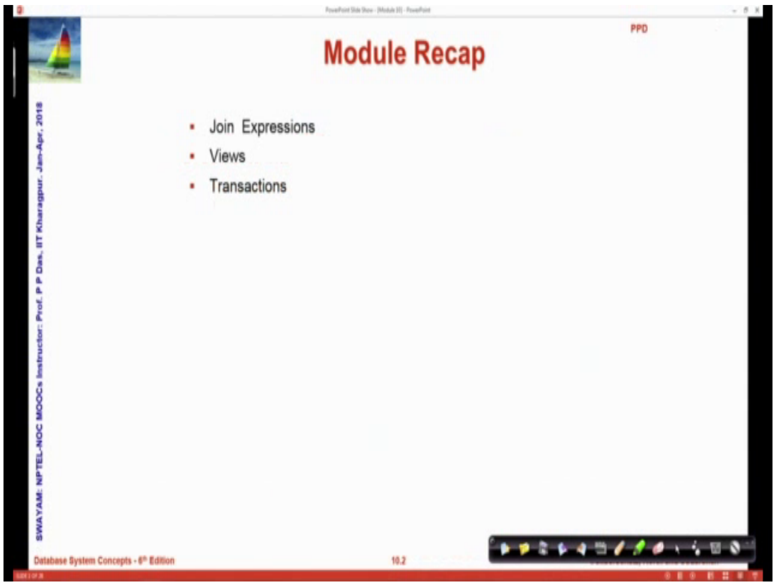
Database Management System
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 10
Intermediate SQL/2

(Refer Slide Time: 07:37)



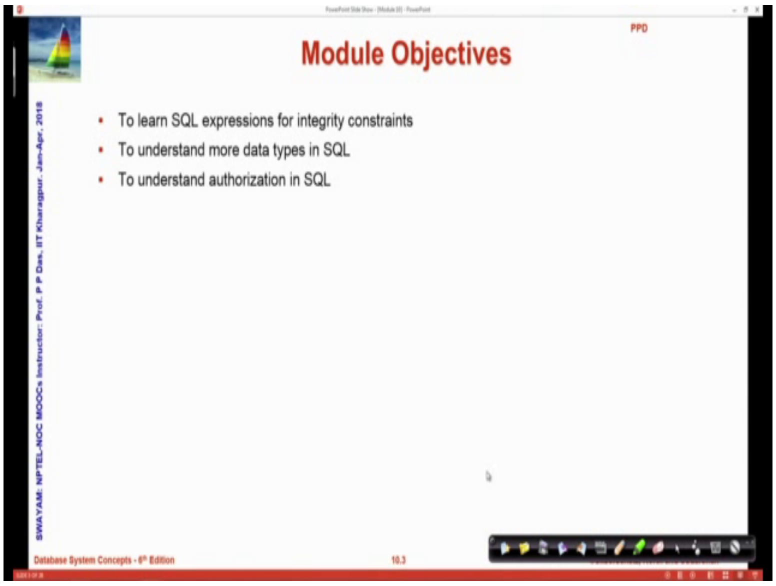
Module Recap

- Join Expressions
- Views
- Transactions

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
10.2

Welcome to module ten of database management systems. We have been discussing about intermediate level features in SQL; and this is a second and closing module on that. So, we have in the last module talked about join expressions, views and transaction in a bit.

(Refer Slide Time: 07:46)



Module Objectives

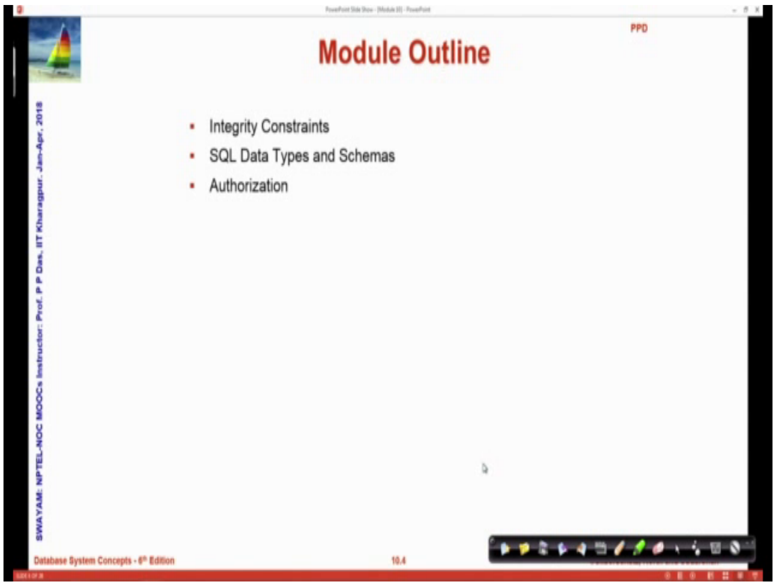
- To learn SQL expressions for integrity constraints
- To understand more data types in SQL
- To understand authorization in SQL

PPD
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
10.3

In this module, we will try to learn SQL expressions that are responsible for maintaining in the integrity of the database. We have talked about integrity a little. We will now see how explicitly integrity can be checked and how different kinds of integrity can be ensured through SQL. We will also talk about more data types.

We have seen the basic primitive data types, and we had promised that we will talk more about the data types including user defined data types here. And finally, we will talk about a very important aspect of authorisation as to who can do what in a database in through SQL.

(Refer Slide Time: 08:27)



Module Outline

- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

PPD

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.4

So, this is a module out line.

(Refer Slide Time: 08:30)

PPD

•Integrity Constraints
•SQL Data Types and Schemas
•Authorization

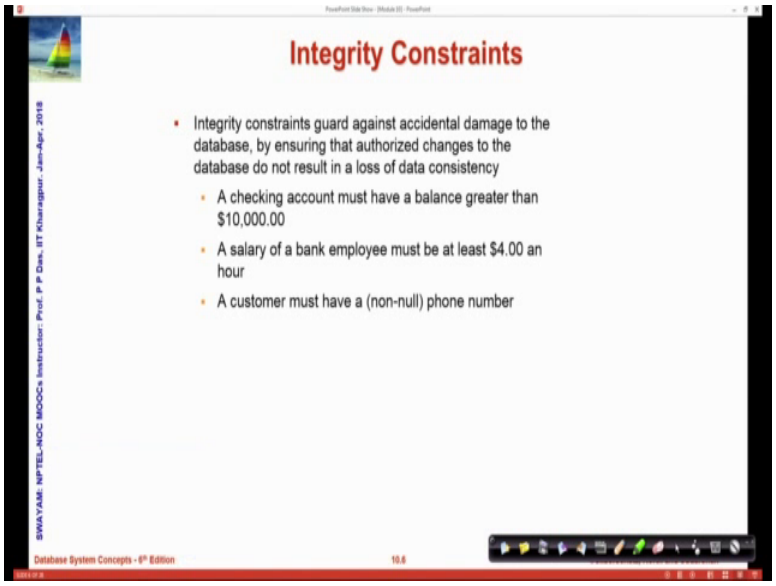
INTEGRITY CONSTRAINTS

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.5

We start with integrity constraints.

(Refer Slide Time: 08:31)



Integrity Constraints

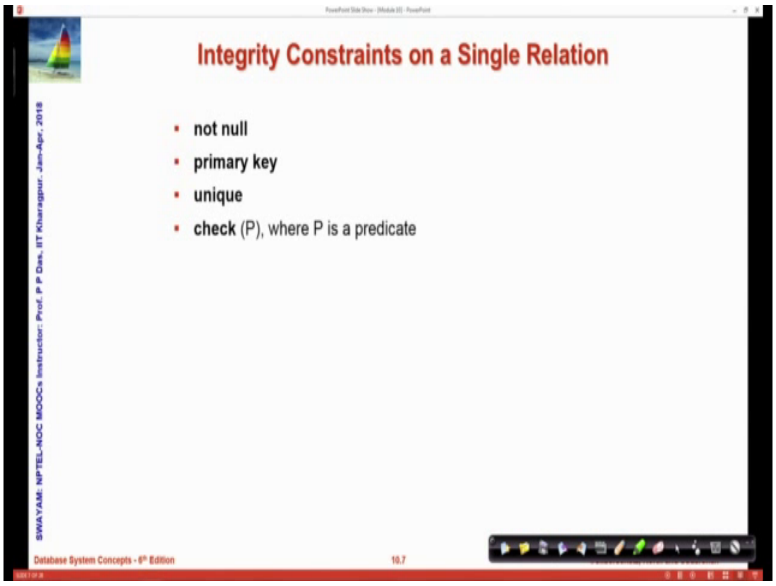
- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.6

Integrity constraints guard against accidental damage to the database that is there are certain real world facts that must be ensured in the database all the time. For example, in bank accounts, we have a minimum balance that need to be maintained; for a particular customer, we might want that the customer's phone number must be present. We may have certain age bar in terms of entering into certain memberships or certain employment and so on. All these kinds of real world constraints need to be represented and maintained in the database and that is the purpose of the integrity constraint.

(Refer Slide Time: 09:16)



Integrity Constraints on a Single Relation

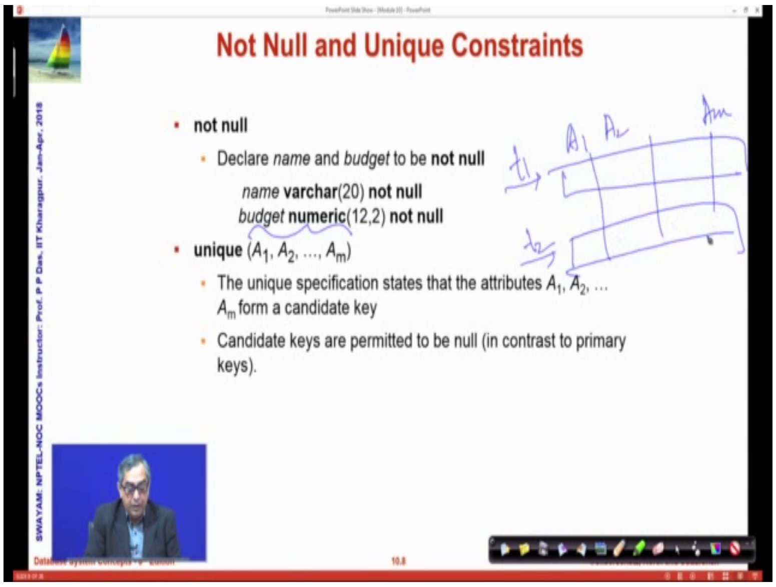
- **not null**
- **primary key**
- **unique**
- **check** (P), where P is a predicate

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.7

And we will first look at the issues of integrity constraint for a single relation, and we have seen the use of not null and primary key. We will also talk about what is unique and we will see how actually general constraints can be checked in terms of a check clause with a predicate P.

(Refer Slide Time: 09:43)



Not Null and Unique Constraints

- **not null**
  - Declare *name* and *budget* to be **not null**

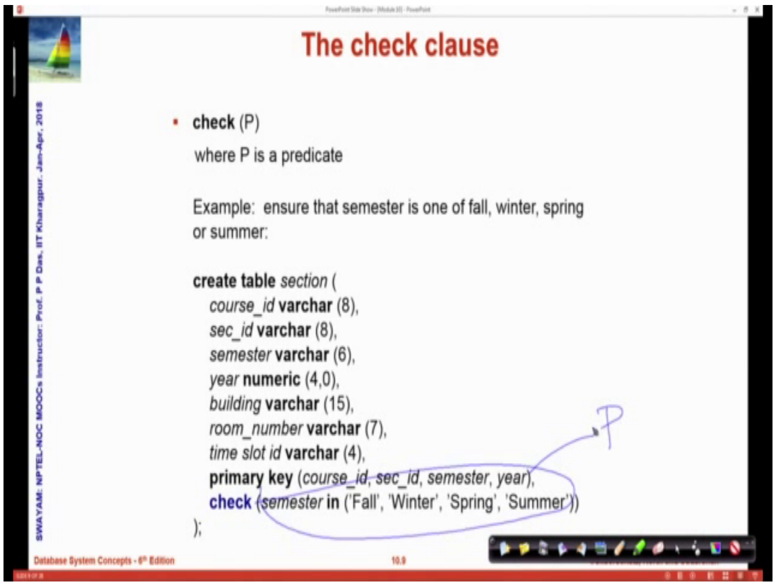
    name **varchar**(20) **not null**
    budget **numeric**(12,2) **not null**
- **unique** ($A_1, A_2, \ldots, A_m$)
  - The unique specification states that the attributes $A_1, A_2, \ldots A_m$ form a candidate key
  - Candidate keys are permitted to be null (in contrast to primary keys).

A_1
A_2
A_m
t_1
t_2
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

So, not null we had seen this before we can while creating the database table. Create table we can specify a field to be not null and then in that case null values will not be allowed in those fields**.** So, they will must be some value given. You can say one or more attributes to be unique. If you specify them to be unique; that means, that in any instance of the table in future that cannot be two tuples which match in all of those attribute.

So, if you say A 1 to A m are unique; that means, that if we have two different tuples in the table anytime in future. Two different rows in the table t 1 and t 2 then across A 1, A 2, A m they must differ in at least one attribute value. So, uniqueness is a basic requirement for being a candidate key, but they are, but still permitted to be null which in contrast to what is the true for primary key already you know that primary key values cannot be null, but uniqueness allows a null values, but they have to be different.

(Refer Slide Time: 10:58)



The check clause

- **check** (P)
  where P is a predicate

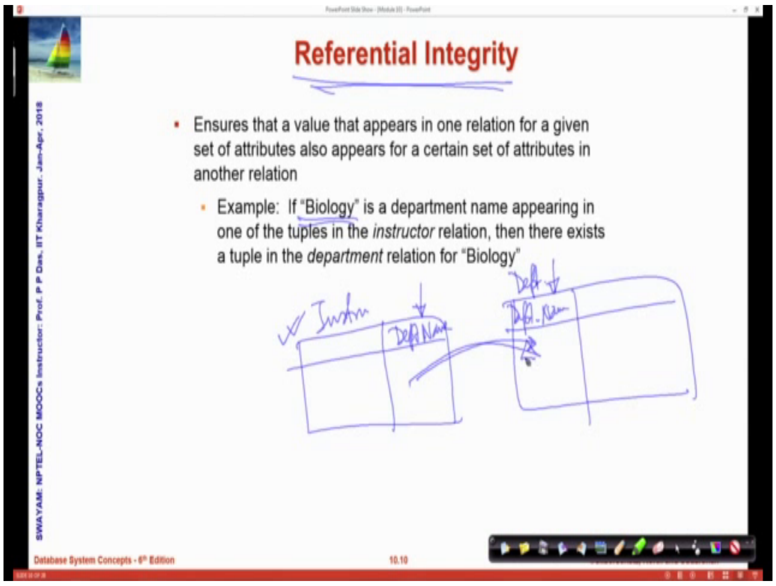
  Example: ensure that semester is one of fall, winter, spring or summer:

```
create table section (
    course_id varchar (8),
    sec_id varchar (8),
    semester varchar (6),
    year numeric (4,0),
    building varchar (15),
    room_number varchar (7),
    time slot id varchar (4),
    primary key (course_id, sec_id, semester, year),
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```

The check clause is where you say check and then you put a predicate. So, the idea is like this suppose I know that I have specified and attribute called semester. It is a varchar 6 which means that it can have a string maximum of length 6, but naturally I can write anything there I can write morning in that field. I can write welcome in that field and so on, but those are not valid names of semester. So, I want that in my design semester must have any of these values only.

So, we say semester in. So, I have listed the values that are allowed in is a set membership. So, it says the semester in so which means the value of the semester is be one of this fall. And this whole thing now becomes the predicate P which on which I give a check which means that whenever I am creating once you have created the table when I want to insert or update the values in the table, the records in the table then e the value of semester has to be always within this. Otherwise, the check integrity constraint will fail, and the update or insert will not happen and an exception will be raised. This is the basic idea of check constraints.

(Refer Slide Time: 12:25)



Referential Integrity

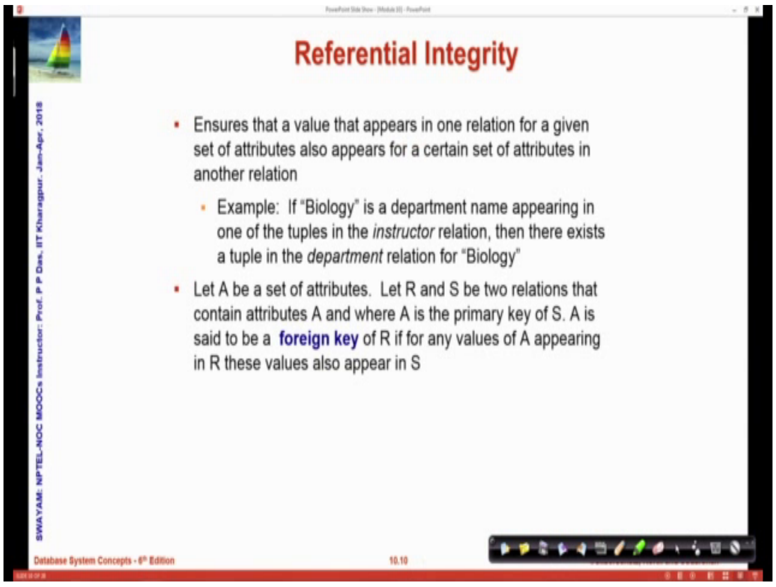
- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
  - Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology"

Instr
Dept Name
Dept
Dept. Name
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
10.10

Now, let us move on to more involved integrity check which goes beyond one table. So, let us suppose that we are talking about the instructor table we have the instructor table. An instructor table has a department name. Similarly, we have a department table which naturally has a department name. Now, we know that this is the key in the department table and therefore, here it is the foreign key.

Now, while we are inserting records in the instructor table how do we guarantee that the record that we insert has a corresponding entry in the foreign key table that is difference table. So, it is when we are inserting and in a faculty in the saying that the faculty belongs to biology department there needs to be a biology entry in the department table as well. So, this is known as a referential integrity that is once you refer from one table to the other that reference must also be a valid one; otherwise, all your computations will go wrong.

(Refer Slide Time: 13:48)



Referential Integrity

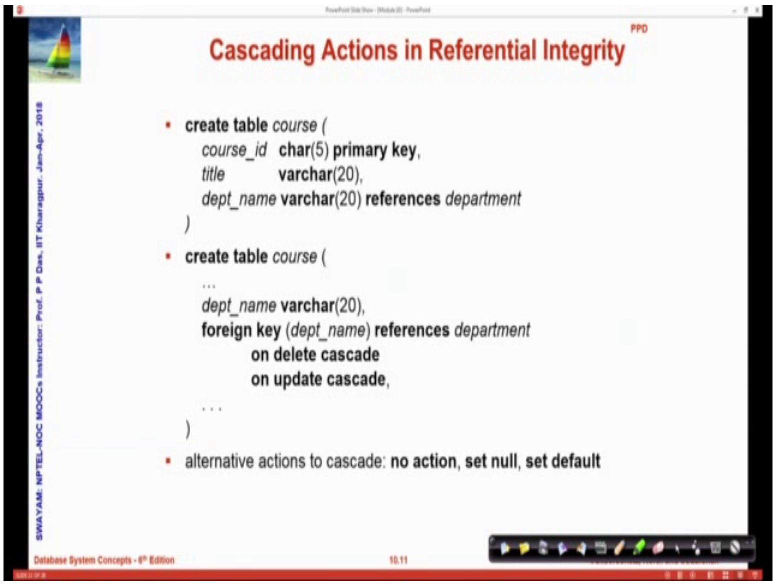
- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
  - Example: If "Biology" is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for "Biology"
- Let A be a set of attributes. Let R and S be two relations that contain attributes A and where A is the primary key of S. A is said to be a **foreign key** of R if for any values of A appearing in R these values also appear in S

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.10

So, this is a for saying it formally there are two relations and one relation as a primary key which is used in the other relation as a foreign key then there is a referential integrity that needs to be maintained.

(Refer Slide Time: 14:02)



Cascading Actions in Referential Integrity

PPD

- **create table** *course* (
```
    course_id  char(5) primary key,
    title      varchar(20),
    dept_name varchar(20) references department
)
```
- **create table** *course* (
```
    ...
    dept_name varchar(20),
    foreign key (dept_name) references department
            on delete cascade
            on update cascade,
    . . .
)
```
- alternative actions to cascade: **no action**, **set null**, **set default**

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

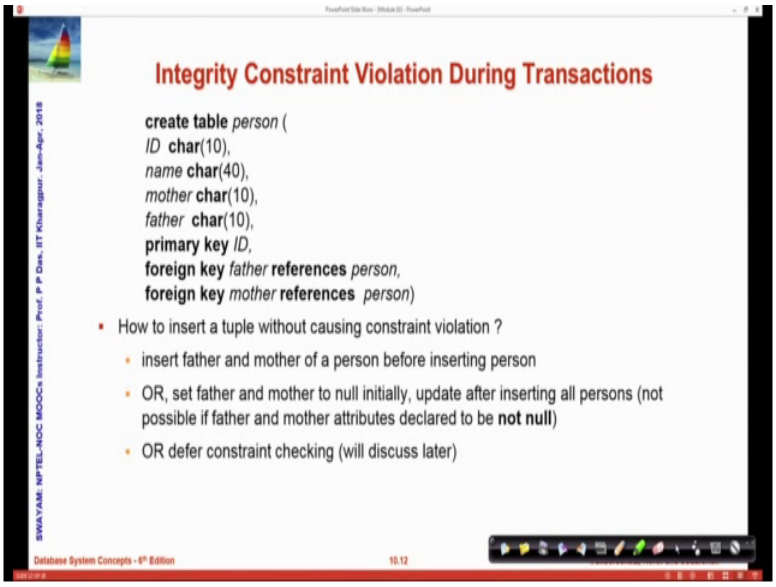
Database System Concepts - 6th Edition 10.11

So, here we are just showing the effect of that. So, we have created a table this is the first one is what you have seen earlier creating the table course and that table course needs the name of the department. So, we are specifying that it references the department table. Now, if it references the department table it must ensure the referential integrity. So, this just says that this refers to the department table, but I can be more specific to say what will happen if the integrity gets violated. For example, I have created this and the course table has an entry, which has a department name say biology. Naturally, biology department should have the entry there should be an entry in the department table with this department name biology for this to be valid.

Now, say for some reason the biology department is abolished and that particular record from the department table is removed, naturally, the course which is referring to biology in terms of its department name that particular record will become invalid. So, we can say that on delete, what you should be doing, one most common action that we specify in referential integrity is cascade that if the referred entity is deleted then the referring entity should also be deleted. So, if you delete the biology entry from the department table, then all courses which have biology through references to department as their field value should also get deleted.

Similar thing can be there on update also. For example, biology department say tomorrow changes the name to bioscience. Now, if I have a referential integrity put on the course table as on update cascade, then as I change the bioscience the name to bioscience. All records in the table course, which had the department name as biology will necessarily get updated. So, this is the way to maintain referential integrity. Cascading is one of the most common way to handle this, but there could be other ways to take action also that could be no action that you say ok, I do not care. Let that happen in that case some, because of the violation that could be some exceptions thrown or even say that if this happens then I will set that field to null or I will set that field to some default value and so on.

(Refer Slide Time: 17:01)



Integrity Constraint Violation During Transactions

```
create table person (
ID char(10),
name char(40),
mother char(10),
father char(10),
primary key ID,
foreign key father references person,
foreign key mother references person)
```

- How to insert a tuple without causing constraint violation ?
  - insert father and mother of a person before inserting person
  - OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be **not null**)
  - OR defer constraint checking (will discuss later)

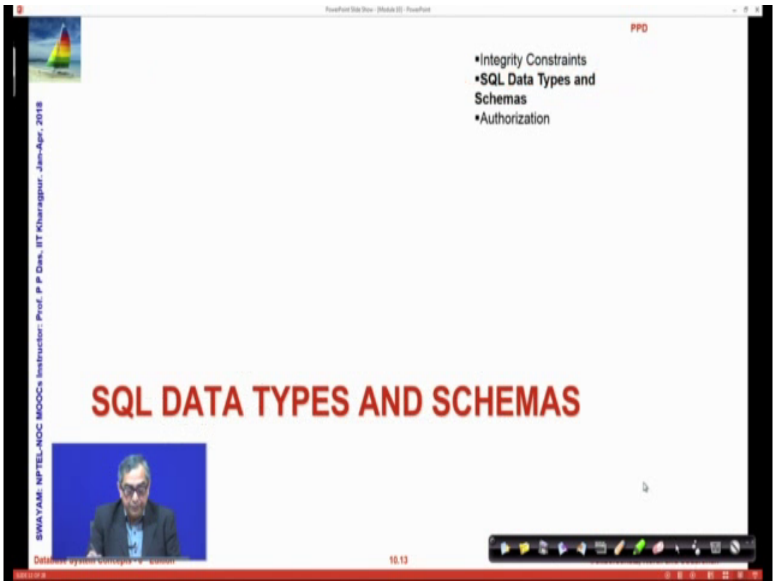
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.12

So, this is how the referential integrity has to be handled. There could be integrity violation during transactions also. This is an example of a self-referential table which of persons which where every person's entry needs the name of the mother and the father which are also entries in this table. So, necessarily if you are entering a person record you need this feels to be populated and that can be populated only if those records already exists.

So, there is some order in which you have to enter the records or you have to set them as null and then update them in future and or some ways to say that well do not check this integrity now. We will talk about this integrity at a later point of time. So, these are the issues that necessarily you will have to be addressed.

(Refer Slide Time: 17:54)

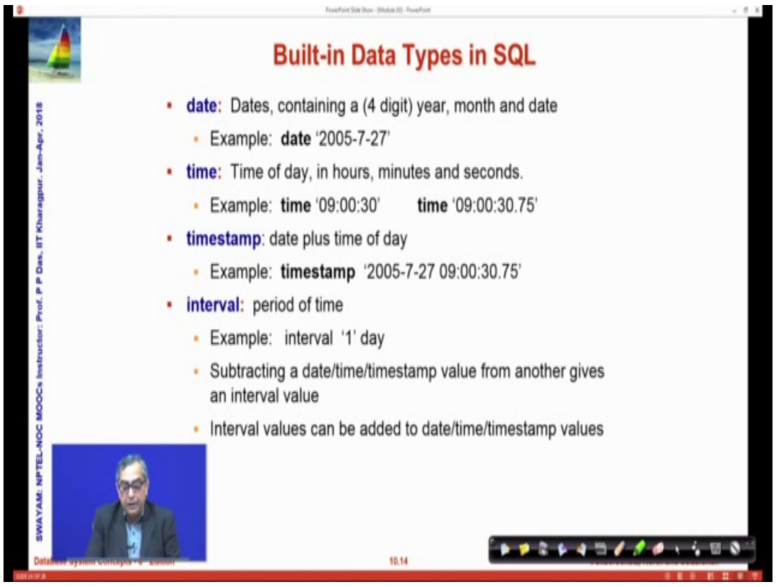


- •Integrity Constraints
- •**SQL Data Types and Schemas**
- •Authorization

SQL DATA TYPES AND SCHEMAS

Let us move on and look at the SQL data types and schemas.

(Refer Slide Time: 18:02)

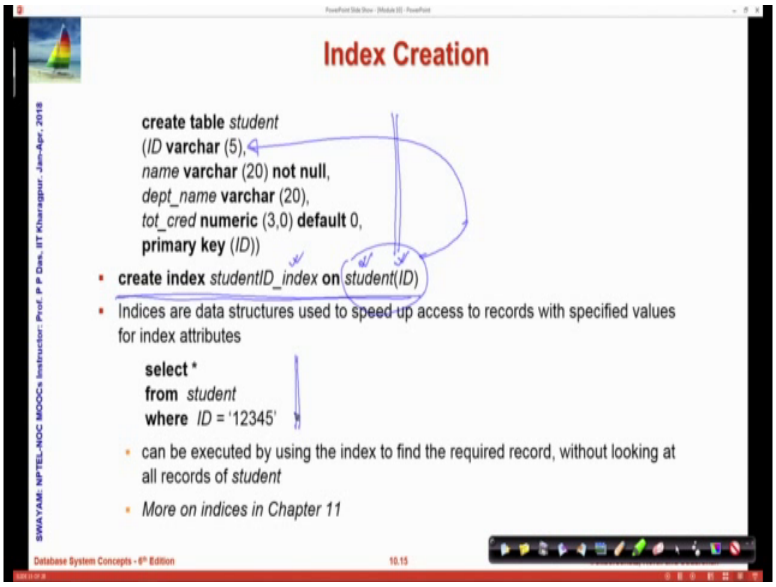


Built-in Data Types in SQL

- **date**: Dates, containing a (4 digit) year, month and date
  - Example: **date** '2005-7-27'
- **time**: Time of day, in hours, minutes and seconds.
  - Example: **time** '09:00:30' **time** '09:00:30.75'
- **timestamp**: date plus time of day
  - Example: **timestamp** '2005-7-27 09:00:30.75'
- **interval**: period of time
  - Example: interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

So, in addition to the data types like char, varchar, int and all that you have an explicit date data type which gives you a year, month, date kind of format with a four digit year, because date is very frequently required. You have a time type to give you hour, minute, second time format. You have a timestamp, which is date and time together and you have what is known as interval where you can do a date or time difference between two different dates, two different time, two different time stamps and so on. So, these are the common added built in types which makes it very easy to handle the temporal aspects in SQL queries.

(Refer Slide Time: 18:54)



Index Creation

```
create table student
(ID varchar (5),
name varchar (20) not null,
dept_name varchar (20),
tot_cred numeric (3,0) default 0,
primary key (ID))
```

- create index studentID_index on student(ID)
- Indices are data structures used to speed up access to records with specified values for index attributes

```
select *
from student
where ID = '12345'
```

  - can be executed by using the index to find the required record, without looking at all records of student
  - More on indices in Chapter 11

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

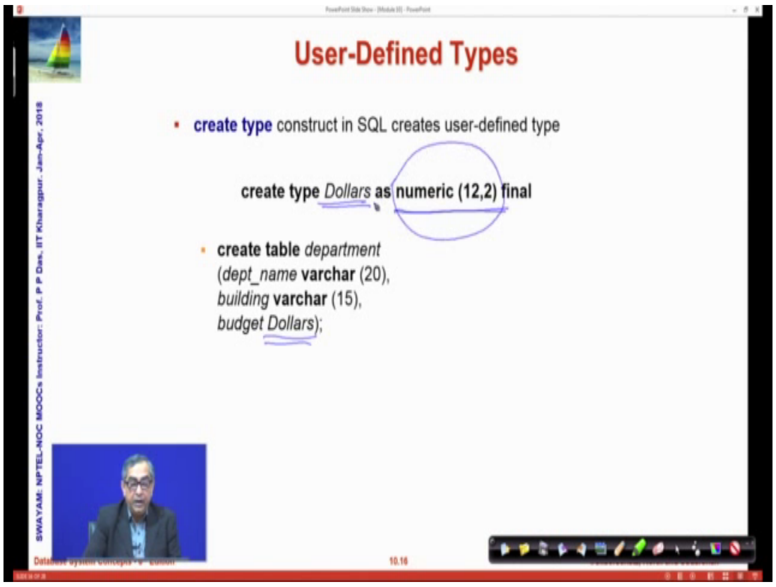
Database System Concepts - 6th Edition 10.15

In addition, the next that you can do is you can create an index. So, let us look at this. So, this create table definition you understand well by now. You can I can do this, I can say create index and give a name for the index and specify which field on which the index should be created. So, here we are saying that the index should happen here. This is the name of the relation, this is the name of the attribute name of the field. Now, this does not change any data neither does it change any schema, but it creates certain additional structure. So, that it becomes easier to search this particular table using IDs.

So, if I have a query like this that I am trying to find out all information about a particular student then as we have said that by default the different entries the rows of a relation are unordered. So, the only way to find out this particular row and in fact, whether it actually exist would be to go over all the relations one by one. But if we index it, then it creates some kind of an efficient data structure through which it can be searched out very efficiently very easily with a later module we will talk about index in. But just to give you the idea that this is similar to finding out a value in an unordered array if you are thinking of C.

In contrast, I can we all know that this can be done, but takes a whole lot of time it takes order in time. But I could keep those numbers in terms of a some binary search tree, balanced binary search tree like red black tree or two three four tree kind of, where the search can be conducted in a login time. Or I could keep it in terms of some efficient hashing mechanism where the search could happen in terms of an ordered one time also. So, index is has a has a lot of importance and we will talk about that more, but this is how you create index in SQL.

(Refer Slide Time: 21:18)



User-Defined Types

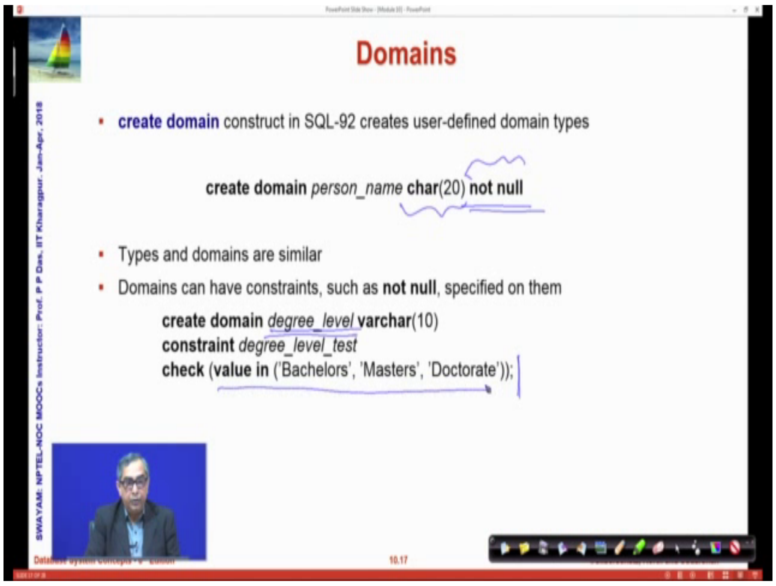
- **create type** construct in SQL creates user-defined type

  create type *Dollars* **as numeric** (12,2) **final**

- **create table** *department*
  (*dept_name* **varchar** (20),
  building **varchar** (15),
  budget Dollars);

You can have user defined types, you can say create type and use some specific you know sub types of a type as and give it a name. So, you send numeric 12, 2. So, which is the 12 digit number with two decimal places of precision you can call it dollar and then use that as a type name. So, type name doing this helps in to visit make sure that wherever you actually have to conceptually refer to dollars you are talking about dollars it is easier to understand and you are making sure that everywhere the same numeric precision is used.

(Refer Slide Time: 22:02)



Domains

- **create domain** construct in SQL-92 creates user-defined domain types

```
create domain person_name char(20) not null
```

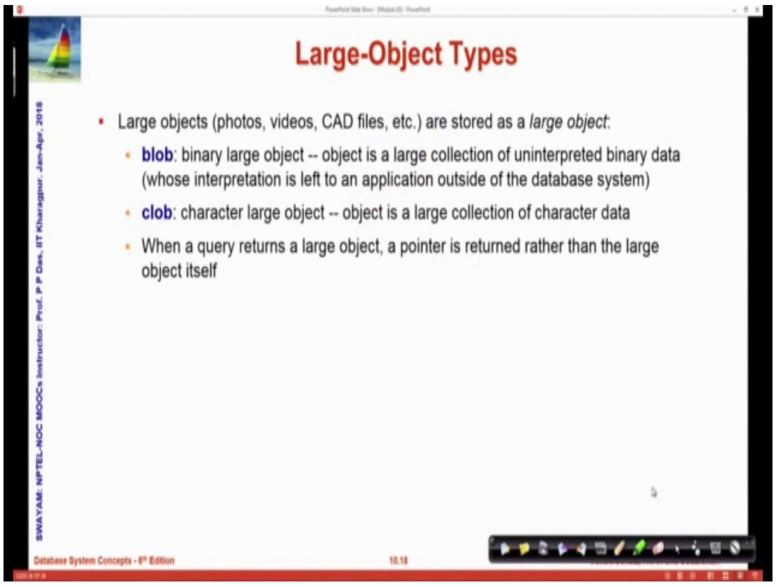
- Types and domains are similar
- Domains can have constraints, such as **not null**, specified on them

```
create domain degree_level varchar(10)
constraint degree_level_test
check (value in ('Bachelors', 'Masters', 'Doctorate'));
```

You can also actually go further and create domains which is very similar to create type. But domains are more powerful in the sense that in a domain. You can also add constraints like not null and you say that this is person name see say that this once you have set that this person name is 20 characters long and it cannot be null then you do not specifically have to every time.

You define a field based on this domain type you do not have to specifically say that it is not null. You could also create a specific constraint in terms of the check clause and make it easier. So, now if you say degree level you do not have to put check clause explicitly in the SQL query, because it is already specified in the created domain.

(Refer Slide Time: 22:52)



Large-Object Types

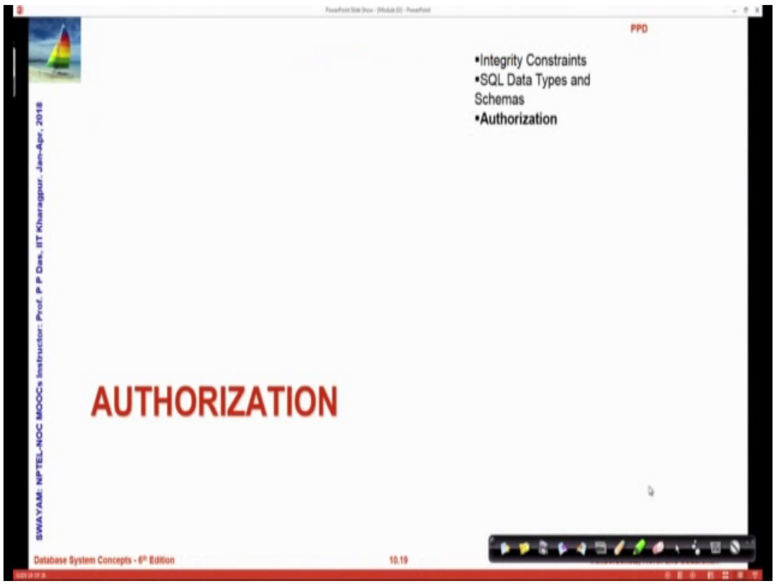
- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - **blob**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - **clob**: character large object -- object is a large collection of character data
  - When a query returns a large object, a pointer is returned rather than the large object itself

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.18

SQL supports certain large-objects which are either called blobs if they are binary, or called clob if they are character objects. The only the major difference in terms of the large object types are they are not stored as a part of the table. They stored elsewhere and you actually maintain a kind of a reference a pointer to that large object. So, this is very useful in terms of handling photos, videos and no big binary files, character files also.

(Refer Slide Time: 23:21)



PPD

•Integrity Constraints
•SQL Data Types and Schemas
•**Authorization**

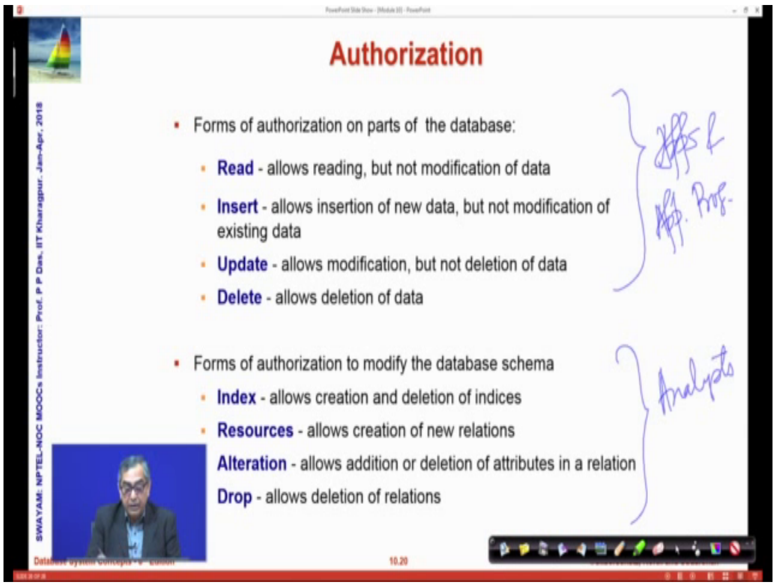
AUTHORIZATION

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.19

Let us move to authorization.

(Refer Slide Time: 23:29)



Authorization

- Forms of authorization on parts of the database:
  - **Read** - allows reading, but not modification of data
  - **Insert** - allows insertion of new data, but not modification of existing data
  - **Update** - allows modification, but not deletion of data
  - **Delete** - allows deletion of data
- Forms of authorization to modify the database schema
  - **Index** - allows creation and deletion of indices
  - **Resources** - allows creation of new relations
  - **Alteration** - allows addition or deletion of attributes in a relation
  - **Drop** - allows deletion of relations

Apps &
App. Prog.
Analysts
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
10.20

Next, authorization is the process by which you restrict different users to be able to do different kind of operations. You recall in the early modules on database overview. We mentioned that there could be several types of users for a database. There could be absolutely application users who may necessarily do not feature as a part of the database development. But there could be application developers expectedly most of you would become application developers or there could be intermediate higher level of analyst who design databases, design constraints, decide on indices and so on and that could be database administrator.

And also in terms of different application programs and programmers there is a need to separate out who can access which part of the database. For example, if you look at a add a banking system, then while I am by net banking application is accessing different information about my account, one part I need to ensure that I can only access my account.

And also what the database system needs to ensure is that a net banking application should in no way be able to access the information about the specific employees, because, in the same database information about the bank employees will also be there. It should not be possible for possible to access information about different physical information about the branches as to where, how many square feet of area that branch has and so on and so forth.

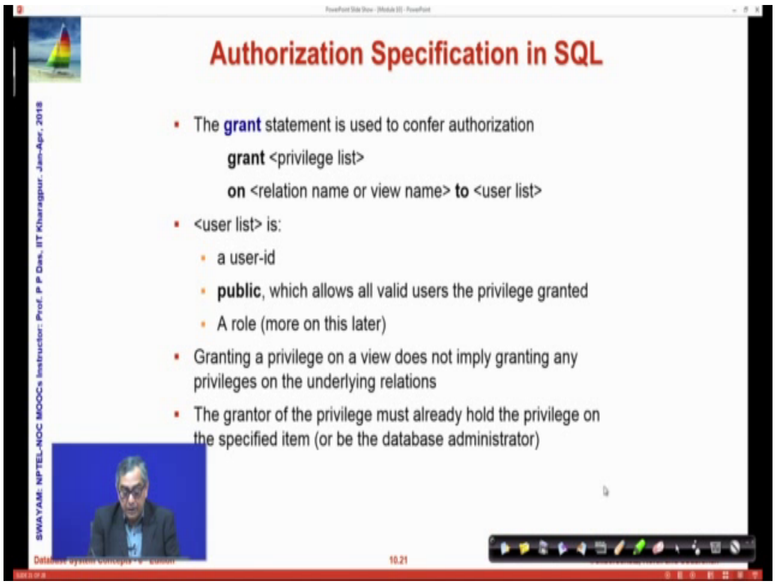
So, we need to put variety of restrictions and as we will see that authorisation or this process of restricting or allowing different access and different authority to operate is decided based on two different factors. One is what you want to do, and two is who wants to do that. So, what and who, so we identify different operations or different operations on certain tables or operations on certain attributes as what needs to be done. And on the other side, we will identify who in terms of specific individual user IDs or groups of user IDs or roles that exist.

So, here we will just try to show you how we can do that in SQL. So, the first part of the authorisation is being able to do different things with the database that means, the instances of the database. So, there are authorisations to read insert, update and delete. So, read is where you can access the data, but you cannot modify; insert is when you can add new data, but you do not with insert rights authorisation, you cannot update an existing data, you can only insert data. You can have update rights, where we can change make modifications, but you may not be you are not allowed to delete data, and you can that would be delete right where it allows you to delete data and my new this authorisations or not these are all independent authorisation.

So, you may have I mean certain authorisations may need certain other authorisations to be present. For example, if you are updating naturally evenly to read, but it is these are all independent authorisations, and you may have one or more of them to be able to do the appropriate actions. Similar set of another set of authorisations will exist, if you want to for those want to modify the database schema, naturally, this is primarily for the applications and application programmers, and this primarily would be for the analysts that you can index the different table you can do.

You can have authorisation for resources which mean you can create new relations, create new schemas, you can alter schemas, you can drop schemas and so on. So, these are the different kinds of authorisation that are possible.

(Refer Slide Time: 28:01)



Authorization Specification in SQL

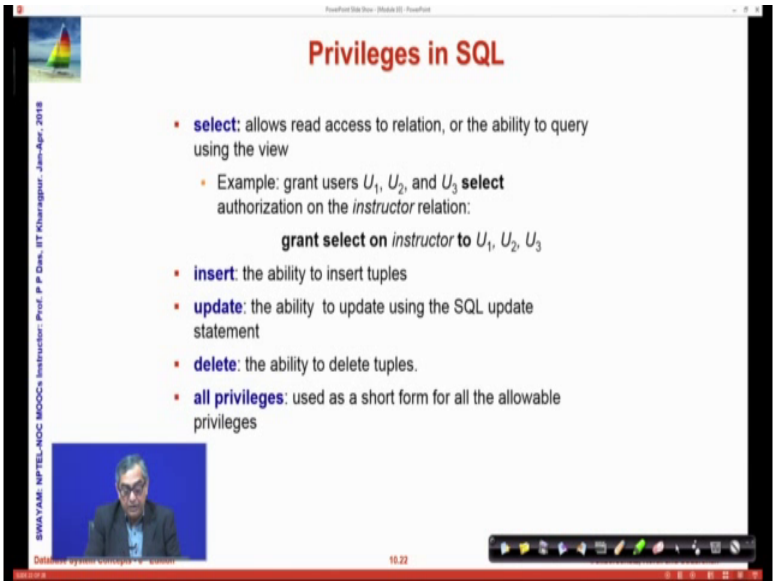
- The **grant** statement is used to confer authorization
  grant <privilege list>
  on <relation name or view name> **to** <user list>
- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator)

So, let us see how it works the authorisation is specified in terms of a statement called grant. So, you grant an authorisation to a privilege list and on certain relation to a group of users. So, grant what kind of authorisation you are granting that is the privilege less list on what relation on view you are granting that is on condition and to whom are you granting those. So, user list could be a specific user ID or you could say public which in this case everybody will have that or this could be a role which will see, what a role is.

Granting a privilege on a view does not imply granting any privileges on the underline relation, please mind this one, because, you have seen that a view can be formed from multiple different relations. So, if somebody has been granted a particular privilege say read privilege on a view, then it does not mean that the corresponding underline relationship, you been granted a read privilege on faculty relation that we faculty view that we did that does not mean that the user will automatically get a read privilege on the underline in structure relation. So, that has to be kept in mind.

The grantor of the privilege must already hold the privilege that is you cannot grant, naturally, grant will be done also by somebody in some of the users you may be, so that user who is granting must also have the privilege same privilege on the specific item. So, you cannot grant privilege on something, some relation or view on which you yourself do not have that or it has to be the database administrative who naturally has privilege for everything.

(Refer Slide Time: 30:13)



Privileges in SQL

- **select:** allows read access to relation, or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

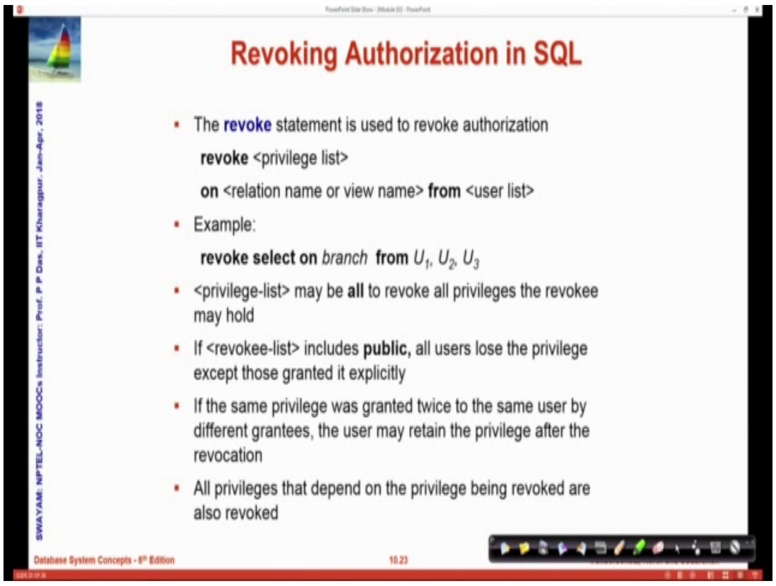
    grant select on *instructor* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
10.22

The privileges on SQL there is a select privilege, which is so you know this is the privilege list. Select privilege which basically means read access. So, it is saying select privileges is read access on instructor and these are the different users. So, this is how typically, you can have insert to ability to insert tuples, the update privilege this should be easy to understand now delete privilege.

So, only thing is read is a called select here, so these are the different privileges that you have in a SQL. And you have one all in compressing privilege which is called all privileges so as a short form of allowing all these allowable privileges.

(Refer Slide Time: 31:00)



Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization

  revoke <privilege list>

  on <relation name or view name> **from** <user list>
- Example:

  revoke select on *branch* **from** U_1, U_2, U_3
- <privilege-list> may be **all** to revoke all privileges the revokee may hold
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation
- All privileges that depend on the privilege being revoked are also revoked

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

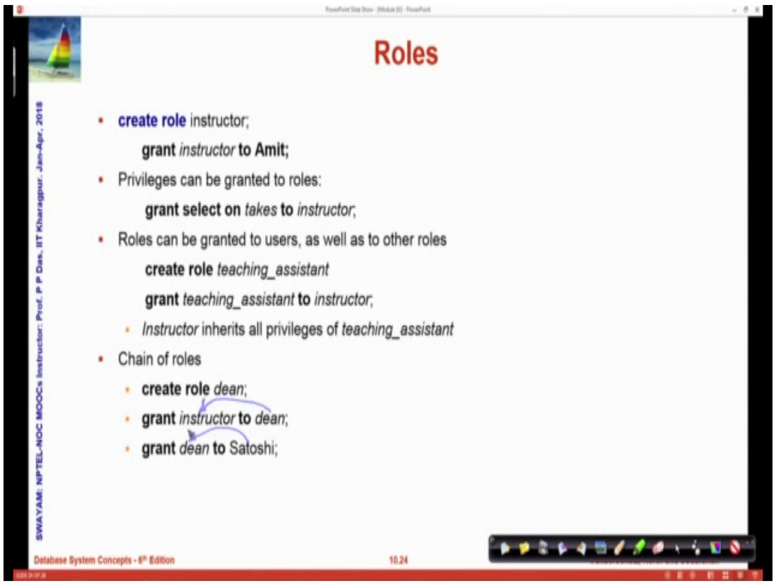
Database System Concepts - 6th Edition 10.23

Certainly, if you can grant an authorization, then needs to be a reverse process that is if we want to withdraw authorization of certain privileges on certain items for certain users, so that is known as a revoke statement. So, we can revoke it the structure looks exactly similar to the grant. So, you revoke a privilege list, select, insert this kind of on certain relation and view from a set of users. So, you can say that revoke select on branch from this. So, once this is done, then U 1, U 2 and U 3 will not be able to read the branch relation or the branch view. The privilege list may be all to revoke all privileges. So, instead of revoking select insert, separately, you can just say all and revoke all of that.

The list of revoking can include public which means all users lose that privilege and or though those were granted. If the same privileges granted twice, now it is possible that a user gets privilege granted to him or her by two different granting authorities, then the if ones is one of them is revoked the other will still continue to remain; So, every privilege that is granted needs to be explicitly revoked that is a basic meaning.

So, all privileges that depend on the privilege being revoked are also revoked. So, some privilege which is dependent on some other privilege, if you revoke the update privilege then this select privilege will remain. But, if you revoke the select privilege then if you also had the update privilege that will certainly get revoked, because if you cannot read then naturally you cannot change.

(Refer Slide Time: 33:01)



Roles

- **create role** instructor;
  - **grant** *instructor* **to Amit;**
- Privileges can be granted to roles:
  - **grant select on** *takes* **to** *instructor;*
- Roles can be granted to users, as well as to other roles
  - **create role** *teaching_assistant*
  - **grant** *teaching_assistant* **to** *instructor;*
  - *Instructor* inherits all privileges of *teaching_assistant*
- Chain of roles
  - **create role** *dean;*
  - **grant** *instructor* **to** *dean;*
  - **grant** *dean* **to** Satoshi;

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.24

The SQL also allows you to create certain roles. Roles are kind of like virtual use that so, we all say that we all play certain role. So, I have an entity as an individual say I may be user called P P B, but I have a role as an instructor, I have a role as a say the head of the department, I have a role as a chairman of committee and so on. So, often times it becomes easier to grant privileges to different roles.

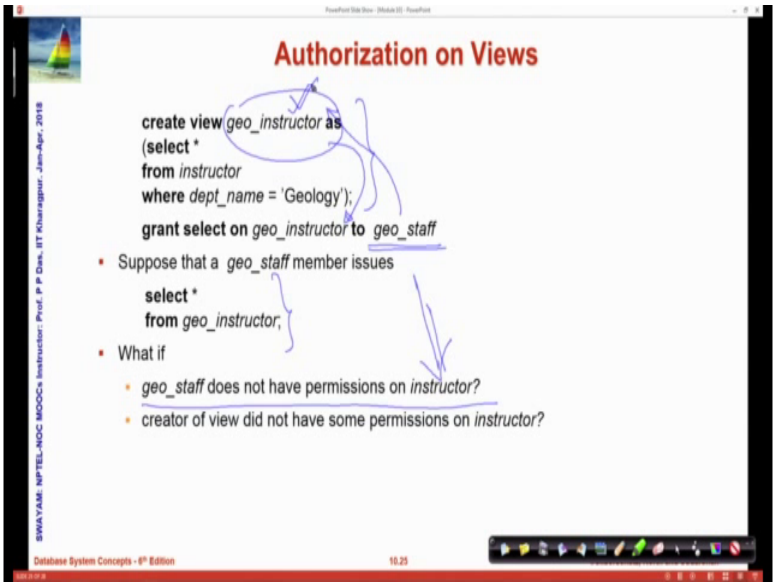
You do not really care immediately about who that individual could be who that particular user could be who has that privilege, whoever plays that role gets that privilege, whoever becomes the director of IIT Kharagpur has the privilege to appoint faculty members it is of that kind. It does not specifically. So, role is of that kind of a concept. So, you create role we are saying that role is the role instructor is created.

And then you are saying that you grant instructor to Amit which says that Amit now plays the role of instructor. So, any privilege that the instructor role has Amit will enjoy that. So, let us see more of this the privileges can be grant to roles. So, earlier we said it could be public, it could be users, but now you are saying that it could be two roles. So, here this role was creates and the privilege is being granted to that. And since Amit plays that role it will mean that Amit with this grant select on takes to instructor, Amit will actually get a privilege of select on takes relation that is the kind of derived structure that roles give you roles can be granted to users as well as to other roles.

So, roles are becoming like virtual users. So, you can create role teaching assistant and grant teaching assistant to instructor, which means that if you do that you are granting this. So, it means that any privilege the teaching assistant will have the instructor will get those privileges, because, you have made in instructor to also play the role teaching assistant mind you instructor itself is virtual entity. So, if Amit is an instructor by this, then Amit plays this role and this role plays teaching assistant role and this teaching assistant role has certain privileges, so naturally through this chain process Amit will get those privileges. So, in an instructor inherits all privileges of teaching assistant.

So, this is what exactly, what I was talking of you can have a chain of roles create a role dean new one, you have created, then grant instructor to dean grant dean to Satoshi. So, which means; that once you grant dean to instructor; so anybody who plays the dean's role will get all privileges of instructor. Here you are saying that Satoshi is going to play the dean role. So, the Satoshi in terms of chaining gets all the privileges that instructor has.

(Refer Slide Time: 36:41)



Authorization on Views

```
create view geo_instructor as
(select *
from instructor
where dept_name = 'Geology');
grant select on geo_instructor to geo_staff
```

- Suppose that a geo_staff member issues

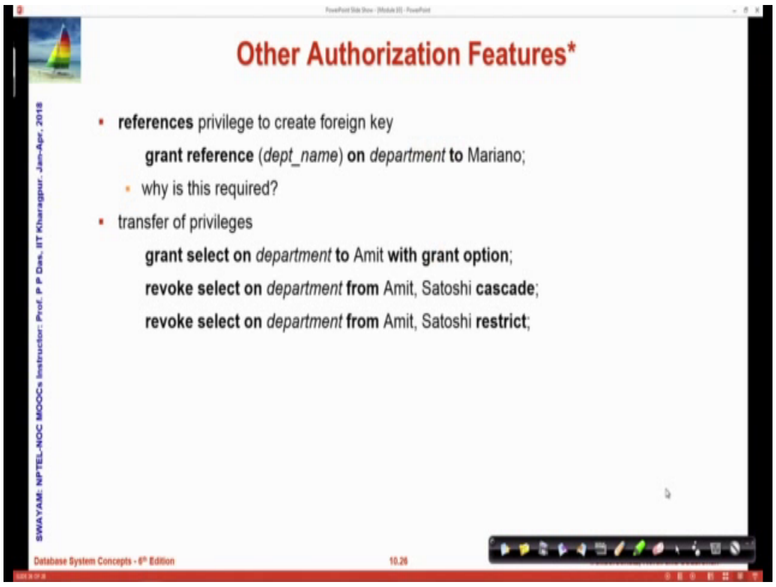
```
select *
from geo_instructor;
```

- What if
  - geo_staff does not have permissions on instructor?
  - creator of view did not have some permissions on instructor?

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
10.25

So, once this has been done, then you can have authorization on views as well. So, you have created a view here. So, this is the view created the geo instructor the Geology instructor. And on that view particularly you have given the privilege to the geo staff. So, a geo staff member would be able to access this view. And if this query is fired by a geo staff member, which I am assuming is a role then this view will get executed and the results of all instructors in the department geology will be update. But, what if the geo staff does not have permission on instructor, does not matter that is the beauty of the whole thing. The geo staff may not have permission to do select on instructor, but the geo staff has permission to select on geo instructor. So, the geo staff will be able to execute this view, but the geo staff will not be able to do a select on from the instructor database instructor table.

(Refer Slide Time: 38:16)



Other Authorization Features*

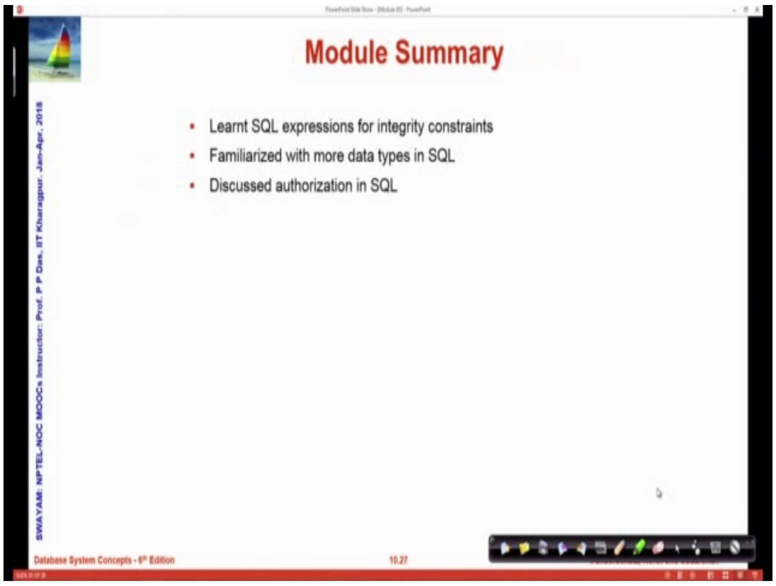
- **references** privilege to create foreign key
  - **grant reference** (*dept_name*) **on** *department* **to** Mariano;
  - why is this required?
- transfer of privileges
  - **grant select on** *department* **to** Amit **with grant option**;
  - **revoke select on** *department* **from** Amit, Satoshi **cascade**;
  - **revoke select on** *department* **from** Amit, Satoshi **restrict**;

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
10.26

There are several other authorization features, the references a privilege to create foreign key. So, we talked about basic read, write, data manipulation privileges, but there could be other privileges like whether you can create a foreign key, whether you can transfer of privilege. So, whether you can give one privilege to another, so whether you can cascade, whether you can restrict and so on.

So, transfer of privileges is also privilege. I am naturally will have to think of this is an authorization. So, actually what we can authorize is also a privilege that needs to be authorized. So, these are the derived privileges that I have.

(Refer Slide Time: 39:00)



Module Summary

- Learnt SQL expressions for integrity constraints
- Familiarized with more data types in SQL
- Discussed authorization in SQL

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 10.27

So, in summary, we have learnt about SQL expressions to deal with integrity constraints. We are familiarized with more data types particularly user defined types and domains creation of index and we have discussed about authorization in SQL. Each one of them particularly authorization has lot more details, but at this intermediate level we just wanted to get a basic idea about authorization to be able to deal with that.

So, with this we close our discussion on the intermediate level SQL features. In the next module that we start next week, we will talk about some of the advanced SQL features.