

CS3743 Introduction to Database Systems

SQL (3)

Weining Zhang

Department of Computer Science
University of Texas at San Antonio

August 15, 2009

Aggregate Functions

Example

Find the total number, the average, minimum, and maximum GPAs of students who are 17.

```
select count(*), avg(GPA), min(GPA), max(GPA)
from Students where Age = 17
```

Aggregate functions are count(), sum(), max(), min(), avg()

Example

Find ids and names of students who have taken 5 or more courses.

```
select SID, Name from Students s
where 5 <= (select count(distinct Cno)
from Enrollment where SID = s.SID)
```

Outline

- 1 More Query Statements
 - Additional Clauses
 - Recursion
- 2 More Schema and Updates
 - More Complex Schema
 - More About Data Updates
 - Sequence
 - SQL*Loader
- 3 Views and Access Control
 - View and View Update
 - Access Control

Group

Example

List ids and names of students and the number of hours they still need to fulfill the 120-hour requirement.

```
select s.SID, Name,
       120-sum(Hours)HoursNeeded
from Students s, Enrollment e, Courses c
where s.SID = e.SID and e.Cno = c.Cno
and Grade <= 'C'
group by s.SID, Name
```

- Select clause must only contain grouping attributes and aggregate functions

Choose Groups for Output

Example

If an age group has more than 50 students, list the age and the average GPA of students of that age group.

```
select Age, avg(GPA)
from Students
group by Age
having count(*) > 50
```

The aggregate functions are applied to each age group.

Order Output Tuples

Example

List students whose GPA are higher than 3.5, sorted in descending GPAs then ascending names

```
select * from Students
where GPA > 3.5
order by GPA desc, Name asc
```

The default order is ascending order

Conceptual Evaluation (Revisit)

The six clauses of an SQL query is evaluated conceptually in following order.

- 1 Evaluate From (using cross product \times)
- 2 Evaluate Where (using selection σ)
- 3 Evaluate Group By (by forming groups)
- 4 Evaluate aggregate functions (in Select and in Having) for each group
- 5 Evaluate Having (by choosing groups for output)
- 6 Evaluate Select (using projection π)
- 7 Evaluate Order By (by sorting output tuples)

Recursive Query

Example

Given table ParentOf(Parent, Child). Find ancestors of Mary.

```
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';
```

- Use with statement to define a table before specifying a query.

Create Table Using a Query

Example

Create a FullProfessor table based on Faculty table.

```
create table Full-Professors as
  select FID, Name, Office
  from Faculty
  where Rank = 'Full Professor';
```

Assertions

Example

No student is allowed to take more than six courses

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

Assertion is typically used to define constraints that involves multiple tables.

Object Type

Example

Define an object type to represent names.

```
Create type name_t as object (
  fname varchar2(10),
  lname varchar2(10) );
```

Use the name_t type in table schema

```
Create table Persons (
  PID char(9) not null,
  name name_t,
  age number);
```

Update Involving Objects

Example

Insert a tuple into Persons

```
Insert into Persons
  values(123456789, name_t('John', 'Smith'), 33);
```

Example

Query a table that contains objects

```
Select p.name.fname
From Persons p
Where p.name.lname like '%Smith%' and age < 40;
```

Update Data Using a Query

Example

Add students whose GPAs are at least 3.8 into an already created table Top_Students(SID, Name, GPA).

```
insert into Top_Students
select SID, Name, GPA
from Students where GPA >= 3.8;
```

Example

Delete students who take no course.

```
delete from Students
where SID not in (select SID from Enrollment);
```

Use Of Sequences

A sequence can be accessed using two functions:

- seq_name.nextval returns the next value of seq.
- seq_name.currval returns the current value of seq, after the first use of the nextval function

Example

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

Sequence

Example

Create 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

SQL*Loader

- sqlldr is a Unix command for loading data from a data file (of type .dat) into an Oracle database table.

```
sqlldr userid/passwd control=foo.ctl
```

- Other options: direct (direct load), skip (skip n lines), load (load m lines) Will generate .log .bad files.
- Requires two files: control file (with extension .ctl) and data file (with extension .dat)

Using SQL*Loader

Example

Control file

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

data file

```
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
```

Load Sequence and Object

Example

To load data with a sequence seq, In the control file, include the sequence function

```
(SID"seq.nextval", name, age, gpa, major)
```

In data file, keep the corresponding field empty

```
(,John Smith,18,3.25,CS)
```

To load objects, specify object structure in control file

```
(SID, name object(fname, lname), age)
```

Define and Use a View

Example

Define a view for students whose GPA is at least 3.8

```
create view Top_Students as
select SID, Name, GPA
from Students
where GPA >= 3.8
```

Find top students whose names begin with a 'K'.

```
select SID, Name
from Top_Students
where Name like 'K%'
```

Evaluation of Queries on Views

Example

Evaluate query

```
select SID, Name from Top_Students
where Name like 'K%'
```

by folding it into view definition

```
select SID, Name
from (select SID, Name, GPA
      from Students where GPA >= 3.8)
where Name like 'K%'
```

and simplify it into

```
select SID, Name from Students
where GPA >= 3.8 and Name like 'K%'
```

Purpose of Using Views

- Providing data independence. Database applications based on views are shielded from changes in base table schemas.
- Protecting sensitive data. Sensitive attributes of base tables can be hidden by views.
- Improving user productivity. Frequent queries can be used to define views to simplify user queries.

Example of Views

Example

- Data independence: Applications need no change whether Age is stored or derived.
- Privacy protection: Phone and Birthday of students are hidden from users.
- Productivity improvement: “Find all courses taken by a given student” is much simpler

```
select Cno, Title
from Student_Course
where SID = X
```

Example of Views

Example

Suppose we have following base tables.

Students (SID, Name, Birthday, GPA, Phone)

Enrollment (SID, Cno, Grade)

Courses (Cno, Title, Hours, Dept)

and define a view

```
create view Student-Course as
select SID, Name, Age(Birthday) Age,
       GPA, c.Cno, Title
from Students s, Enrollment e, Courses c
where s.SID=e.SID and e.Cno = c.Cno
```

View Updates

Example

Given following view

```
create view Student-Course as
select SID, Name, Age(Birthday) Age,
       GPA, c.Cno, Title
from Students s, Enrollment e, Courses c
where s.SID=e.SID and e.Cno = c.Cno
```

Can we change data by

```
insert into Student-Course
values (1234, 'Dave Hall', 32, 3.15, 'CS334', 'B')
```

- Cannot update through view defined with multiple tables or aggregate functions.

Grant

Used to control user privileges

Examples

Grant select and insert access on Students table to John .

```
grant select, insert on Students to john
```

Grant all privileges on Students to user John and Terry.

```
grant all on Students to john, terry
```

Allow all users 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
```

Rules on 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.

Revoke

Granted privileges can be revoked

Example

Revoke all privileges granted to John

```
revoke all on Students from john
```

- Owner's privileges can not be revoked.
- Use *cascade* to remove privileges granted with *with grant* option

Role

Definition

A role is a named collection of privileges that can be granted to users as a unit.

Example

Define a role for TA and grant the role to two TAs

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