SQL - Data Query language

Eduardo J Ruiz

October 20, 2009

1 Basic Structure

- The simple structure for a SQL query is the following:

```
select a1...an
from  t1 .... tr
where C
```

  Where $t_1 \dots t_r$ is a list of relations in the database schema, $a_1 \dots a_n$ the elements in the resultant relation and C a list of constraints.

- a_i can be
  1. A constant
  2. An attribute in the resultant relation
  3. A function applied to one attribute or several attributes
  4. An aggregation function
  5. * if we want to list all the attributes of a particular relation
- n is the arity of the resultant relation.
- C contains the list of restrictions that enforces the joins and the selections
- Some queries are equivalent to $\pi_{a_1 \dots a_n}\sigma_C(t_1 \times t_2 \dots t_r)$

  Select all the attributes from the students with a GPA greater than 3.0

  $\sigma_{gpa>3.0}student$

```
select *
from  student
where gpa > 3.0
```

  The following query use a function to present the student full-name (concat is a mysql function)

  $\pi_{pantherid,name+lastname}(\sigma_{gpa>3.0 \wedge gender='F'}(student))$

```
select panterid, concat(name,' ',lastname)
from  student
where gpa > 3.0 and gender='F'
```

Cartesian product. All the students with all the courses.

$\sigma_{pantherid,courseid}(student \times enrolled)$

```
select panterid, courseid
from  student,enrolled
```

- Query construction:
  1. Select all the useful tables from the schema
  2. Add the necessary join conditions (this is the most important step)
  3. Add the necessary filter conditions
  4. Select the attributes (apply fancy functions as needed)
  5. Sort the results

2 Renaming

- You can rename tables and attributes
- Why: to avoid ambiguity (two tables with the same attributes, self joins), to use clearer names, to add names to anonymous columns

  In the following example we rename the concatenated attribute to full name and we add an extra attribute. Notice the complexity of the almost equivalent relational algebra expression)

  $\rho(fellows(1 \rightarrow id, 2 \rightarrow fullname))(\pi_{pantherid,name+lastname}(\sigma_{gpa>3.0 \wedge gender='F'} student))$

```
select s.panterid as id, concat(s.name,' ',s.lastname) as fullname, 'fellow' as description
from  student s
where gpa > 3.95
```

  In the following example we rename the tables to do a self join

  $\pi_{s2\_name}\sigma_{s\_gpa>s2\_gpa \wedge s\_name='Jhon'}\rho(ASP(1 \rightarrow s\_name, 4 \rightarrow s\_gpa, 5 \rightarrow s2\_name, 8 \rightarrow s2\_gpa)$
  $(\rho(S, student) \times \rho(S2, student))$

```
select s2.name
from  student s, student s2
where s.gpa > s2.gpa and s.name='Jhon'
```

3 Distinct

- Finding duplicates hurts the performance. For this reason the SQL query language does not remove duplicates
- Sometimes we want to remove duplicates from our output. You can force this with the DISTINTC clause
- Find all the courses that have failed at least one student in fall09.

  $\pi_{courseid}(\sigma_{grade='F' \wedge term='FALL09'}(Student \bowtie_{pantherid} enrolled))$

```
select distinct courseid
from  enrolled e
where e.grade = 'F' and e.term='FALL09'
```

- Hint: Apply the distinct clause after you finish with the main body
- Hint: Do not use distinct when you don't need it (the optimizer is not as smart as you think)

4 Order by

- When: sorting reports
- Hint: in most case you don't want to sort in your application (the DBMS optimize sorts for disk storage)
- Order By: sort the result using a list of attributes $a_1 \ldots a_k$
- Sort by a_1 ,breaks ties with a_2, breaks ties with a_3 .... breaks ties with a_n. You must be careful with the default definition of order for each data type
- You can use the keywords ASC and DESC to control if the results are sorted in ascending or descending order. In MySQL the default order is ascendant.

```
select e.pantherid,s.name, e.grade, e.gpa
from  student s, enrolled e
where (e.term='FALL09' OR e.term='SPRING09')
       and e.courseid='COP5725'
       and s.pantherid = e.pantherid
order by grade, gpa desc
```

- Hint: Sort after you finish with the main body

5 UNION, INTERSECT, EXCEPT

Read the book

6 Nested Queries

- In theory, a select statement returns a relation. Some DBMS allows to use this result in the FOR clause.
- We can also use **correlated** or **uncorrelated** queries. In this case the sub-queries appears in the WHERE clause as a **condition**
- The **att IN sub-query** condition returns true if the value of the attribute is in the resultant sub-query relation. (You can think this as a look-up table). Notice that the arity of the sub-query should be the same as the number of atts
- The **EXISTS sub-query** returns true exists at least one tuple in the resultant sub-query relation. (Correlated Queries, PLUG a tuple and TEST if the circuits is On/Off)
- The **att $>$ ANY sub-query** returns true if exists at least one tuple t in the resultant sub-query relation, such that $att > t$
- The **att $>$ ALL sub-query** returns true if for all the tuples t in the resultant sub-query relation, we have $att > t$

  Select all the students that are not recipient of the presidential fellowship and have a gpa greater than 3.7

```
select pantherid
from student
where gpa > 3.7 and pantherid not in (select sid
                                      from fellowship f
                                      where f.name = 'presidential')
```

```
select s.pantherid
from student s
where s.gpa > 3.7 and not exists (select *
                                  from fellowship f
                                  where f.name = 'presidential' and f.sid = s.pantherid )
```

  Select the student with the greatest GPA

```
select s.pantherid
from student s
where not exists (select *
                  from student s2
                  where s2.gpa > s.gpa)
```

  Select the student with the best grade in the database course (grades should be represented as numbers)

```
select s.name
from student s,enrolled e
where s.pantherid = e.pantherid  and
      e.name ='COP5725'          and
```

```
e.term ='FALL09'           and
e.grade >= ALL(select e2.grade
               from enrolled e2
               where e2.name ='COP5725'  and
                     e.term  ='FALL09')
```

- Both queries could have been expressed using as joins. Always write with joins if possible
- Efficiency depends on the size of the sub-query and the number of tuples that are tested. Sometimes a IN can be faster, sometimes EXISTS could be faster

7 Aggregation Functions

- COUNT, SUM, AVG, MAX , MIN
- Aggregates the values of one column in the resultant relation
- You can not combine aggregation attributes with non grouped columns (you must use a group by)

```
select avg(gpa),count(*),max(gpa),min(gpa)
from student

select name
from student
where gpa in (select max(gpa)
              from student)
```

8 Group By

- The **GROUP BY att** clause split the table using the attributes in *att*
- All the tuples that have the same values in these GROUP BY attributes are part of the same group
- A simple implementation: sort by the same attributes and find the groups
- You combine the GROUP BY clause with aggregation functions to collapse the group in one tuple (think as the attributes in the clause are keys of the group and apply the function to the elements in this group)

```
select courseid,term,count(*)
from courses
group by courseid, term

select courseid,avg(gpa)
from student s ,enrolled e
where e.pantherid = s.pantherid
group by courseid
```

9 Having

- Sometimes you want to make a comparison of the values of the grouped tuples to filter the results
- Examples:
  1. Select the courses such that the average grade is above some letter
  2. Select all those students such that took at least 10 courses
  3. Find those fellowships such that the average gpa of all the recipients is greater than 3.7
- Hint: Find the groups. Apply the extra filter in the having.

```
select f.name, avg(gpa)
from fellowship f , student s
where s.pantherid = f.sid
group by f.name
having avg(gpa) > 3.7
```

TYPICAL ERROR (THE AVG FUNCTION CAN NOT BE USED IN THE WHERE CLAUSE!!!)

```
select f.name,
from fellowship f , student s
where s.pantherid = f.sid and avg(gpa) > 3.7
```

10 Alternative Join Definition

- The SQL standard allows a syntax for defining joins that is more clear that the Cartesian semantics
- Joins are defined explicitly in the FROM clause
- Joins have different types: natural, inner, outher. The syntax also make easy to set hints to the optimizer

```
select distinct pantherid, courseid
from student s join enrolled e  using (e.pantherid = s.pantherid)

select distinct pantherid, courseid
from student s natural join enrolled e
```