SQL: Interactive Queries (2)

Prof. Weining Zhang
Cs.utsa.edu

Aggregate Functions

- Functions that take a set of tuples and compute an aggregated value.
- Five standard functions:

  count, min, max, avg, sum
- They ignore null values.
- Find the total number, the average, minimum, and maximum GPA of students whose age is 17.

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

Aggregate Functions (cont.)

- Find id and name of students who take 5 or more courses.

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

- ☛ Count(distinct Cno) ≠ distinct count(Cno). Why?
- ☛ Must make sure the subquery generates a value comparable in the predicate.

Group By Clause

- List id and name of students together with the number of hours still needed to graduate, assuming 120 hours are required.

```
select s.SID, Name,
       120 - sum(Hours)  Hours-Needed
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
```

- ☛ Enrolled courses are grouped by students.

Group By Clause (cont.)

- Aggregate functions often applied to groups.
- One tuple is generated per group
- When using group by, select clause can contain only grouping attributes and aggregate func.
- Every grouping attribute must be in the select clause. The following is an illegal query (why?):

```
select Age, SID, avg(GPA)
from Students
group by Age
```


Having Clause

- For each student age group with more than 50 members, list the age and the number of students with that age.

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

- ☛ Conditions on aggregate functions are specified in the having clause.
- ☛ Select & Having may have different functions.

Order By Clause

- List student names in ascending order.

```
select Name from Students
order by Name asc
```

- The default is ascending order.
- List students with GPA higher than 3.5, first in descending order of GPA, and then in ascending order of name.

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


Some Complex Queries

- Find the average number of CS courses a student takes.
- For non-CS major students who take more CS courses than he does with his major courses, and have taken at lease 2 CS courses, list their id, name, number of CS courses, number of major courses, sorted first in descending order of number of CS courses, then in ascending order of name.

Interactive SQL Summary

- A query may have six clauses: select, from, where, group by, having, order by.
- Conceptual evaluation of the query:
  1. Evaluate From (cross product)
  2. Evaluate Where (selection)
  3. Evaluate Group By (form groups)
  4. Evaluate Aggregate functions on groups
  5. Evaluate Having (choose groups to output)
  6. Evaluate Order By (sorting)
  7. Evaluate remaining Select (projection)

Interactive SQL Summary (count.)

- Many ways to express a query.
  - Flat queries may be more efficient.
  - Nested queries may be easier to understand.
- Duplicate elimination may be costly.
- <> (not equal) at predicate level often gives a wrong answer. Use set difference, not in, not exists, etc. instead.
- Need to handle null values explicitly.
- DBMSs often provide many convenient functions. But need to check the compatibility.

Expressive Power of SQL

- SQL is relational complete.
  - Can express any relational algebraic query.
- SQL is more powerful then relational algebra.
  - Can express aggregation, ordering, recursion, etc.
- SQL is not computational complete.
  - Can not do everything a general programming language can do.

Create Table Re-visited

- Can combine table creation with insertion of tuples using a query.

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

Update By Queries

- Relation: Top_Students (SID, Name, GPA)
- Insert students with a GPA 3.8 or higher into the Top_Students table.

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

- Delete all students who take no courses.

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


Update Statement

- For every student who takes Database I, set the Grade to 'A'.

```
update Enrollment
set Grade = 'A'
where Cno in
    (select Cno
     from Courses
     where Title = 'Database I')
```

Truncate vs Delete *

- Use delete to remove data and keep the table storage space.

```
delete from Departments;
```

- Use truncate to remove data and release table storage space.

```
truncate table Departments;
```


Views

- A view is a virtual table (as opposed to stored base table) defined by a query, directly or indirectly, on base tables.

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

- A view may be defined in terms of other views.

Views (cont.)

- The query in view definition is usually not executed until the view is queried. Typically, no data is stored for a view.
- A view is queried as if it is a base table.
- Find name and GPA of top students whose name starts with a `K'.

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


Query Modification

- Queries on a view are translated into queries on base tables by folding the view.
- Previous query is translated first into:

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

Then into

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

Why Use Views?

- Data independence: keep existing application programs from changes of base table schemas.
- Access control: provide a mechanism for hiding sensitive data from certain users.
- Productivity improvement: make user queries easier to express.

Example of Using Views

Consider following base tables and a view:

Students (SID, Name, Birthday, GPA, Phone)
Emrollment(SID, Cno, Grade)
Courses(Cno, Title, Hours, Dept)

```
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
```

Example of Using Views (cont.)

- Data independence: Applications using the view are not affected if Age is stored or derived.
- Access control: 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
```


Views and Updates

- What should happen if a user changes the data in the Student-Course view?

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

- A view can not be updated if
  - Contains group by and aggregate functions
  - Involves multiple tables
- A single-table view can be updated if it contains a key of the table

View Update Example *

- Which student should be deleted?

```
create view Age_distribution
as select Age, count(*) TotalNo
   from Students group by Age
update Age_distribution
set TotalNo = TotalNo – 1 where Age = 20
```

- Which base relation should be changed?

```
delete from Student_Course
where SID = '1234'
```


Maintaining Materialized Views

- One may want to *materialize* a view (i.e., run its definition query and store the result) as is commonly done in industry (data warehouse). (Why?)
- *View Maintenance*: How to maintain the consistency between a view and its base tables, when base tables are updated?
- *Incremental View Maintenance*: How to maintain a view without re-computing the entire view?