Introduction to Querying

Databases for Many Majors

Suzanne W. Dietrich
Arizona State University

Don Goeltenen
Villanova University

Version 4
August 2016

Choose a topic on the left to run the tutorial interactively or
Run Tutorial in Demonstration Mode

*This material is based upon work supported by the National Science Foundation under Grant No. DUE-1431848, DUE-1431661, DUE-0941584, and DUE-0941401. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Last Revision: July 2017
Tables

Note that Name is a reserved word in Access so the attribute in Students changed to SName.

![Students Table](image1)

![Courses Table](image2)

![Take Table](image3)
Creating a Query in Access

Example Query – Which students took "College Algebra"?

Choose the Create tab and then Query Design:

- This brings up a popup: Show Table
• Add the tables that you need to answer the query: *Courses, Take, Students*

![Diagram of tables and relationships](image1.png)

Note that Access visually shows the primary keys in gold and the primary-foreign key relationships using links, which means that the query will be joining on these values. *Also, the 1 and ∞ labels on the link between CrsID in Courses to CrsID in Take indicate that a CrsID value may appear many times in Take and the CrsID from Take appears only 1 time in Courses. (See the Conceptual Design visualization for more information.)*

• Drag and drop the attributes/fields that you want to horizontally or vertically filter to answer the query: *The query asks to see all Student attributes in the result, so use the * shortcut (just like SQL) to drag those attributes to the field part of the query specification. Since the query requires horizontal filtering on the value of the CrsTitle, drag that attribute as well.*

![Diagram with selected attributes and criteria](image2.png)

• Select the **Show** box for attributes that you want to see in the result of the query: *Students.* shows all attributes of Students – Id, SName, Class, Major

• Specify **Criteria** for attributes to be horizontally filtered: *CrsTitle = "College Algebra"*
• Save the query with a descriptive name:
  
  *CollegeAlgebraStudents*

• Run the query to see the results

SQL:

```sql
SELECT S.Id, S.Name, S.Class, S.Major
FROM Courses C, Take T, Students S
WHERE C.CrsId = T.CrsId AND T.Id = S.Id AND C.CrsTitle = "College Algebra"
```

The remainder of this document will show the screen captures from Access along with the corresponding SQL in MySQL to compute the answer to the queries in the Introduction to Querying animation.

*Note:* The graphical interface for designing queries in Access is known as Access QBE – Query By Example. In Access, this view is called the Design View. When you Run a query, it shows the Datasheet View. The View Menu for a query also indicates that there is an SQL View. You can see the generated SQL for the graphically designed query. Some queries cannot be represented graphically so you can choose to answer the query only using SQL.

Below is a screen capture of the Access SQL View for the CollegeAlgebraStudents query:

In this query, Access SQL is using the JOIN in the FROM clause. The JOIN is prefixed by the word INNER, which represents the default type of JOIN. Other types of joins are advanced SQL topics and are beyond the coverage of the visualizations. The SQL also shows extra parentheses in the WHERE clause, which are unnecessary and somewhat typical in generated code.
**Topic**: Query  
**Subtopic**: Query | Example  
**Query**: Find the semester that "Jeff Carter" took "CSE 303"  

**Access**:

SQL:

```sql
select T.Semester 
from Students S, Take T 
where S.Id = T.Id and 
  S.Name = "Jeff Carter" and 
  T.CrsID = "CSE 303"
```

**Results**:
**Topic:** Sets

**Subtopic:** Sets | Intro

**Query:** CSE returns unique Id of Students taking CSE courses

**Access:**
- Criteria for CrsID: LIKE "CSE*
- Property Sheet: Unique Values: Yes

**SQL:** The keyword **distinct** provides unique values in the result.

In SQL, a named query is represented as a view, which is defined once and re-executed when referenced. Note that MySQL uses the % sign as the wildcard to match the rest of the string.

```sql
create view CSE as
select distinct ID
from Take
where CrsID LIKE "CSE%";
```

**Results:**

Similarly, for students taking MAT courses.
Subtopic: Sets | Union

Query: Id of students who took CSE or MAT courses

Access: Union Query

Access QBE does not support set operations: union, negation, intersection. However, Access does support the specification of the union query in SQL.

SQL: Note that the * symbol represents a shortcut for selecting ALL attributes from a table, which is just Id in this example.

select * from CSE
union
select * from MAT;

Result:
**Subtopic:** Sets | *Negation*

**Query:** Ids of students who have taken CSE courses and **not** MAT courses

**Access:**

Negation set-based queries are not inherently supported in Access QBE or Access SQL. However, there are typically multiple ways of answering a query. In SQL, shown below, nested queries provide an alternative to answering a negation query – asking for those students who took CSE that are not in the subquery asking for the students who took MAT. For the shown Access SQL query, CSEnotMAT, there is a Design View available that represents a hybrid query between Access QBE and Access SQL.

**Access SQL**

![Access SQL](image)

**SQL:**

```
select * from CSE
except
select * from MAT;
```

**Access QBE**

![Access QBE](image)

```
select *
from CSE
where Id not in (select Id from MAT);
```

**Result:**

![Result](image)
**Subtopic:** Sets | Intersection

**Query:** Ids of students who have taken CSE courses and MAT courses

**Access:**

Intersection queries are not inherently supported in Access QBE or Access SQL. However, there are typically multiple ways to find an answer to a query. In this case, the same result can be obtained by joining CSE and MAT so that the value of the Id attributes are equal.

- On the Show Table popup, choose the Queries tab and then select both CSE and MAT
- To join on Id, select the Id attribute in CSE and drag it to the Id attribute in MAT

**SQL:** There are multiple ways of answering this query in SQL including using a nested query:

<table>
<thead>
<tr>
<th>Query 1</th>
<th>Query 2</th>
<th>Query 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>select * from CSE intersect select * from MAT;</td>
<td>select * from CSE natural join MAT;</td>
<td>select * from CSE where Id in (select Id from MAT);</td>
</tr>
</tbody>
</table>

**Result:**

![Result Image]
Topic: Filtering

Subtopic: Filtering | Horizontal

Query: Find the students who are "Math" majors

Access:

SQL: Recall that the * symbol represents a shortcut for selecting ALL attributes from a table.

```sql
select *
from Students
where Major = "Math";
```

Result:
Subtopic: Filtering | Vertical

Query: Retrieve the Name and Class of all students

Access:

SQL:

```sql
select Name, Class
from Students;
```

Result:
**Subtopic**: Filtering | *Combined*

**Query**: Find the Name and class of students who are "Math" majors

**Access**:

```sql
select Name, Class
from Students
where Major = "Math";
```

**Result**:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred Hopewell</td>
<td>Freshman</td>
</tr>
<tr>
<td>Valerie Dunbar</td>
<td>Freshman</td>
</tr>
</tbody>
</table>
```
**Topic:** Joining  
**Subtopic:** Joining | CartesianProduct  

**Query:** Illustrating a Cartesian product of students who have taken a MAT course with a table representing the vertical filtering of Id and Course on the StudentsTakingCourses table.

**Access:**

**SQL:**

```sql
create view NameldStudentsMAT as select distinct S.SName, S.Id as SId from StudentsTakingCourses T, Students S where Course LIKE "MAT%" and T.Id = S.Id;
create view TakesMATCourse as select Id as TId, Course from StudentsTakingCourses where Course LIKE "MAT%";
select * from NameldStudentsMAT, TakesMATCourse;
```

**Result:**

<table>
<thead>
<tr>
<th>SName</th>
<th>SId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred Hopewell</td>
<td>3333</td>
</tr>
<tr>
<td>Andrew Spoth</td>
<td>4444</td>
</tr>
<tr>
<td>Valerie Dunbar</td>
<td>5555</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TId</th>
<th>CrsId</th>
</tr>
</thead>
<tbody>
<tr>
<td>3333</td>
<td>MAT 118</td>
</tr>
<tr>
<td>3333</td>
<td>MAT 243</td>
</tr>
<tr>
<td>4444</td>
<td>MAT 118</td>
</tr>
<tr>
<td>5555</td>
<td>MAT 118</td>
</tr>
</tbody>
</table>
Subtopic: Joining | Join

Query: Illustrating a join of students who have taken a MAT course with a table representing the vertical filtering of Id and CrsId on the Take table.

Access: To join on SId and TId, drag SId to TId to create the join link.

SQL:

```
select *
from   NameldStudentsMAT join TakesMATCourse on SId = TId;
```

Result:
**Subtopic: Joining | NaturalJoin**

A natural join is a shortcut for joining two tables such that the columns with the same name are equal, including only one copy of that attribute in the result. Therefore, this shortcut is available in SQL. In Access QBE, you would not show the extra attribute. Assuming that the Id attributes of NameIdStudentsMAT and TakesMATCourse are renamed to be the same:

**SQL:**

```sql
select  *
from    NameIdStudentsMAT natural join TakesMATCourse;
```

**Result:**

![NaturalJoin](image_url)
**Topic: SQL**

**Subtopic: SQL | Select**

**Access:** see Subtopic: Query | Example for Access screenshots

**SQL:**

```
select  T.Semester
from     Students S, Take T
where    S.Id = T.Id and
         S.Name = "Jeff Carter" and
         T.CrsID = "CSE 303"
```

**Subtopic: SQL | Sets**

**Access:** see Topic: Sets for Access screenshots of Union, Negation, and Intersection

**SQL:**

Assuming that the views of CSE and MAT are defined, the following represents specifications in the SQL standard for:

- **Union**

  ```
  select * from CSE
  union
  select * from MAT
  ```

- **Negation**

  ```
  select * from CSE
  except
  select * from MAT
  ```

- **Intersection**

  ```
  select * from CSE
  intersect
  select * from MAT
  ```

MySQL does not support except and intersect. See Subtopic: Sets | Negation and Subtopic: Sets | Intersection for the associated MySQL.
Subtopic: SQL | Postscript

Concept: Ordering

In practice, it is very important to order the results returned by a query for ease of interpretation. In SQL, there is an `order by` clause to order the results of the query. The default ordering of the listed attributes is ascending order when no keyword is specified. To change the ordering to descending, use the keyword `desc`.

**SQL:**

```
select *
from Students
order by Major desc;
```

**Access:**

Access also has the ability to order the results. Use the drop-down box under **Sort** to choose **Ascending** or **Descending**.

---

**Results:**

<table>
<thead>
<tr>
<th>SName</th>
<th>Id</th>
<th>Class</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valerie Dunbar</td>
<td>5555</td>
<td>Freshman</td>
<td>Math</td>
</tr>
<tr>
<td>Fred Hopewell</td>
<td>3333</td>
<td>Freshman</td>
<td>Math</td>
</tr>
<tr>
<td>Andrew Spoth</td>
<td>4444</td>
<td>Junior</td>
<td>English</td>
</tr>
<tr>
<td>Anne Penny</td>
<td>2222</td>
<td>Senior</td>
<td>Computer Science</td>
</tr>
<tr>
<td>Jeff Carter</td>
<td>1111</td>
<td>Junior</td>
<td>Computer Science</td>
</tr>
</tbody>
</table>
Concept: Counting

Besides ordering of results, there are additional features to answering queries that are very important in practice. The visualization illustrates a counting query, which finds the number of tuples in the table:

- **count(*)** indicates to count all of the tuples in the table
- **as numberOfTuples** is renaming the column to a meaningful name, since the name of the counting column is not part of the SQL standard and will vary.

**SQL:**

```sql
select count(*) as numberOfTuples
from Students;
```

**Access:**

Access also has the ability to count. To see the Total row, under Query Design, select the ∑ Totals. Then use the drop-down box under **Total** to choose **Count.**

**Results:** Note that Access named the column CountOfId. You can rename columns in Access by prefixing the field with the newname:
**Concept:** Aggregation

The **count** operator is an example of an aggregation operator. The term “aggregation” refers to providing an “aggregate” or “one” answer based on the details. The other aggregate operators in SQL are: **min**, **max**, **avg**, and **sum**. These operators are typically performed on a numeric attribute to find the associated minimum, maximum, average, and sum. In the schema of the visualization, there is only one numeric column – the number of credits for each course. The following query finds the min, max, avg, and sum of the credits over the *entire* Courses table:

**SQL:**

```
select min(Credits) as mincredits, max(Credits) as maxcredits,
       avg(Credits) as avgcredits, sum(Credits) as sumcredits
from Courses
```

What if you wanted to find the total number of credits taken by each student? This type of query requires performing an aggregation over only parts of the information – called groups. SQL uses a **group by** clause to create these groups:

```
select Id, sum(Credits) as totalcredits
from Takes natural join Courses
group by Id;
```

To visualize each group based on the Id value, consider the following related query:

```
select Id, Credits
from Takes natural join Courses
order by Id;
```

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>2</td>
</tr>
<tr>
<td>1111</td>
<td>3</td>
</tr>
<tr>
<td>2222</td>
<td>3</td>
</tr>
<tr>
<td>2222</td>
<td>4</td>
</tr>
<tr>
<td>3333</td>
<td>3</td>
</tr>
<tr>
<td>3333</td>
<td>2</td>
</tr>
<tr>
<td>3333</td>
<td>3</td>
</tr>
<tr>
<td>4444</td>
<td>4</td>
</tr>
<tr>
<td>4444</td>
<td>3</td>
</tr>
<tr>
<td>5555</td>
<td>3</td>
</tr>
<tr>
<td>5555</td>
<td>2</td>
</tr>
<tr>
<td>5555</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Id</th>
<th>totalcredits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>5</td>
</tr>
<tr>
<td>2222</td>
<td>7</td>
</tr>
<tr>
<td>3333</td>
<td>8</td>
</tr>
<tr>
<td>4444</td>
<td>7</td>
</tr>
<tr>
<td>5555</td>
<td>8</td>
</tr>
</tbody>
</table>
```
**Access:**

Access also has the ability to aggregate over groups. To see the Total row, under Query Design, select the ∑ Totals.

- Drag the Id attribute from Take to the Field window. Under the Total drop down, select **Group By**
- Drag the Credits attribute from Courses to the Field window. Under the Total drop down, select **Sum**.
- Rename the result of the sum by prefixing the column name with **totalcredits**:
The Checkpoint section provides formative self-assessment to check your understanding of the concepts presented in the visualization. The checkpoint includes the following queries if you want to work them out in Access or MySQL:

- Assuming that the desired attribute in the result is Id, find the students who have taken "MAT 118"?
- Assuming that the desired attribute in the result is Id, find the students who are seniors, given by the classification "Senior"?
- Assuming that the desired attribute in the result is Credits, find the number of credits for the course titled "College Algebra"?
- Assuming that the desired attribute in the result is Semester, find the semesters that the course titled "College Algebra" has been offered?
- Assuming that the desired attribute in the result is Name, find the students who have taken "MAT 118"?
- Assuming that the desired attribute in the result is Name, find the students who have taken "College Algebra"?
- Find the students along with the courses that they have taken, returning all attributes.
## Introduction to Querying: Summary

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Symbol</th>
<th>Access</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>projection; vertical filter</td>
<td>( \pi )</td>
<td>( \sqrt ) in “show” box</td>
<td>select</td>
</tr>
<tr>
<td>selection; horizontal filter</td>
<td>( \sigma )</td>
<td>“criteria”</td>
<td>where</td>
</tr>
<tr>
<td>which tables are needed</td>
<td></td>
<td>Add Table</td>
<td>from</td>
</tr>
<tr>
<td>Cartesian product</td>
<td>( \times )</td>
<td></td>
<td>commas in “from” clause</td>
</tr>
<tr>
<td>(natural) join</td>
<td>( \bowtie )</td>
<td>automatic (links shown in relationships diagram)</td>
<td>condition in “where” clause (may need to prefix attribute names!)</td>
</tr>
<tr>
<td>union</td>
<td>( \cup )</td>
<td>“or” in “criteria”</td>
<td>“or” in “where” clause; union</td>
</tr>
<tr>
<td>intersection</td>
<td>( \cap )</td>
<td>“and” in “criteria”</td>
<td>“and” in “where” clause; intersect</td>
</tr>
<tr>
<td>negation</td>
<td>(-)</td>
<td>“and not” in “criteria”</td>
<td>“and not” in “where”; except</td>
</tr>
</tbody>
</table>