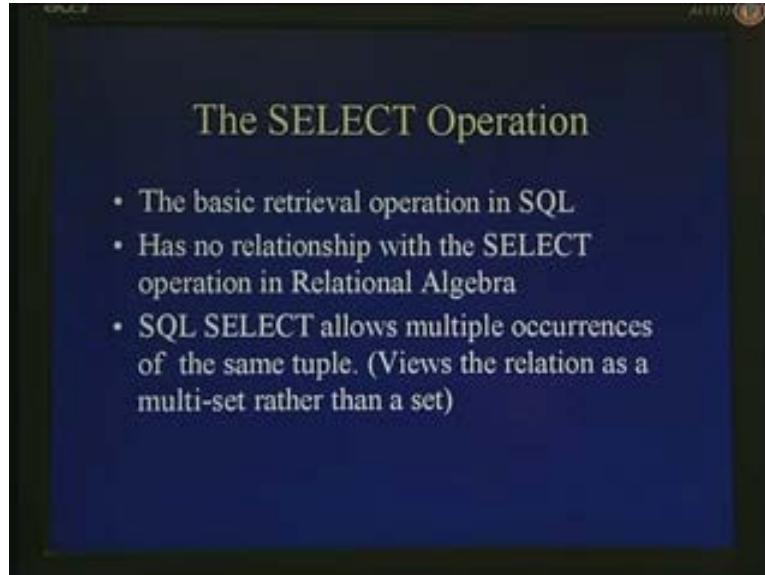
(Refer Slide Time: 00:27:41)



It is also possible to alter a column definition rather than just adding and deleting new columns. Column also has a particular domain and a particular constraint set of constraints that are associated with it. Now it is also possible to add and drop these domains or constraints that refer to a column and this slide here gives certain examples. The first example shows alter table company dot employee and in turn says alter reports to that is alter the column called reports to set default as 007. So what is this do? This basically says that wherever the reports to, wherever the reports to column is null set it with the default value called 07.

So whoever whichever employee does not have any manager, assign him to the manager called 007 which is what is being set by this command. The second command shows alter table company dot employee, alter reports to drop default which is basically the other way around. That is suppose it already has a default value then all those rows where this column has the default value are set to null and the default value is going to be dropped for this column.

(Refer Slide Time: 00:29:11)

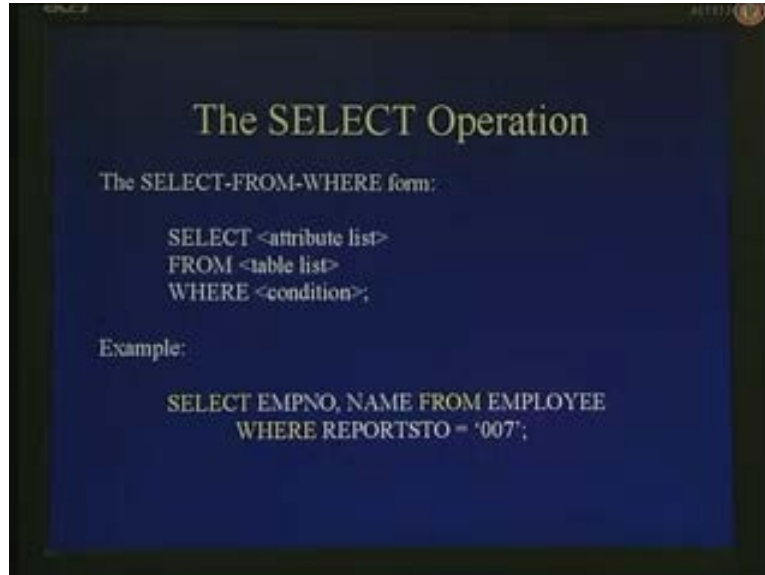


We now come to the main operation in SQL, the most frequently used operation for retrieval of data elements from um from tables which is called the select operation. We have not really seen how to add data into a table as yet but let us first see how to retrieve data from a table and then we are going to consider how to add or modify data elements to and from a table.

So the select operation is a most detailed operation in SQL and is the most frequently used operation and it has the variety of forms which we are going to see in a step by step fashion. So the SQL operation is the basic retrieval operation, the select operation is the basic retrieval operation in SQL, it has no relationship with the select operation in relational algebra. Just to reemphasize the point that SQL is actually based on tuple relational calculus and we are going to see here that the select operation of SQL can perform both select and project that are defined in relational algebra.

And SQL select one major difference between SQL select and that of relational algebra is that it considers relations as a bag. We saw in the session on relational algebra that by default relational algebra expects relation to be sets and when we convert them to bags we have to take care of certain algebraic conditions with sets which does not necessarily hold for bags. But SQL by default considers tables to be bags and not sets. There may be multiple occurrences of the same tuple.

(Refer Slide Time: 00:31:05)



The SELECT Operation

The SELECT-FROM-WHERE form:

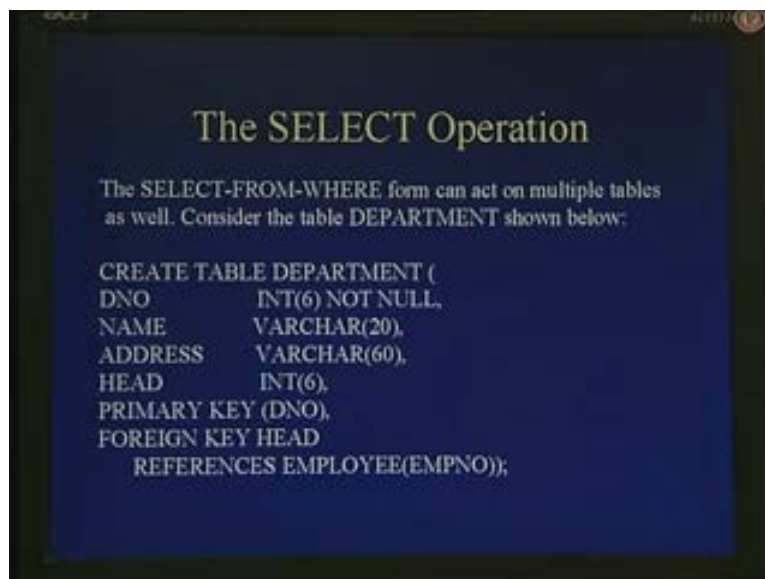
```
SELECT <attribute list>  
FROM <table list>  
WHERE <condition>;
```

Example:

```
SELECT EMPNO, NAME FROM EMPLOYEE  
WHERE REPORTSTO = '007';
```

The basic syntax of a select operation is shown in this slide here. It is very simple, it says select attribute list from table list where a given condition. so there is an example which shows here select employee number, name that is the list of attributes emp number, name from employee which is the name of the table where reports to equal to 007 which essentially means that show me all employees, that is give me the employee numbers and names of all the employees who reports to a manager whose employee number is 007. The select from where is the basic operation that we saw here and it can also act on multiple tables, it need not act on a single table. Until now we have being considering one single table called the employee table. Now let us work with two tables just to show that select can act on multiple tables

(Refer Slide Time: 00:32:03)



The SELECT Operation

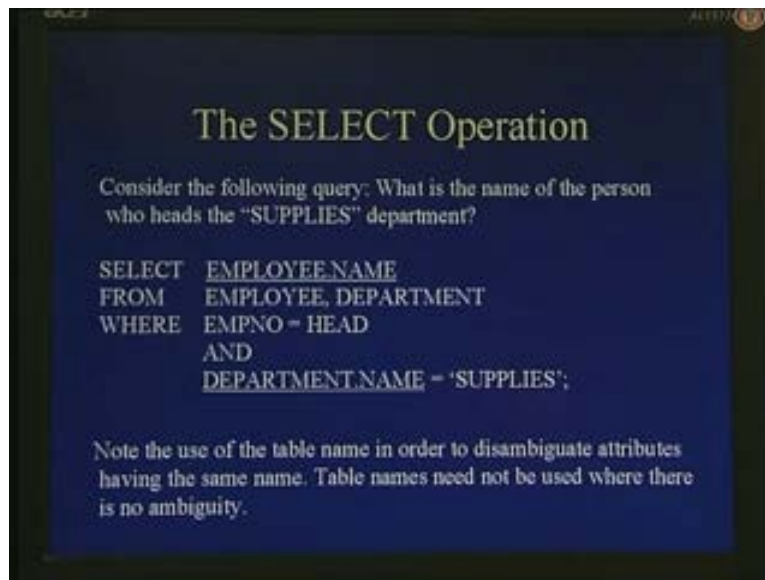
The SELECT-FROM-WHERE form can act on multiple tables as well. Consider the table DEPARTMENT shown below:

```
CREATE TABLE DEPARTMENT (  
DNO          INT(6) NOT NULL,  
NAME         VARCHAR(20),  
ADDRESS      VARCHAR(60),  
HEAD         INT(6),  
PRIMARY KEY (DNO),  
FOREIGN KEY HEAD  
REFERENCES EMPLOYEE(EMPNO));
```

Now let us first define a new table called department. so this slide shows the definition of department that is create table department and which says where the first attribute is called dnumber which is also the primary key, the department number which is int of 6 and not null and name address and head that is name of the department, address of the department and the head, the employee number of the person who heads the department. Now it also retrace the fact that head is a foreign key that refers to employee number from the employee database or from the employee table.

Consider the following query what is the name of the person who heads the supply department? If you look back at the definition of department, we have seen that the department contains department name, address and head which is the employee number. It does not contain the employee name. But the query here requires the name of the person who heads the supplies department. So the name of the department is supplies and we require the name of the person, so this can be specified by small SQL statement like this, select employee dot name from employee and department where employee number equal to head and department name equal to supplies.

(Refer Slide Time: 00:32:51)



The SELECT Operation

Consider the following query: What is the name of the person who heads the "SUPPLIES" department?

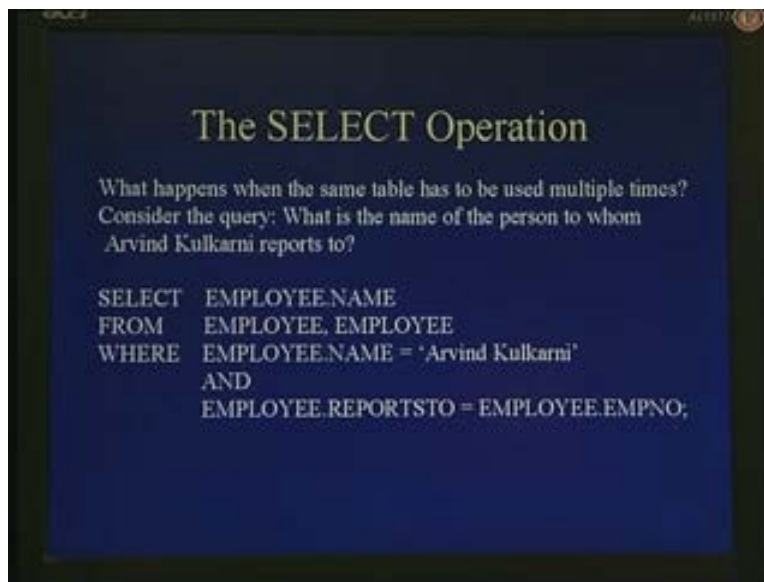
```
SELECT EMPLOYEE-NAME
FROM EMPLOYEE, DEPARTMENT
WHERE EMPNO = HEAD
AND
DEPARTMENT-NAME = 'SUPPLIES';
```

Note the use of the table name in order to disambiguate attributes having the same name. Table names need not be used where there is no ambiguity.

As you might have imagined this is quiet similar to performing a relational algebra select on a Cartesian product of two tables. In this case there is a Cartesian product of two tables employee and department and we are stipulating the fact that employee number equal to head in this Cartesian product. That is considering only those tuples where the head of the department corresponds to the employee number of record in an employee and the department name equal to supplies. And also note the use of that table name in order to disambiguate attributes having the same name. Now even the employee table, in the definition of the employee table the name of the employee is specified by an attribute called name.

But in the same way in the department record as well, in the department table the name of the department is also referenced by an attribute called name. Now when we say select name, which do we mean? Do we mean the employee name or the department name? In order to disambiguate this we can prepend the name of the attribute with the name of the table. So the query here says select employee dot name rather than saying just name and then also in the where condition where department dot name equal to supplies. So for some strange reason if some employee is called supplies that should not be matched, its only the department name which should be matched against supplies. However this disambiguating attributes by prepending them with the table name is not always sufficient.

(Refer Slide Time: 00:35:40)



Consider the next query here. Now the query here says what is the name of the person to whom Arvind Kulkarni reports to? Now here is an employee with name called Arvind Kulkarni and he reports to some person. Now we need to know the name of the person. Note that in the employee table, we only have the employee number of the person to whom each employee reports to. So obviously we need to have a join of the employee table on the employee table itself that is you have to have a self join for the employee table. So suppose we write a query like this that is suppose we try to write or we try to disambiguate attribute names by putting the table names before them. So such a query shown here that is select employee dot name, so we need employee dot name from employee, employee because both employee and manager are both employees.

So it's a select employee dot name from employee, employee where employee dot name equal to Arvind Kulkarni and employee dot reports to equal to employee dot emp number. Obviously you see that there is something, there is quiet a bit that's wrong here. You don't know which employee table are you referring to, is it the first employee table or the second employee table?

(Refer Slide Time: 00:37:14)

The SELECT Operation

What happens when the same table has to be used multiple times?
Consider the query: What is the name of the person to whom
Arvind Kulkarni reports to?

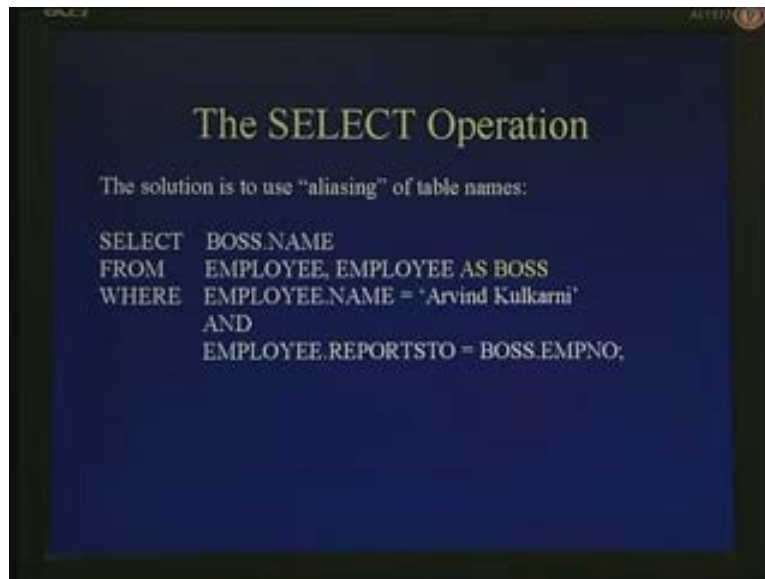
```
SELECT EMPLOYEE.NAME  
FROM EMPLOYEE, EMPLOYEE  
WHERE EMPLOYEE.NAME = 'Arvind Kulkarni'  
AND  
EMPLOYEE.REPORTSTO = EMPLOYEE.EMPNO;
```

Ambiguous!

So this is still ambiguous. So in order to disambiguate this attributes names in such a situation, SQL provides as with the opportunity of using what are called as aliasing. So aliasing can be used as follows. Now consider the same query shown in this slide here. So this slide for the time being concentrate only on the second line of this slide where it says from employee, employee as boss.

So the entire query is like select boss dot name from employee and employee as boss. So essentially what it saying here is that take the first table employee and the second table employee, however use an alias called boss for the second table. So we know whether we are talking about an employee or his boss. And then we say employee dot name where employee dot name equal to Arvind Kulkarni and boss employee number is the same as the employee reports to number.

(Refer Slide Time: 00:37:36)

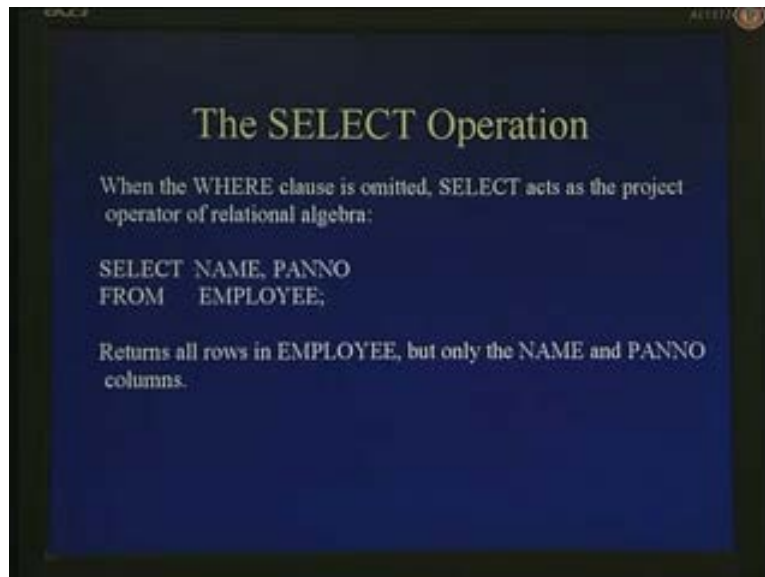


The solution is to use "aliasing" of table names:

```
SELECT BOSS.NAME
FROM EMPLOYEE, EMPLOYEE AS BOSS
WHERE EMPLOYEE.NAME = 'Arvind Kulkarni'
AND
EMPLOYEE.REPORTSTO = BOSS.EMPNO;
```

So employee dot reports to equal to boss dot employee number. So in this case we will be able to identify which name are we referring to from which relation.

(Refer Slide Time: 00:38:44)



When the WHERE clause is omitted, SELECT acts as the project operator of relational algebra:

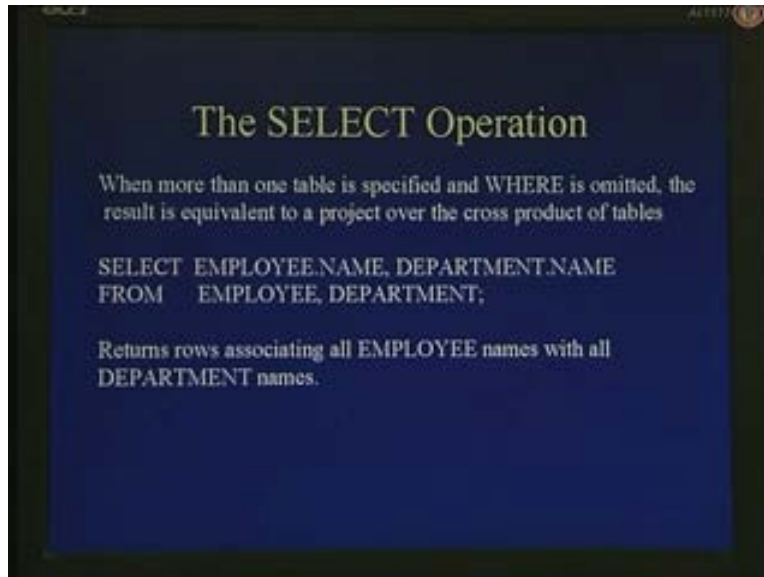
```
SELECT NAME, PANNO
FROM EMPLOYEE;
```

Returns all rows in EMPLOYEE, but only the NAME and PANNO columns.

Suppose we omit the where clause in the select from and where syntax and we just give a query of the form that is shown here. That is select name, pan number from employee. What is going to be the output of this? As you might have imagined this, such a select statement is similar to the project operation in relational logic. So what this statement does is it returns all rows in the table called employee, however only the columns name and pan number.

So it is similar to saying project name and pan number from employee. And what happens when the where clauses omitted and instead of saying just one table name, we actually specify more than one table name. This is shown in the query here. It says select employee dot name, department dot name from employee, department that's it.

(Refer Slide Time: 00:39:43)



The SELECT Operation

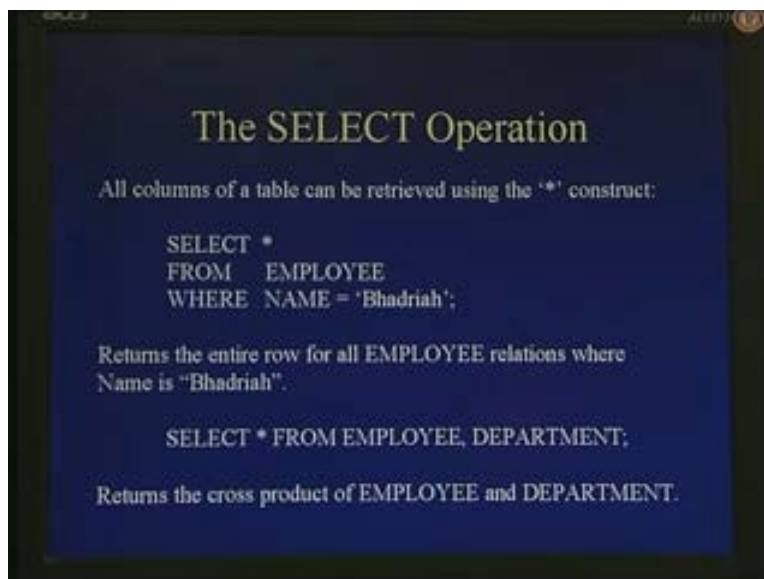
When more than one table is specified and WHERE is omitted, the result is equivalent to a product over the cross product of tables

```
SELECT EMPLOYEE.NAME, DEPARTMENT.NAME  
FROM EMPLOYEE, DEPARTMENT;
```

Returns rows associating all EMPLOYEE names with all DEPARTMENT names.

Now what happens here in this case of course we get only two columns as output that is employee dot name and department dot name, however we get all possible combinations of employee dot name and department dot name. In other words we have computed a Cartesian join or a Cartesian product between employee and department with this operation.

(Refer Slide Time: 00:40:07)



The SELECT Operation

All columns of a table can be retrieved using the '*' construct:

```
SELECT *  
FROM EMPLOYEE  
WHERE NAME = 'Bhadriah';
```

Returns the entire row for all EMPLOYEE relations where Name is "Bhadriah".

```
SELECT * FROM EMPLOYEE, DEPARTMENT;
```

Returns the cross product of EMPLOYEE and DEPARTMENT.

Suppose we want to select all columns of particular table that is similar to the select operation in relational algebra. We want to select all or entire tuples and based on certain conditions, in such a case you can use the term called star as shown in this query. Here it says select star from employee where name equal to Bhadriah. So, essentially this query is similar to the relational algebra expression which says select or sigma name equal to Bhadriah from employee. That means the entire row or the set of all attributes of relations where the name attribute is called Bhadriah is going to be returned.

Similarly, if I say select star from employee, department in the second query that is shown in the slide here, it computes the complete Cartesian product between employee and department. We now come to the fact that how tables are treated in SQL. In relational algebra we have seen that by default relations or tables are considered to be sets. On the other hand in SQL, tables are considered to be multi-sets or bags that is multiple tuples having the same values are tolerated. We have also seen why this is sometimes not only desirable but also necessary, it is desirable because it is expensive to remove duplicates.

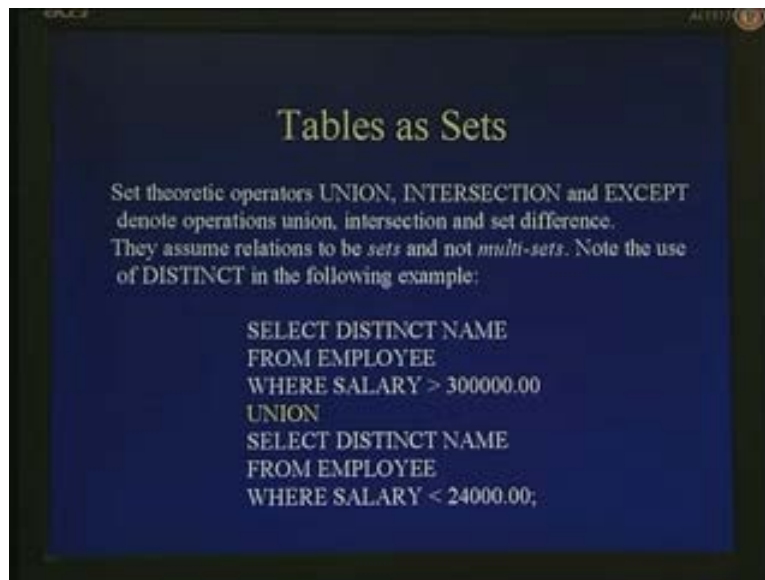
(Refer Slide Time: 00:41:13)



Suppose I return a query with 10000 records of which there may be hundreds of duplicates. I need to perform, I need to first sort each of these, this whole set of records and then remove duplicates and then reorder the records in whatever order that the user has asked for. Therefore it is very difficult or it's an unnecessary over head to remove duplicates, therefore it is desirable in many cases to tolerate duplicates. And in some cases it's actually necessary to tolerate duplicates. We have also seen examples of these. Suppose I want to compute the average marks of all students in a particular course, it is not only desirable but it is actually necessary that I retain the duplicates because the duplicates all contribute to the total number of marks which have to divide by the total number of occurrences to find out the average marks.

So for computing any aggregate properties, I need to have duplicates. However in some cases if I want to remove duplicates explicitly from the output of a query, in SQL you can give the clause called distinct. As part of your select statement, the table, the slide here shows such an example, it says select distinct name from employee so which simply says that show me the set of all distinct names that the employees have. So if two or more employees have the same name then they are shown only once as part of this query.

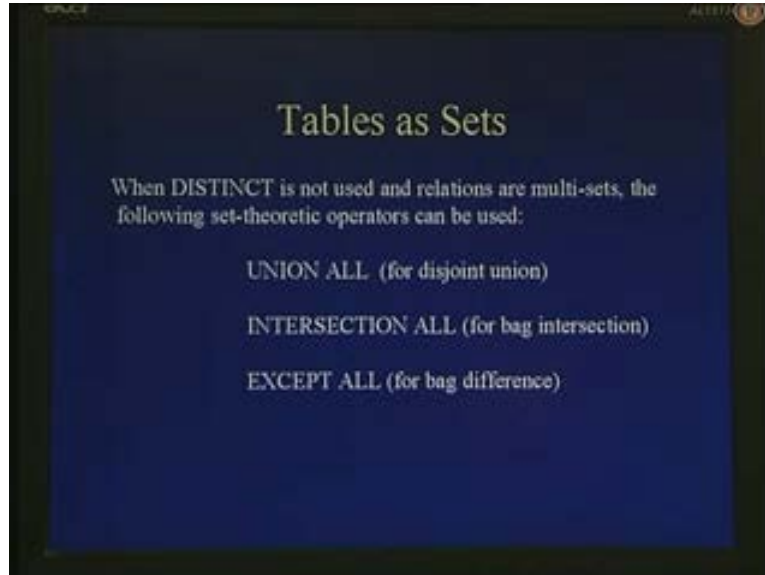
(Refer Slide Time: 00:43:32)



Similarly one can perform several set theoretic operations like union, intersection and set difference using SQL. So union is operated by using the clause called union and intersection by the clause called intersection and set difference by the clause called except as shown in the slide here. Now, by default union, intersection and except assume that the sets that they are operating upon are actually sets and not multi-sets. So note the use of distinct in this example. The example here says that select distinct name from employee where salary is greater than 3 lakhs union select distinct name from employee where salary is less than 24000.

So essentially what its doing is that it is selecting the set of all names of employees who are earning more than 3 lakh and combining them with the set of all names of employees who are earning less than 24000 and duplicates are removed in these sets of name. So the union operator assumes that duplicates are removed, when it is performed in the union of these two sets.

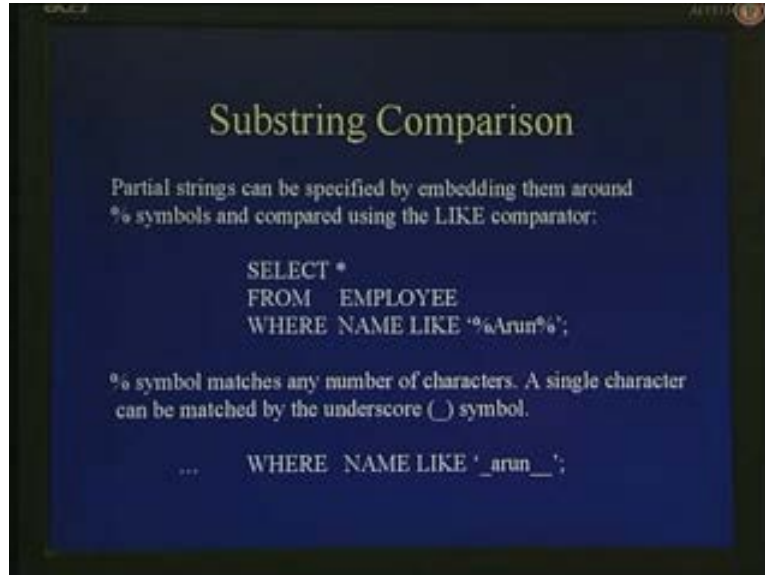
(Refer Slide Time: 00:44:55)



On the other hand if I want to tolerate duplicates or if I want to specify that the sets are actually bags and not pure sets, therefore I need to perform a disjoint union or a disjoint intersection. Remember what is the disjoint intersection of two sets, if a tuple or if the data item occurs multiple times in an intersection, for an example it has to occur the minimum of the two number of times. So if I have to specify that I am actually working on bags and not sets, I need to specify that with the key word called all which is shown in the slide here. So if I have not specified the distinct construct in my select statements as in my previous examples, I should use the term union all for disjoint union and intersection all for bag intersection and except all for difference or set difference between bags.

We can also perform comparisons over character attributes especially string attributes by comparing partial strings or comparing wild cards. So this slide here shows two such examples. The first examples says select star from employee where name like percent, Arun percent so note the use of firstly the key word like and secondly the use of the percentage symbol. So a percent symbol matches any number of characters wherever it occurs, therefore this query here matches employee name where the employee name contains Arun as a substring, A r u n as a substring and where it may be preceded by any number of characters and succeeded by any number of characters.

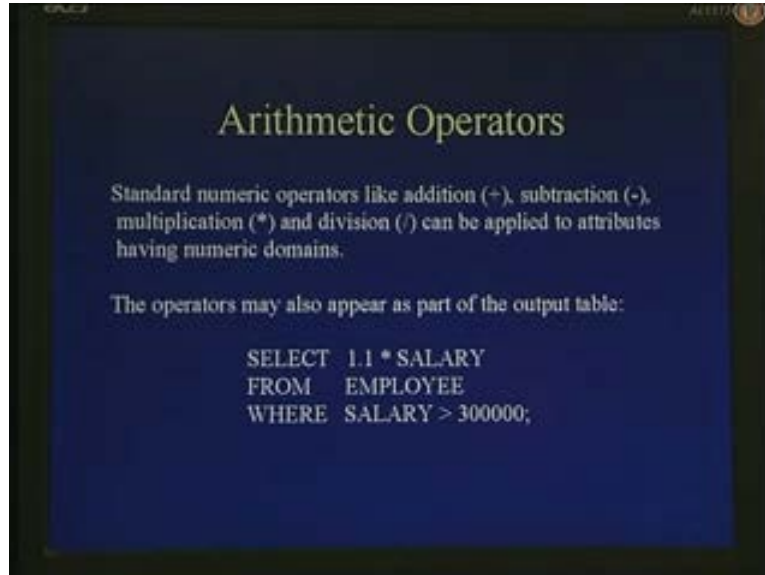
(Refer Slide Time: 00:45:53)



So while the percent symbol matches any number of characters, a single character can be matched with the underscore symbol. So suppose if I had said where name like underscore arun underscore underscore, so it essentially looks for one character before Arun and two characters after Arun. So any kind of character, any kind of name where Arun occurs as a substring with exactly one character before it and two characters after it. One can also specify arithmetic operators like addition, subtraction, multiplication and division. So I can say where salary plus perks not greater than 50000 or so on.

And I can also use these arithmetic operator not only in the where clause but also in the select clause. So have a look at the example shown in the slide, this slide shows a query which says select 1.1 times salary that is 1.1 into salary from employee where salary greater than 3 lakh, so which basically says that show me what would be the figures, salary figures of employees, if salaries were to be raised by 10 % effectively.

(Refer Slide Time: 00:47:22)



Arithmetic Operators

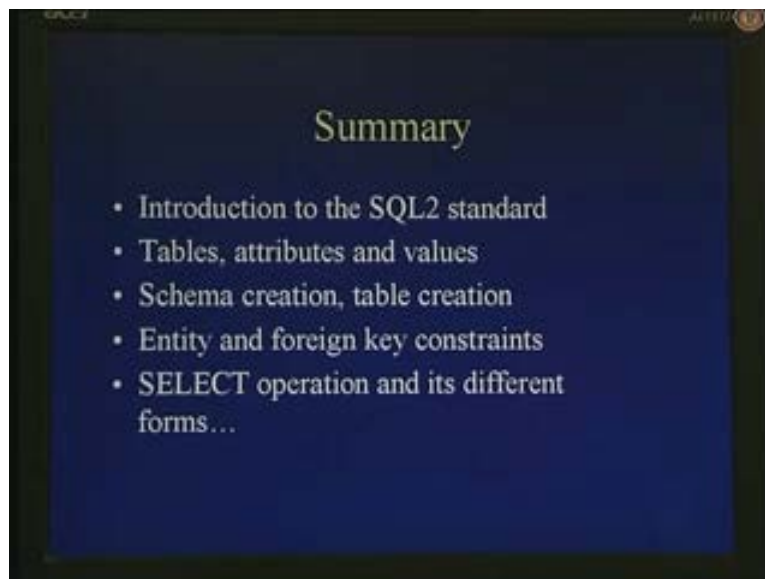
Standard numeric operators like addition (+), subtraction (-), multiplication (*) and division (/) can be applied to attributes having numeric domains.

The operators may also appear as part of the output table:

```
SELECT 1.1 * SALARY
FROM   EMPLOYEE
WHERE  SALARY > 300000;
```

That is I am multiplying existing values of salary by 1.1 and showing that as the result.

(Refer Slide Time: 00:48:21)



Summary

- Introduction to the SQL2 standard
- Tables, attributes and values
- Schema creation, table creation
- Entity and foreign key constraints
- SELECT operation and its different forms...

So this brings us to the end of the first session on the structured query language where we have looked in to the basics of what makes up the structured query language. And we have seen how to create a schema using SQL and what is a catalog that is a collection of tables and how to specify the structure of a table by specifying the name, the attribute names, the attribute domains, the constrains on the attributes like not null, unique, default values and so on and the key constraints like the primary key which identifies what is the

primary key in this and also referential integrity constraints like using the foreign key constraint.

We have also seen how to alter schemas and table constructs or table structures and what are the implications of these constraints on these modifications? That is what happens if I drop a particular column name but that column name is actually referred to as a foreign key from some other table. Now in such cases I can also specify whether to drop all foreign key references or to drop this column only if there is no foreign key reference coming into the table.

So using either the notation of cascade or restrict. We also saw the most widely used operation in relational algebra namely the select operation. However we have not finished looking into the different forms of select operators. In the next session we would be looking at some more features of select as and how to nest select operators and how do we dereference or how do we disambiguate attribute names in the nested select operators. However as we have seen because select is the most widely used SQL operator or SQL statement, it has varied number of forms and several different notations and one of the main properties that we have to remember about the select operator is that the select operator treats tables as bags or as multi-sets rather than as sets that is it tolerates duplicates in the sets unless we specify explicitly that we do not want duplicates and this is specified by the distinct clause.

So to summarize, the slide here shows the summary of what we have seen today. We have seen an introduction to the SQL standard tables, attributes and values and we saw how schemata are created and tables can be created and constraints and essentially the select operation in its different forms that brings us to the end of this session. Thank you.