Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 13

Constraints & Triggers

Hello and welcome to another session in database managements systems. We have covered a quite a bit of ground in the explorations of dbms already. We have looked into logical models of data management from the conceptual perspective, from the physical perspective and so on. We also looked at physical requirements of data storage although we have only looked at very specific kinds of physical data storage and retrieval strategies for databases. We also looked at few kinds of index structures using which we can efficiently access or efficiently search and retrieve for the relevant tuple or the relevant record that we are looking for in a database management system.

Usually when we are taking about databases there is sometimes a physiological question that is raised as to whether any collection of data is a database. Especially perhaps in the late 90's there was this ranching debate about whether the web, the world wide web is just a large database. Now of course one might be tempted to believe that any collection of data is a database and some might argue that is not really true, collection of data is just a collection of data. It is something like the difference between a set, a bag and a relation and so on. All of them are just collections of something's, a set of tuples or a bag of tuples or relation of tuples all of them are just a collection of tuples.

However some of them for example sets or bags impose what are called as additional constraints over this collection. A bag of tuple is simply a collection of tuples and there is no constraint what so ever on what should be the tuple, what kinds of tuple should be there, what should be the size of each tuples, what should go into or what should define the attributes that make up the tuples and what about duplicates in the tuples, how many different kinds of the same, how many different numbers of the same kinds of tuples can be there and so on. There are absolutely no kinds of constraints that are imposed on a bag of tuples. However, when we consider a set of tuples there is an implicit constraint that is imposed that a set is a collection of a things of the same kind.

Although not always so but usually we talk about a set as a collection of things of the same kind. Therefore when we say a set of tuples, we tend to believe that it is a collection of things or collection of tuples of the same kind that is all of the tuples have the same number of attributes and each attribute or each corresponding attribute of each tuple has the same domain and so on. And there is also an implicit constraint that there are no repetitions in sets that is two or more tuples cannot have the same value for every corresponding attribute that they contain. Therefore there is an implicit constraint of no duplicates that is imposed on a set of tuples and of course in order to implement this, that is in order to implement a bag of tuples and to implement a set of tuples, the kinds of programming that we have to do, the kinds of logical checks that we have to do, changes.

Implementing a bag of tuples is the easiest. We just maintain one collection of tuples using whatever data structure that we can use for maintaining collections, link list, trees, whatever arrays and so on. Similarly in order to implement a set of tuples we can use the same kind of data structure in order to implement a collection. However we have to keep making checks so that the set property of this tuples are maintained. That is we have to maintain checks that all of the tuples are of the same kind and there are no duplicates that are there in the tuples. Moving on, if we consider a relation of tuples, relation of tuples is also a collection of tuples just like a bag or a set, however it imposes even more constraints on the set of tuples that is we have seen some kinds of relational constraints like the entity constraint, the key constraints and referential constraints, no duplicates and so on.

So a relation also does not allow for duplicates in tuples. It also explicitly states that all tuples have to be of the same kind that is they have adhere to the same schema that is defined by the relation and it explicitly forbids any tuple that does not adhere to the schema and there are strong constraints of about how the values of each attribute should be that is what should be the domains and how each what is the domain of each attribute and where can a value lie and so on. And there are other constraints like key constraints which can uniquely define a tuple and because there are no repetitions in a relation, by default the set of all attributes of a tuple form its super key and so on and so forth. In this session we are going to look at these constraints in much more detail. We are going to be concentrating on the concept of constraints and an associated concept of triggers in database systems. Triggers are concepts that are quite prevalent in what are called as active database systems.

(Refer Slide Time: 07:14)

And we are going to be looking at triggers in this context, in this session. So let us look at what are the different kinds of constraints that are imposed or either implicitly or explicitly in a typical database management system. Let us define the term integrity

constraints. As the term implies integrity constraints are constraints that strive to enforce the integrity of the database system. What is an integrity of a database system? The integrity of a database system or a system of data elements essentially states that the set of all data elements that are stored in this system is a valid set.

(Refer Slide Time: 08:01)

Note that a valid set or validity is different from correctness. Validity simply states that or validity is some kind of correctness that is independent of what the user or what the application really considers semantically correct from the usage perspective. For example if I state that the marks obtained by a student can range from 0 to 100 then a value of 200 for the attribute called mark is an invalid number. It is not at all, it is not valid so this pertains to validity. However if I enter a value called let us say 50 for a particular student it may be valid. However it need not be correct, it could be an incorrect entry for that field. The student may have actually obtained 90 marks whereas I would have entered 50 for the student.

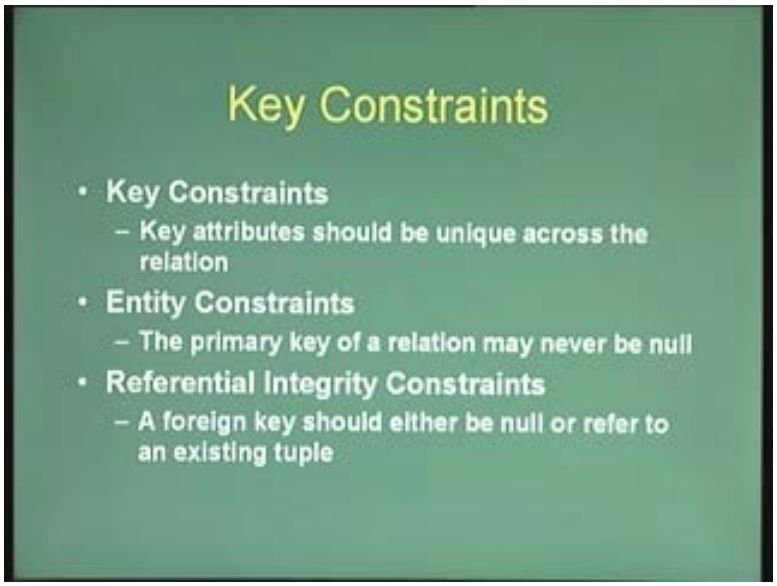
Therefore it is an incorrect value, however it is a valid value. Integrity constraints as part of the dbms are independent of the application programs. They have no idea about what is the application context or what is the usage scenario in which this particular data element is being used. Therefore they can only talk about or they can only enforce validity of the database values or of the data elements stored in the database. They cannot obviously ensure correctness of the data that is stored in the database.

(Refer Slide Time: 09:47)

One of the first form of key constraints in the relational model, one of the first form of constraints in the relational model is the key constraints. We have seen key constraints when we talked about the relational model and also in sql but let us revisit key constraints here again for the sake of completeness, when we are talking about constraints. A key constraints is a very fundamental form of relational constraint and it manages entity existence and of referential integrity.

What is meant by entity existence? Entity existence essentially means that we should be able to deference that is we should be able to identify each entity or each tuple in our database uniquely so that essentially means that each entity should have at least one super key using which it can be uniquely identified or it can be uniquely distinguished from the rest of tuples in the database. In the relational model or in the relational algebra since a relation forbids duplicate tuples, the entire tuple in the worst case itself is the super key of the tuple.

(Refer Slide Time: 11:03)



So the key constraints therefore states that key attributes should be unique across the relation that is there should be no two tuples having the same key attribute that is if I identify a subset of attributes as a key attribute, there should be no two tuples such that the key attributes are the same that is such that they are indistinguishable as far as the key attribute is concerned. The entity constraint stipulates that the key attribute can never be null because all null attributes are the same, no matter where they occur and there is no difference between a null attribute in the name field verses a null attribute in the age field. All null attributes are the same, it means there is no value associated with it.

The entity constraints therefore states that the primary key relation that is the relation, the subset of attributes that can uniquely identify tuples in a relation may never be null. And the third kind of key constraint is a referential integrity constraint. The referential integrity constraint essentially is a constraint over the foreign keys. Remember what a foreign is. A foreign key is a set of attributes or subset of attributes within a tuple that refer to another tuple in a different relation, mainly the primary key or any key attribute of another tuple in a different relation. A foreign key or the referential key integrity constraint states that a foreign key should either be null that is it should not refer to any other tuple or if it refers that is if it is not null then it should refer to an existing tuple.

We cannot refer to a tuple that does not exist; we cannot assign a manger to a department for example which is non-existent. We cannot assign a manager to a non-existent department. On the other hand we can say that a manager is not assigned to any department at all that is set foreign key as null which is quite acceptable as far as a referential integrity constraint is concerned.

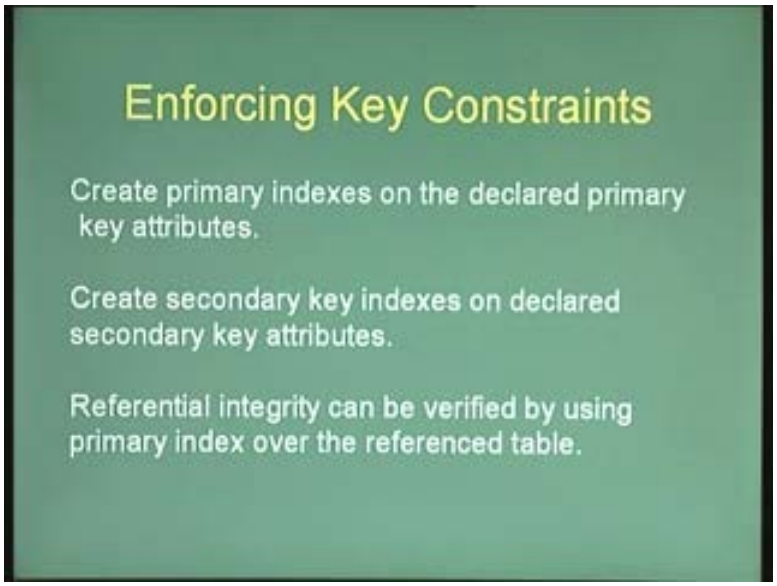
(Refer Slide Time: 13:20)

How do we specify key constraints using sql? We have seen this in the session on sql, let us briefly summarize it here again. Note the creation of a table in sql. The following slide shows a table called employee which has two key constraints, one is the employee number which is the primary key which is marked as a primary key and there is a pan number for each employee which is a secondary key. That is it is given some constraints called non-null and unique and so on. And there is also a foreign key that is the field called REPORTSTO which states who is the supervisor or who is the manager to which this employee reports to.

So the foreign key references another employee tuple and references the field called or references the attribute called employee number in another employee tuple or in another employee of another tuple in employee relation. So these are the circles that are shown in this slide depict how the key constraints are identified. The first circle shows that when we stipulate that pan number is not null then we have identified that as a key constraint that is it is one of the keys by which can uniquely identify tuples in this relation. The second circle shows that the employee number is a primary key which implicitly states that employee number may not be null. So therefore the not null constraint for the employee number is actually spurious but nevertheless it states that employee number is another key which is used as the primary key that is which is used to, which is actually used to deference every tuple in this relation.

Finally we identify foreign keys using the foreign key constructs that is we identify which, first of all which field in this tuple is the foreign key and it refers to which field in which other relation. That is it references the relation called employee and the field called or the attribute called employee number.

(Refer Slide Time: 15:53)

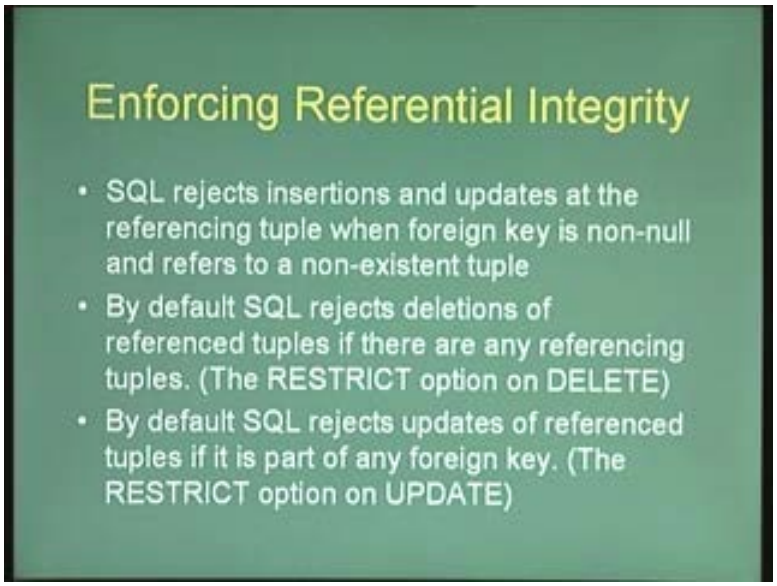


How can key constraints be enforced? Now, that we have seen some physical aspects of data storage or data management. Let us go inside the dbms to see how or what could be a possible means by which constrains or key constraints can be enforced. A simple way of enforcing constraints is to create an index structure over the field that form the keys. For example if we create any kind of index structure, let us say primary index on the declared primary key attribute then we can easily enforce the unique constraint. Note that primary key has to be unique, all keys have to be unique. Therefore whenever a new tuple is been inserted into the database system, we just use the index structure to verify whether a key attribute of this value already exist in the database system or not.

If it already exists then inserting a new tuple with the same value is going to violate the uniqueness constraint so in which case this can be rejected and the integrity constraint can be enforced. Similarly for all other attributes that is all secondary key attributes, we can maintain secondary indexes such that we can always verify whether before insertion of a tuple whether the corresponding value for that key actually exist in the database or not.

Referential integrity similarly can be verified by using a primary index over the other table that is the over the table that is being referenced. Therefore whenever I insert a tuple in a relation that contains a foreign key, I first verify whether this tuple in fact really exists in other table whether the foreign key, whether the foreign key references to a tuple that actually exists in the reference table. Only if this is so then I have to allow the insertion of the tuple in the referencing table.

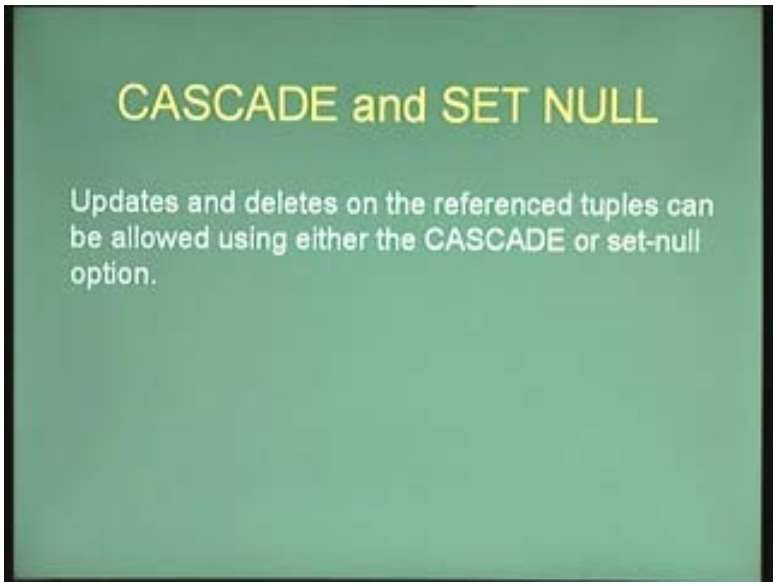
(Refer Slide Time: 18:03)



So let us probe a little further into enforcing referential integrity. Sql automatically rejects whenever a tuple is being inserted into a relation containing a foreign key such that the foreign key is not null and it refers to a non-existent tuple and this rejection is performed using corresponding index structures. And by default sql uses the restrict option for managing alterations or updates in tables. For example if a tuple is deleted or if a tuple is asked to be deleted in the referenced table that is let us take an example of a manager working in a department. If the department tuple of the corresponding department table is to be deleted then sql rejects such a deletion because there is a foreign key constraint that is from the table called manager which is referencing this tuple.

Unless of course we use the cascade option in which case even the corresponding referencing tuple would be deleted that is the corresponding manager tuple which is referencing the department tuple would also be deleted when the department is deleted. Similarly sql rejects any updates to the part to tuples that can affect the foreign key constraints. For example if I try to update the department id in the manager table such that it now points to a non-existent department, sql rejects this update. On the other hand if I try to update the department table and change the department number such that it violates some foreign key constraints that are pointing to it that is it makes some foreign keys dangling then sql would reject such an update.

(Refer Slide Time: 20:21)



Of course updates and deletes, we can force the dbms to go ahead with the updates and deletes by using the cascade option in which case the overall referential integrity is still maintained. However updates and deletes are cascaded that is every referencing tuple is also updated whenever the reference tuple is being updated.

(Refer Slide Time: 20:48)

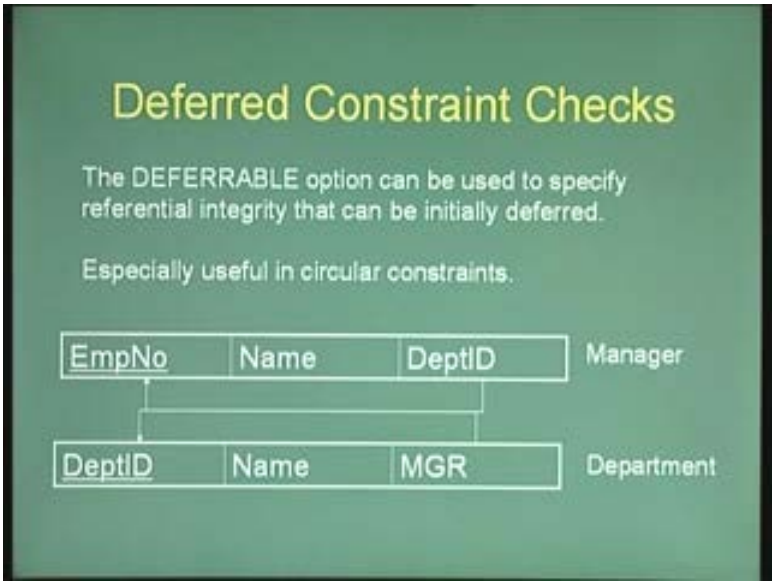
There are also other constructs in sql that instructs the dbms to perform in the different fashion, to act in a different fashion than mere cascade. A cascade simply says that whatever changes is being made in the reference tuple make the corresponding changes in the referencing tuple.

That is if I change my department id from 50 to 102 change all referencing tuples that are referencing to department number 50 to department number 102 that is simple cascade. However we may want to do some other operations other than simple cascade. There are other options that we can use to instruct the dbms to perform differently. One of such option is a set null option. Have a look at the relation shown in the slide here. The slide creates or the relation creates a table called EMPLOYEE where employee number is the primary key and there is a field called name and there is a foreign key called REPORTSTO which is the employee number of the manager or the other employee to which this particular employee reports to.

Therefore, foreign key references employee relation and the employee number field and then in the foreign key relation or in the foreign key specification there are two other constructs. The first construct says that on delete set NULL and then the second relation says or the second construct says that on update CASCADE. What do these things mean? The first thing, the first construct says that on delete set NULL. Essentially it means that if the tuple that I am referencing to is deleted then set NULL. That is set this field to be NULL. Note that a foreign key can be NULL without violating referential integrity. Therefore this semantics of this statement means that if the person whom I am reporting to is for some reason deleted from the set of employees. Then set the field as NULL that is I am not reporting to anyone.

On the other hand if the person to whom I am reporting to changes his employee number let us say I am reporting to a manager with employee number 102 and employee number is changed to 150 then it says on update cascade. That is cascade this new employee number to and make corresponding modifications in the foreign key construct so that referential integrity is maintained.

(Refer Slide Time: 23:43)



Sql also supports what are called as deferred constraint checks. A deferred constraint check essentially says or tells the dbms that when inserting a tuple that don't make a constraint check right now. Why do we require a deferred constraint checks? Deferred constraints checks are especially useful when we have circular constraints or circular referential integrity. Have a look at the figure shown in the slide here. The figure shows two tables with corresponding referential integrity. The table essentially states that the first table is a manager table which contains a primary key called employee number and the name of the manager and the department id that is managed by the manager.

The department field is a foreign key into the department relation. The department relation in turn is having a primary key called department id, it has a department name and a field called manager which in turn is a foreign key onto the employee number. Consider that this company which is having a schema like this has recently upgraded its database system and now they have installed a new dbms and they are now porting the set of all data that they have from their old database system to the new database system. Now when they are porting which essentially means that they are adding these tuples in a batch mode, adding a set of tuples in a batch mode. There is no guarantee that or it is very difficult to guarantee that the tuples would be added in the order in which they are required. That is whenever I am adding a department id, the employee the corresponding employee id for the manager already exist and whenever I am adding a manager tuple, the corresponding department id for which he is a manager already exist. This is very difficult if not impossible to sustain or maintain. So in such situations it is useful to have a deferred integrity check.

So if we use the deferred or deferrable option in sql then the dbms defers integrity checks till the present transaction is completed. We have not looked into the concept of transaction has yet. However let me give a brief intuition about the notion of transaction. A transaction simply is a logical unit of database operations. For example if I talk about debiting and crediting between two accounts, let us say I am performing wired transfer between my account and my friend's account. So a transaction in this case is the set of operations that debits a set of, an amount of money from my account and credits it to my friend's account. This entire set of operations belongs to one semantic entity or semantic transaction so to say and either both of them have to be performed or none of them should be performed. That is either both debit and credit should happen or none of them should happen.

It shouldn't be the case that my account is debited but my friend's account is not credited or the case that my friend's account is credited with some money from somewhere but my account is not debited. Either both of them should happen or none of them should happen. The deferrable option essentially defers integrity constraints that is until the end of the transaction. In the middle of the transaction we might encounter a situation where the integrity is violated. However that is as far as we have at the end of the transaction, we have perfect integrity that is we have maintained the integrity of the overall database system. This slide shows how or what is the syntax of the deferrable construct. Have a look at the slide here. This slide creates a table called manager which is shown in the previous slide or which was shown in the previous slide and manager has an employee

number attribute which is a primary key and name attribute and a department id which is the foreign key.

(Refer Slide Time: 28:32)

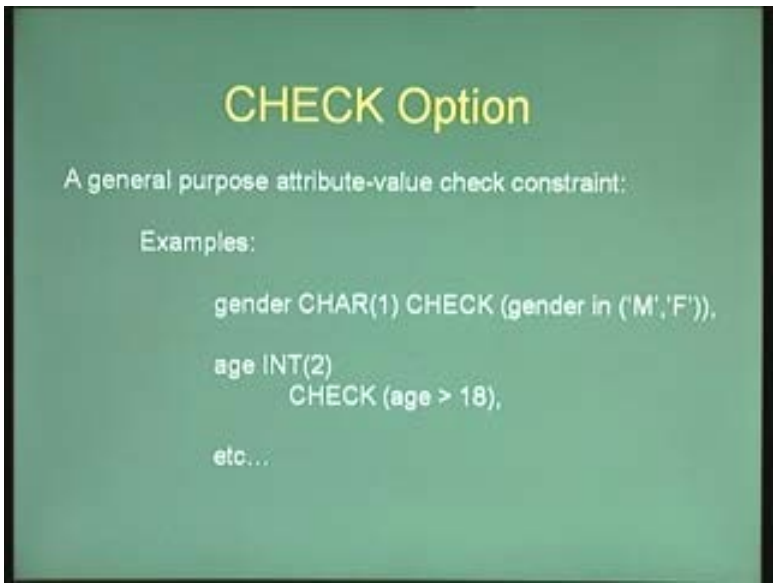
Of course there is no need to separately specify foreign key if we directly say that department id references some other tuple that is department department id and we say that it is deferrable and also that initially deferred. That is it is initially deferred that means to say that deferrable essentially means that it is possible or it is ok if integrity check on this referential integrity is deferred till the end of the transaction.

And this second one says that initially deferred means explicitly tells the dbms engine that as soon as a tuple is tuple of type manager is inserted, initially defer the referential integrity checks. Let us to move onto other kinds of constraints from key constraints. The next kind of constraints that we are going to look at are constraints on attributes. Of course we have already seen a few constraints on attributes as some examples are shown in the slide here.

(Refer Slide Time: 29:42)

The first constraint that we have seen was the NOT NULL constraint which essentially states that null value for this particular attribute is not a valid value. And the second kind of constraint which is also key constraint but some sense also an attribute constraint is the unique constraint. It essentially checks against duplicate values of the attribute being inserted into the relation.

(Refer Slide Time: 30:13)



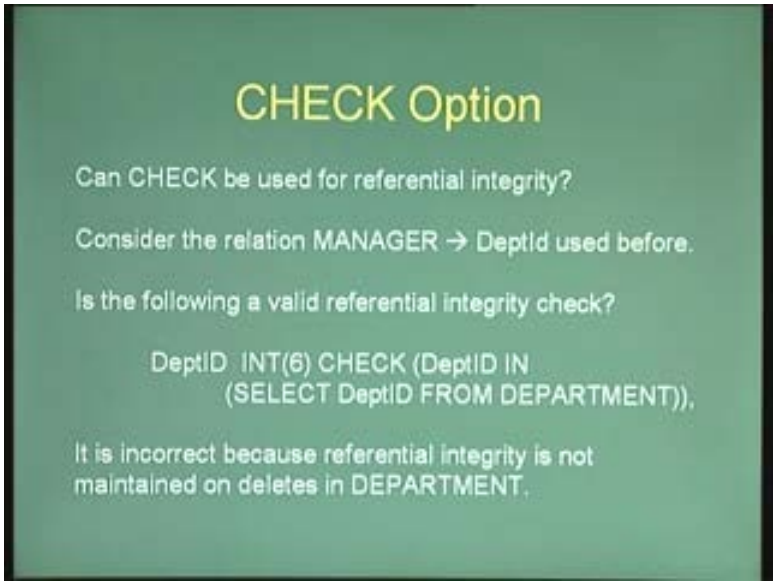
A more general form of attribute constraints in sql is what is called as the check option. We can specify any general kind of attribute constraints using the check option on a relation. There is an example or there are two examples shown in this slide here. The first

example defines a field or an attribute called gender and says that it comprises of a single character char 1 and then imposes an integrity check on the value of this field or the value of this attribute. It says that check gender in M or F that is those are the only two valid characters that can be assigned for gender. Even though it is a single character, it is not just a single character but it is a single character in this set of two characters M and F.

Similarly the second example shows a field called age that contains an integer of two digits which can contain an integer having two digits and there is a check that states that check that age is greater than 18 especially if you are talking about employee records and so on. There might be a legal integrity constraint that is a legal minimum age for an employee of 18 years. Therefore that can be directly included as part of the check constraint that is check age greater than 18. Note that the first constraint that is check gender in M or F is a physical constraint that is it is a constraint that is given as part of the physical world around us.

And the second kind of check that is used that is check age greater than 18 is the normative constraint. It is a constraint that is imposed by the local laws or the set of norms which define the set of correct behavior within this system. And it can change from system to system in some places probably the minimum working age is let us say 16 or in some other places it could be 21 and so on. So that is a normative constraint while the formal constraint is a physical constraint.

(Refer Slide Time: 32:38)

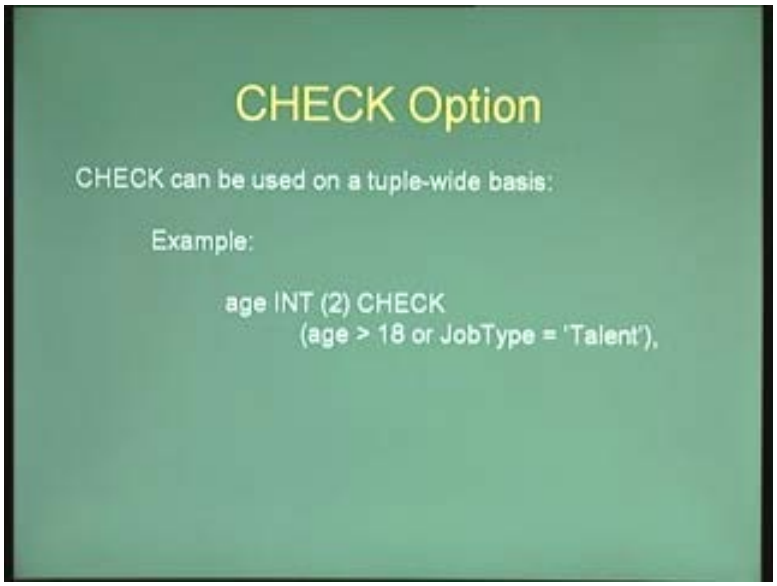


When you ask a kind sort of interesting question, can the attribute constraint that is can the check constraint on values of attributes be used to enforce referential integrity. Referential integrity as you know is a key constraint that is it is a integrity constraint across different relations. However have a look at the small declaration here which might seem to say that we can actually use a check as check for enforcing referential integrity.

The example that shows the declaration of a field called department id and department id shown as integer having 6 digits and then there is a check here. The check says that check department id in select department id from department. Now this is the department id of the manager tuple and when a manager tuple is being inserted there is a check that is made to see that the department id that is being inserted for the manager actually exists in the department relation which looks like or which seems to suggest that we can enforce a referential integrity using, referential integrity using check statement. That is if it try to insert a tuple or if we try to insert or even modify a tuple of the manager relation such that it tries to reference to a non-existent department id, this check fails and the updates or insertion is rejected which seems to suggest that referential integrity can be enforced using check.

However look at the other way around. What happens if a department existed when the manager tuple was inserted and then at a later point in time, the corresponding department tuple was deleted. There is no way that this check constraint is now enforced because the check doesn't even know that the corresponding tuple from that is being referenced in the department table is being deleted. Therefore because of this, because such situations cannot be handled by check, we say that we cannot use the check condition for enforcing referential integrity. The check constraint can not only be used on just an attribute basis, it can be used on a tuple wide basis.

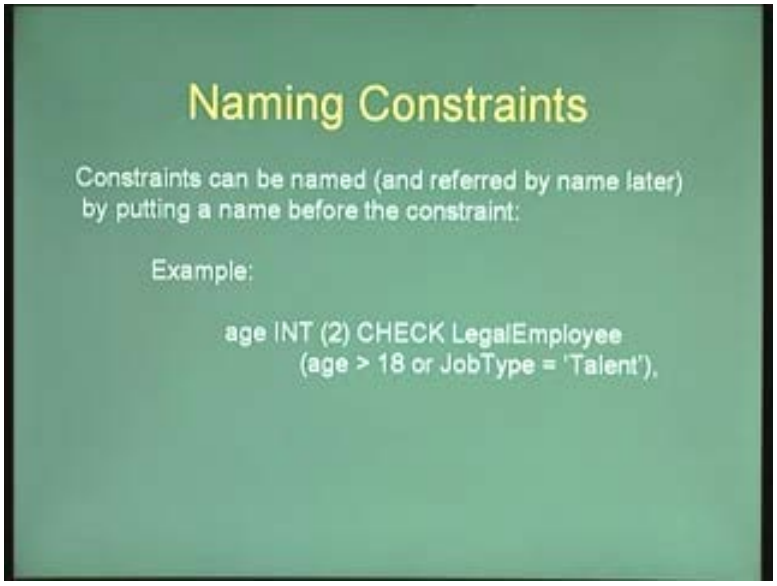
(Refer Slide Time: 35:28)



That means a check constraint can actually perform checks on several attributes on a given tuple. The following example shows such a situation where it says age INT of 2 that is the age of an employee let us say is declared as an integer having two digits and a check is performed to see whether age is greater than 18 that is the legal age for a particular employee in a given company setting should be greater than 18 or there could be children employed by the company as long as the job type is based on talent. That is job type is based on encouraging the child's talent.

Therefore you can either perform a check on the age field which says age should be greater than 18 or if age is not greater than 18, you can check a separate field or a separate attribute called job type to see whether it is set as talent in this particular tuple.

(Refer Slide Time: 36:39)



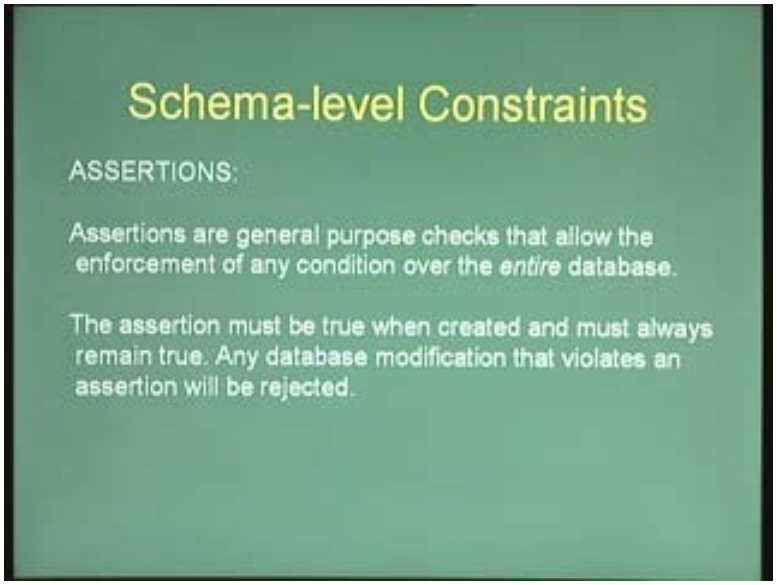
It is possible to give names to constraints to declare in sql so that they can be referenced by their names rather than by their actual conditions. This particular slide shows such an example where the check called the legal age of an employee or the minimum age of an employee, that constraint on this is given a name called legal employee. That is age INT of 2 and name is given called legal employee which is the constraint called check age greater than 18 or job type equal to talent.
(Refer Slide Time: 37:20)

Once we have named tuples, we can alter constraints that is once we have named constraints we can alter constraints that is we can add or delete constraints by referring to them by their names. For example the following slide shows two different alter commands. The first command says that alter table employee drop constraint legal employee. If the particular constraint is no longer valid, we can refer to a constraint by name and then say drop it. So that constraint is no longer enforced in future additions of or future updates to the table.

Similarly this second alter statement shows how a new named constraint can be added to the table at any later point in time. That is alter table manager and add constraints all works that is the constraint name is called allworks and check the department id is not null. That is check to see that for all the employee manager tuples in this manager relation, the department id that is the corresponding department which they have to manage is not null. That is there is no such manager who is not managing a department.

Now what happens if there are already some fields or already some tuples in the relation whose department id's are null. In such a case the addition of this constraint fails that is the constraint cannot be added because it cannot be enforced on this table because there are already some managers whose department id's are null.

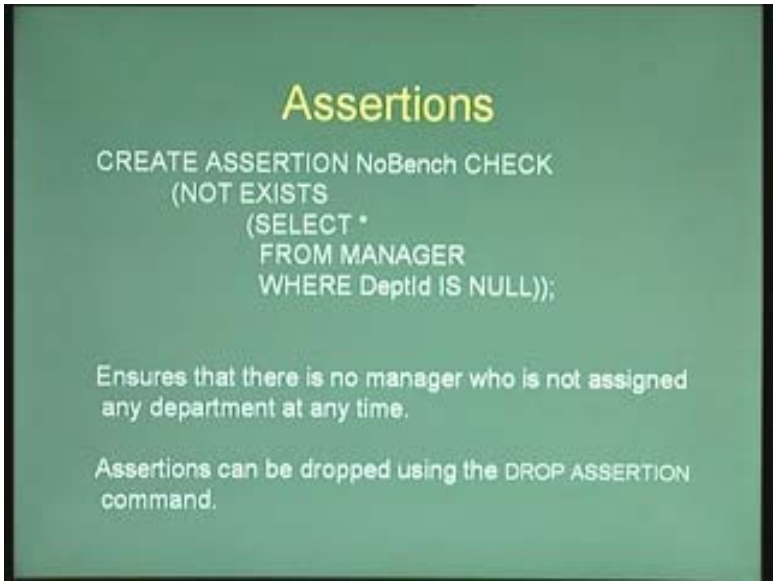
(Refer Slide Time: 39:11)



A more powerful kind of constraint on in sql is what are called as schema level constraints. Until now we have been looking at constraints that acted at the tuple level or at the key level; tuple or attribute or keys or so on. That is which acted on specific instances of a relation. A schema level constraint on the other hand acts at the level of a schema that is which is enforced for all tuples on a database wide basis. One such powerful constraint, a powerful general purpose constraint is what is called as assertions.

Assertions are general purpose checks that can be performed on the entire database and can be enforced on the entire database for the entire time in which the database is in operation. That is if I make up a specific assertion, the assertion must be true when it is made otherwise the assertion fails of course and the assertion must be true when it is made. And it must be true throughout the life time of the database or until the assertion is dropped. That is once an assertion is made, it will be enforced that is every modifications to the database in whatever table, in whatever tuple would be checked to see that the particular or this specific assertion is not being violated.

(Refer Slide Time: 40:50)



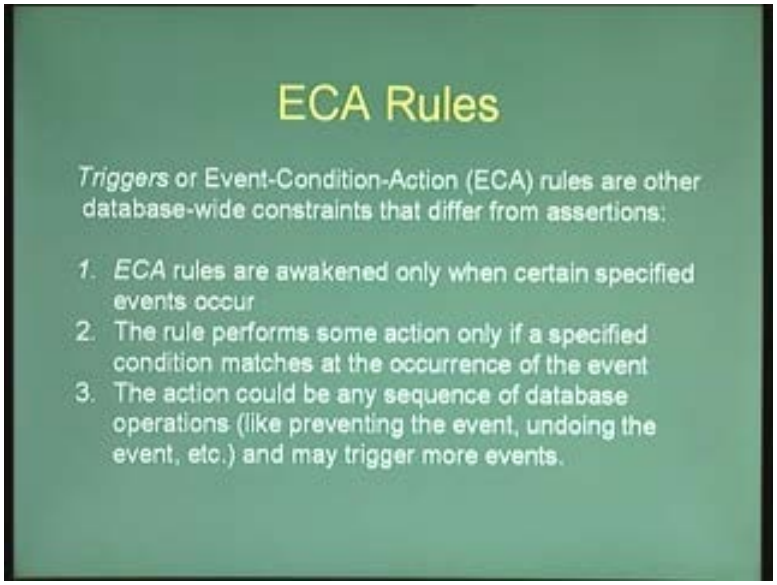
This slide shows an example of an assertion. An assertion can be created using the create assertion command. This slide shows create assertion command that creates an assertion called NoBench. NoBench is a name of an assertion and the assertion essentially checks that is an assertion is a check statement which checks to see that they does not exist any manager whose department id is null. That is there is no manager who is on the bench so to say that means who is not assigned any department to manage and this assertion should be true when this is first asserted. That is when I execute the create assertion statement, all managers in the database that are listed in the database should be associated with a department. And if there are any managers who are not associated with a department then the create assertion fails.

The assertion can be created only when either the managers who are not assigned any departments are modified to assign departments or those tuples are deleted from the database. And once an assertion is created that is once such an assertion is created it is enforced or it is maintained whenever there is a database update that is happened. And any update or any addition of a manager tuple whose department id is null is rejected. And this set of assertions holds for the life time of the database or until the assertion is dropped or until the assertion is deleted. Assertions can be dropped using the drop assertion command.

So we just say drop assertion NoBench and then the assertion no longer holds and further on tuples violating this assertion can be inserted into the database. The second kind of schema level constraints that we are going to look at are what are called as ECA rules or triggers. An ECA rule expands to an even condition action rule, this is an integral part of an specific kinds of databases is called active databases. However ECA rules have been incorporated into most commercial databases as of now, that is let us say db, ibm db 2 oracle 9i or 10 g which is the latest all of them incorporate some form of ECA rules.

An ECA rule or a trigger is again a database wide constraint but which differs from an assertion in the sense that there are not always active that is they are not always enforced and ECA rule is enforced or an ECA rule is awakened only when certain specified events occur.

(Refer Slide Time: 44:08)

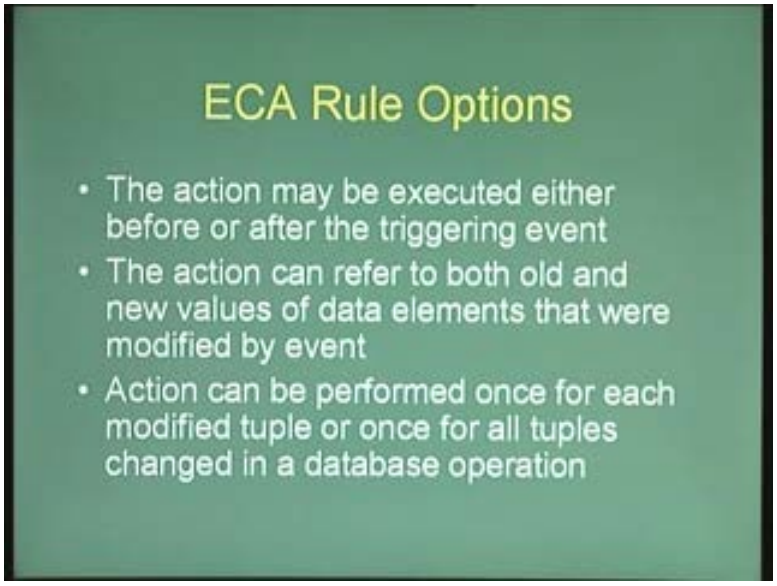


That is note that ECA stands for event condition and action. So when an event occurs a corresponding ECA rule is awakened. The rule then performs a or the rule then performs a condition check to check whether the condition holds. That is when the event condition, when the event occurs if the condition holds then the rule performs a given set of actions that is the A part in the ECA rule. And the action could be anything, it could be like preventing the event from proceeding or undoing the event or any other set of database update operations that could be totally unrelated to the event.

It could actually be something like intimating the user about the event and so on. And the action in turn may generate more events which in turn could trigger more ECA rules and so on. There are several options that sql provides to handle ECA rules. We can either specify that the action part of a ECA rule be executed either before the event occurs or after the event occurs. That is before a particular event is going to happen, make a check for any ECA rule that is waiting on this event or one could check for performing the

action after the event occurs. The action part of an ECA rule can refer to both old and new values of data elements that are modified by the event.

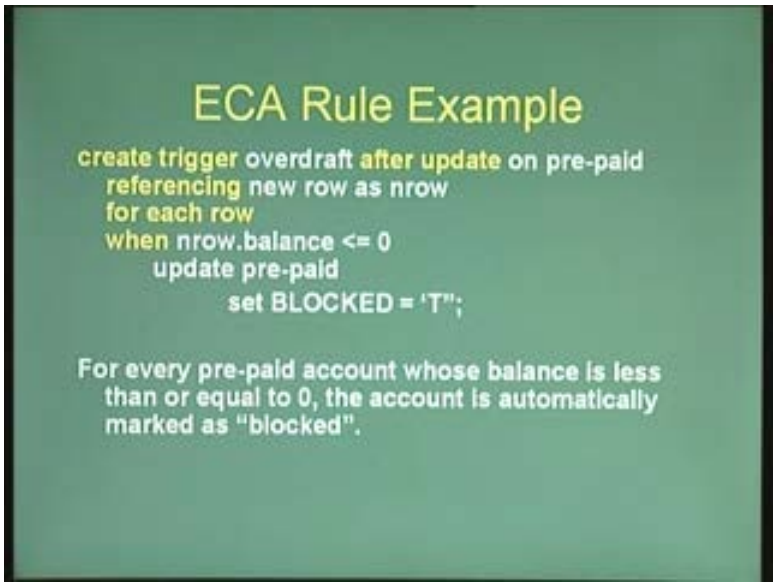
(Refer Slide Time: 45:34)



That is sql or several or most commercial dbms systems would maintain the older values of data elements that were modified by certain update events in case they trigger certain ECA rules and the ECA rule would want to refer to the older value of the data element. For example one might specify an ECA rule that says alert the user whenever the current stock price let us say falls below or falls by greater than 5%.

Therefore we need to know the older and newer value of this particular stock price in order to see whether the change in one update is greater than 5%. If that is so then the user is alerted and the action part of an ECA rule can be either specified to be performed once for each modified tuple or once for all tuples that are modified during the update event that is during the database operation.

(Refer Slide Time: 46:49)



This slide shows a particular example of an ECA rule. It says all the keywords are shown here in a highlighted form, the keyword to specify an ECA rule is the term called create trigger. So this statement creates a trigger called overdraft which has to be performed after update that is the action part of this trigger has to be performed after the update. Note that actions could be performed either before or after the updates, so this has to be performed after update on a relation called pre-paid and referencing each new row that is the new row that were modified has nrow and for each row that was modified, it checks whenever that is when balance that is nrow dot balance attribute is less than or equal to 0 then update that row and set block or set the attribute block equal to true.

(Refer Slide Time: 48:11)

That is for every pre-paid account whose balance is less than or equal to 0, the account is automatically marked as blocked as part of this trigger operation. What are some of the properties of ECA rules? ECA rules are automatically triggered by the DBMS. The application program need not even be aware of the corresponding ECA rules that were triggered. It is the rule writers responsibility that is whoever enforces these rules or whoever wrote these rules to enforce these integrity constraint, it is the rule writers responsibility to ensure that the rules terminate. And it is also the rule writers responsibility to prevent cascading triggers that is this action would in turn create another cascading action which in turn would trigger some more rules which in turn would create another cascading action triggering more rules and so on.

So it is a rule writers responsibility to prevent such conditions from happening. A rule write can also write a cyclic rule. That is rule A triggers rule B and the action of rule B triggers rule A that is rule A tries to undo something, undo an update and which in turn triggers rule B which tries to redo it back and it can obviously go into infinite rules. In most dbms systems termination and infinite cycles are simply handled by seeing or just maintaining a count of the nesting level that is how many iterations have been made and if this crosses a maximum threshold then the entire set of operations including the update that triggered this infinite rules are rolled back. That is they are all undone and nothing is changed and the database state is taken back to the state before the event that created this infinite looping of rules.

(Refer Slide Time: 50:09)

The last kind of integrity checks that we are going to be looking at in an sql based database system are the notion of authorization and privileges. Authorization and privileges talk about which user is authorized to do what or which user enjoys what kind of privileges. Users in an sql based system can enjoy certain privileges that grants them certain authorization. What do we mean by authorizations? And this could mean an authorization to read a particular tuple for example is a particular or is a given user

authorized to let us say look at the account details of some other user or is a given employee is authorized to look at salary details of some other employee and so on.

So there could be a read authorization, there could be inset authorization that is a user allowed to insert tuples into the database. There is an update authorization, delete authorization, index authorization that is index essentially means that can a user be allowed to create and delete indexes on a particular table. Why is this security constraint, why is index important, that is why should I use authorize to create and delete indexes this is because what happens if I use a weather intentionally or unintentionally deletes the primary key or the primary index structure on a tuple. Then the entire database or the entire data file becomes extremely inefficient to access. Therefore a user used should be authorized to create or delete indexes then alternation and drop authorizations.

(Refer Slide Time: 52:01)

What about views, what kinds of authorizations can we impose on views? Most database systems allow authorizations to be specified on views without obtaining the specific authorizations on the base table. That is let us say a hr manager would have a read authorization on a hr based view of the employee records. That is which contains employee salary details, perks and so on. However there may not be or the manager may not enjoy read authorization on the actual employee tuple which may contain some more information like his or her personal details and so on which the manager may not be authorized to access. Therefore even if the corresponding authorization does not exist on the base table, it is possible to obtain certain authorizations on the view. And since views are derived table, whenever a query is given through a view authorization checks should be performed before the query is answered.

(Refer Side Time: 53:09)

The general form of providing authorizations is to use a grant command. The grant command simply says that grant privilege list on a particular relation or view name to a set of users.

(Refer Slide Time: 53:26)

This slide shows such an example which says GRANT insert update and select privileges on the manager table to public. Public means that all users that is everybody, every user in the database system.

(Refer Slide Time: 53:42)

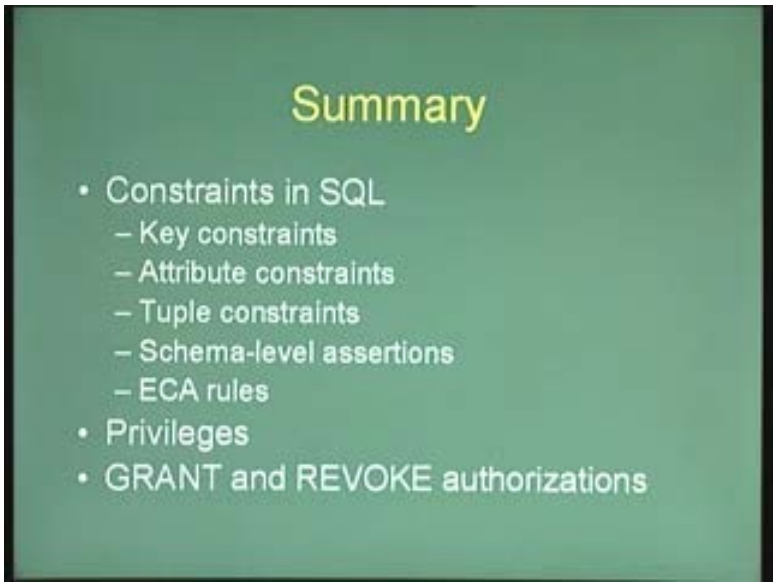
The privilege names used in the grant command or the same names as the sql command that is used for this privilege that is a select privilege gives an authorization for reads, an insert privilege gives an authorization for inserts and so on. An update privilege gives an authorization for modification and so on and all privileges, the keyword called all privileges suppose we say grant all privileges, it means all the above privileges can be granted to the specific user in question.

(Refer Slide Time: 54:16)

It is possible to revoke privileges that is by using the revoke command that is revoke privilege list on relation or view name from user list and of course we can use now

familiar terms called either restrict or cascade. Restrict basically means that if I revoke a particular privilege from a set of, from a given user all the privileges that were granted by the user to other users will not be revoked, they stay in place. On the other hand a cascade privilege says that if I revoke privilege x from a user and if that privilege X has been passed on or granted by the user to other users, they are also invoked in a cascading fashion.

(Refer Slide Time: 55:04)



So that brings us to the end of this session where we have looked into the notion of constraints in the database system. Essentially we have seen in different kinds of constraints that a typical dbms system provides and they can be classified as either key constraints that act on the key attributes of a particular relation or they could be attribute constraints which can act on any attribute which checks domains or validity of the attribute value with reference to its domain in any given relation and there are tuple constraints which act on the complete tuple itself that is where the value of certain attribute may depend on the value of other attributes on a tuple wide basis and so on.

And then there are schema level constraints especially assertions which are a general purpose schema level constraint which maintain certain integrity over the entire database for the lifetime of the database or until the assertions are dropped. And finally we looked at triggers or ECA rules that are not valid always but are awakened whenever certain events happen and a certain condition holds which in turn prompts them to perform certain actions. We finally then looked into the notion of privileges and authorization that different users in a database system can hold and they could be granted certain privileges and privileges could be revoked from them and this can happen in a restricted fashion or in a cascading fashion. So that brings us to the end of this session.