

SQL: Advanced topics

Prof. Weining Zhang
Cs.utsa.edu

Assertions

- ◆ Constraints defined over multiple tables.
- ◆ No student is allowed to take more than six courses.

```
SQL> create assertion Course_Constraint
      check (not exists
            (select * from Students s
             where 6 < (select count(*)
                       from Enrollment
                       where SID = s.SID)));
```

Recursion

Examples of Recursive Queries.

- ◆ Ancestors
 - ▲ Relation: ParentOf(Parent, Child)
 - ▲ Query: Find all of Mary's ancestors
- ◆ Company hierarchy
 - ▲ Relations: Employee(ID, Salary)
 Manager(MgrID, EmpID)
 Project(Name, MgrID)
 - ▲ Query: What's the total salary cost of project "X"

With Statement

- ◆ with R1 as (query),
 ...,
 Rn as (query)
 <query involving R1, ..., Rn & other relations>
- ◆ Conceptual Evaluation
 - ▲ Compute R1, ..., Rn into temporary relations
 - ▲ Evaluate the query
 - ▲ Destroy R1, ..., Rn
- ◆ Can also specify schema for Ri's:
 with R1(A1, A2, ..., Am) as (query), ...

Example of With

- ◆ Relation: Apply(ID, Name, Location, Date)
with DL as(select ID, Date from Apply
where Location = 'Dallas'),
HO as (select ID, Date from Apply
where Location = 'Houston')
select ID, DL.Date DLdate, HO.Date HOdate
from DL, HO where DL.ID = HO.ID
- ☛ Just like "temporary view definitions".
- ☛ The Ri's can be recursive or mutually recursive
 - Must use keyword *recursive*
 - Usually need to *union* base case & recursion

Lecture 12

SQL: Interactive Queries (2)

5

Recursion in SQL

- ◆ Find Mary's ancestors from ParentOf relation.
with recursive Ancestor(Anc, Desc) as
((select Parent Anc, Child Desc from ParentOf)
union
(select A.Anc Anc, P.Child Desc
from Ancestor A, ParentOf P
where A.Desc = P.Parent))
select Anc from Ancestor where Desc = 'Mary'
- ☛ Ancestor = ParentOf
Repeat Ancestor = Ancestor joins ParentOf
Until no more changes to Ancestor

Lecture 12

SQL: Interactive Queries (2)

6

Restrictions & Features

- ◆ Only support "linear recursion": each from-clause can have at most one recursively defined relation.
- ◆ With relations can be defined as views. Not evaluated until being queried (Why useful?).
- ◆ Can define "mutual recursion": two recursive relations mutually define each other.

Commit and Rollback

- ◆ Changes to data in a user session may not be visible to other users immediately (why?)
- ◆ Use *commit* to make changes made by insert, delete and update permanent and visible to other users.
- ◆ Use *rollback* to undo uncommitted changes made by insert, delete and update.
- ◆ Normal exit performs a commit.
- ◆ Abnormal exit performs a rollback.
- ◆ Used for transaction processing (more later).

Grant Statement

- ◆ The owner of a table can grant privileges of access to the table to other users.
- ◆ Syntax: `grant {all | privilege {, privilege} on table_name | view_name to {public | user_name {, user_name} } [with grant option]`
- ◆ Privileges: `select | delete | insert | update [column_name {, column_name ...}] | references [column_name {, column_name ...}]`

Sample Grant Statements

- ◆ Grant select and insert access to Students table to users john and terry.
`grant select, insert on Students to john, terry`
- ◆ Grant all privileges to user john.
`grant all on Students to john`
- ◆ Allow all user to update Age and GPA.
`grant update (Age, GPA) on Students to public`
- ◆ Allow user john to create a foreign key to referencing SID.
`grant references (SID) on Students to john`

Features of Grant

- ◆ The owner of a table has all privileges.
- ◆ Public includes current and future users.
- ◆ If columns are not named, all current and future columns are implied.
- ◆ To grant privileges on a view, one must be the owner of the view and have the privileges on all base tables used to define the view.
- ◆ With grant option allows the grantee to grant the privileges transitively.

Roles

- ◆ A *role* is a named group of privileges that can be granted to users.
- ◆ Used to ease the task of granting privileges.

```
create role TA;  
grant create table, create view to TA;  
grant TA to Wang, Johnson;
```

Create External Schema *

- ◆ Create a view and grant privileges to a group of intended users.
- ◆ Allow John the select and insert access only to SID, Name and Age of students with GPA higher than 3.8.

```
create view Stud1
as select SID, Name, Age
from Students where GPA > 3.8
grant select, insert on Stud1 to john
```

Revoke Statement

- ◆ Revoke granted privileges on DB objects.
- ◆ Syntax:
revoke {all | privilege {, privilege ...} }
on table_name | view_name
from {public | user_name {, user_name ...}}
- ◆ Owner's privileges can not be revoked.
- ◆ Revoke all privileges of John on Students.
revoke all on Students from john

Sequence in Oracle SQL

- ◆ An Oracle object for generating a sequence of integer values. Often used to generate unique key values.
- ◆ Syntax of create sequence:
create sequence sequence_name
[increment by integer]
[start with integer]
[maxvalue integer | nomaxvalue]
[minvalue integer | nominvalue]
[cycle | nocycle]

Sample Sequences

```
create sequence emp_seq start with 1000;  
create sequence even_seq  
  increment by 2 start with 2 maxvalue 2000;  
create sequence negative_seq  
  increment by -1 start with -1;
```

- Default increment value is 1.
- Default start value for positive (negative) increment value is minvalue = 1 (maxvalue).
- With cycle specified, the integers between minvalue and maxvalue will be recycled

Use Sequences

- ◆ Two functions for sequence seq:
 - ▲ seq.currval returns the current value of seq.
 - ▲ seq.nextval returns the next value of seq.

```
insert into Employees(Emp_no, Name, Age)
values (emp_seq.nextval, 'John', 22);
```

- ◆ nextval must be used at least once before using currval.

Oracle SQL*Loader

- ◆ sqlldr (SQL*Loader) is a Unix command (in CSLab) to load data into an Oracle table from a Unix text file.
- ◆ Requires two files:
 - ▲ control file (with extension .ctl)
 - ▲ data file (with extension .dat)
- ◆ Usage: sqlldr userid/passwd control=foo.ctl
 - ▲ Other options: direct (direct load), skip (skip n lines), load (load m lines)
 - ▲ Will generate .log .bad files.

A Sample Control File

The control file is pub.ctl for Publishers table.

```
load data
infile 'pub.dat' into table publishers
fields terminated by ","
(pub_id, pub_name, city, state)
```

- It expects an empty table.
- Other options: append into, replace into.

A Sample Data File

The data file is pub.dat.

```
0736,New Age Books,Boston,MA
0877,Binnet & Hardley,Washington,DC
1111,stone Age BooAo,Boston,MA
1389,Algodata Infosystems,Berkeley,CA
2222,Harley % adkfj,Wash,DC
3333,adfadh adfhj,Berkey,CA
```

- Other formats of control and data files are also supported.