Integrity and Security

Dr. M. Brindha
Assistant Professor
Department of CSE
NIT, Trichy-15

Domain 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.
- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domains can be created from existing data types
  - E.g. create domain *Dollars* numeric(12, 2)
    create domain *Pounds* numeric(12,2)
- We cannot assign or compare a value of type Dollars to a value of type Pounds.
  - However, we can convert type as below (cast *r.A* as *Pounds*) (Should also multiply by the dollar-to-pound conversion-rate)

Domain Constraints (Cont.)

- The check clause in SQL-92 permits domains to be restricted:
  - Use check clause to ensure that an hourly-wage domain allows only values greater than a specified value.

    create domain *hourly-wage* numeric(5,2)
    constraint *value-test* check(*value* > = 4.00)

  - The domain has a constraint that ensures that the hourly-wage is greater than 4.00
  - The clause constraint *value-test* is optional; useful to indicate which constraint an update violated.
- Can have complex conditions in domain check
  - create domain *AccountType* char(10)
    constraint *account-type-test*
    check (value in ('Checking', 'Saving'))
  - check (*branch-name* in (select *branch-name* from *branch*))

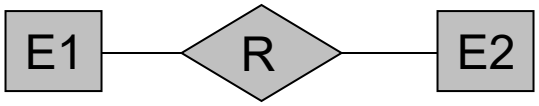
Referential Integrity

- 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 "Perryridge" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch "Perryridge".
- Formal Definition
  - Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively.
  - The subset α of R_2 is a *foreign key* referencing K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$.
  - Referential integrity constraint also called subset dependency since its can be written as

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K1}(r_1)$$

Referential Integrity in the E-R Model

- Consider relationship set R between entity sets E_1 and E_2. The relational schema for R includes the primary keys K_1 of E_1 and K_2 of E_2.
- Then K_1 and K_2 form foreign keys on the relational schemas for E_1 and E_2 respectively.



- Weak entity sets are also a source of referential integrity constraints.
  - For the relation schema for a weak entity set must include the primary key attributes of the entity set on which it depends

Checking Referential Integrity on Database Modification

- The following tests must be made in order to preserve the following referential integrity constraint:

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K}(r_1)$$

- **Insert.** If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K] = t_2[\alpha]$. That is

$$t_2[\alpha] \in \Pi_{K}(r_1)$$

- **Delete.** If a tuple, t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference t_1:

$$\sigma_{\alpha = t1[K]}(r_2)$$

  If this set is not empty
  - either the delete command is rejected as an error, or
  - the tuples that reference t_1 must themselves be deleted (cascading deletions are possible).

Database Modification (Cont.)

- **Update.** There are two cases:
- If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α, then a test similar to the insert case is made:

Let t_2' denote the new value of tuple t_2. The system must ensure that

$$t_2'[\alpha] \in \prod_K (r_1)$$

- If a tuple t_1 is updated in r_1, and the update modifies values for the primary key (K), then a test similar to the delete case is made:

The system must compute $\sigma_{\alpha = t1[K]} (r_2)$

using the old value of t_1 (the value before the update is applied). If this set is not empty

1. the update may be rejected as an error, or
2. the update may be cascaded to the tuples in the set, or the tuples in the set may be deleted.

Assertions

- An *assertion* is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL takes the form

  create assertion <assertion-name> check <predicate>
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
  - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.

Assertion Example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
 (not exists (select * from branch
                where (select sum(amount) from loan
                        where loan.branch-name =
                              branch.branch-name)
                      >= (select sum(amount) from account
                          where loan.branch-name =
                                branch.branch-name)))
```

Assertion Example

- Every loan has at least one borrower who maintains an account with a minimum balance or $1000.00

```
create assertion balance-constraint check
 (not exists (
     select * from loan
       where not exists (
        select *
          from borrower, depositor, account
          where loan.loan-number = borrower.loan-number
              and borrower.customer-name = depositor.customer-name
              and depositor.account-number = account.account-number
              and account.balance >= 1000)))
```

Triggers

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.

Trigger Example

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - setting the account balance to zero
  - creating a loan in the amount of the overdraft
  - giving this loan a loan number identical to the account number of the overdrawn account
- The condition for executing the trigger is an update to the *account* relation that results in a negative *balance* value.

Trigger Example

```
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
        insert into borrower
           (select customer-name, account-number
            from depositor
            where nrow.account-number =
                  depositor.account-number);
     insert into loan values
           (n.row.account-number, nrow.branch-name,
                                        – nrow.balance);
     update account set balance = 0
        where account.account-number = nrow.account-number
end
```

Triggering Events and Actions in SQL

- Triggering event can be insert, delete or update
- Triggers on update can be restricted to specific attributes
  - E.g. create trigger *overdraft-trigger* after update of *balance* on *account*
- Values of attributes before and after an update can be referenced
  - referencing old row as : for deletes and updates
  - referencing new row as : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blanks to null.

```
create trigger setnull-trigger before update on r
referencing new row as nrow
for each row
   when nrow.phone-number = ' '
   set nrow.phone-number = null
```

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use for each statement instead of for each row
  - Use referencing old table or referencing new table to refer to temporary tables (called *transition tables*) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

External World Actions

- We sometimes require external world actions to be triggered on a database update
  - E.g. re-ordering an item whose quantity in a warehouse has become small, or turning on an alarm light,
- Triggers cannot be used to directly implement external-world actions, BUT
  - Triggers can be used to record actions-to-be-taken in a separate table
  - Have an external process that repeatedly scans the table, carries out external-world actions and deletes action from table
- E.g. Suppose a warehouse has the following tables
  - *inventory(item, level):* How much of each item is in the warehouse
  - *minlevel(item, level) :* What is the minimum desired level of each item
  - *reorder(item, amount):* What quantity should we re-order at a time
  - *orders(item, amount) :* Orders to be placed (read by external process)

External World Actions (Cont.)

```
create trigger reorder-trigger after update of amount on inventory
referencing old row as orow, new row as nrow
for each row
      when nrow.level < = (select level
                           from minlevel
                           where minlevel.item = orow.item)
           and orow.level > (select level
                             from minlevel
                             where minlevel.item = orow.item)
  begin
       insert into orders
            (select item, amount
             from reorder
             where reorder.item = orow.item)
  end
```

Triggers in MS-SQL Server Syntax

```
 create trigger overdraft-trigger on account
for update
as
if  inserted.balance < 0
begin
   insert into borrower
     (select customer-name,account-number
      from depositor, inserted
      where inserted.account-number =
                    depositor.account-number)
   insert into loan values
     (inserted.account-number, inserted.branch-name,
                    – inserted.balance)
   update account set balance = 0
    from account, inserted
    where account.account-number = inserted.account-number
end
```

Thank You!!!