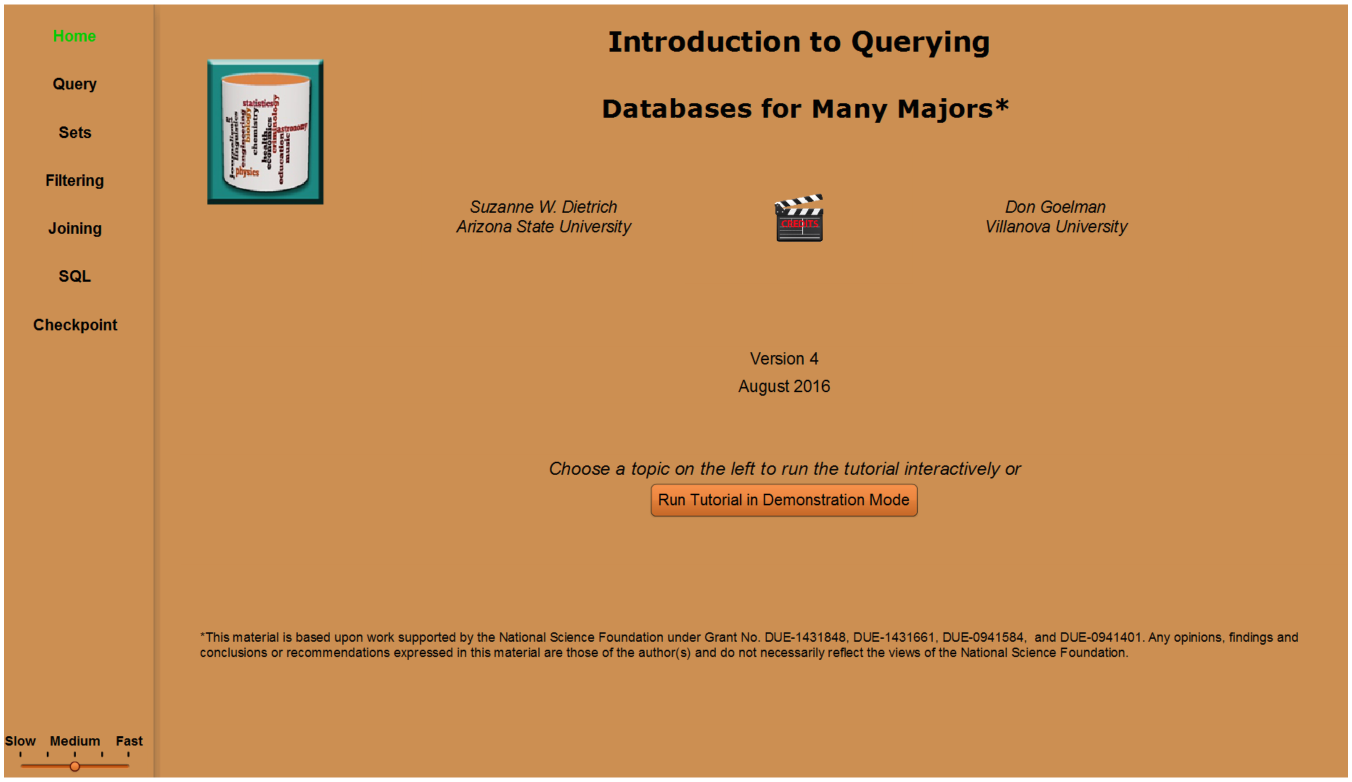
Databases for Many Majors

Queries in Access and SQL



Introduction to Querying

Databases for Many Majors*

Suzanne W. Dietrich
Arizona State University

Don Goelman
Villanova University

Version 4
August 2016

Choose a topic on the left to run the tutorial interactively or

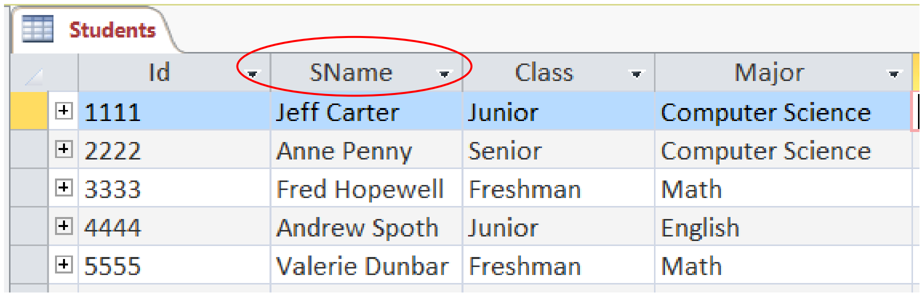
Run Tutorial in Demonstration Mode

*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

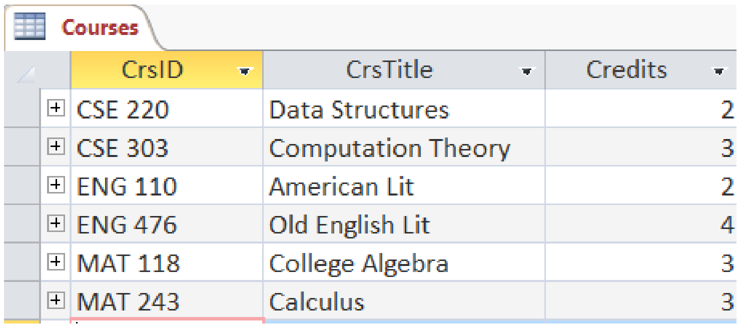
Tables

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



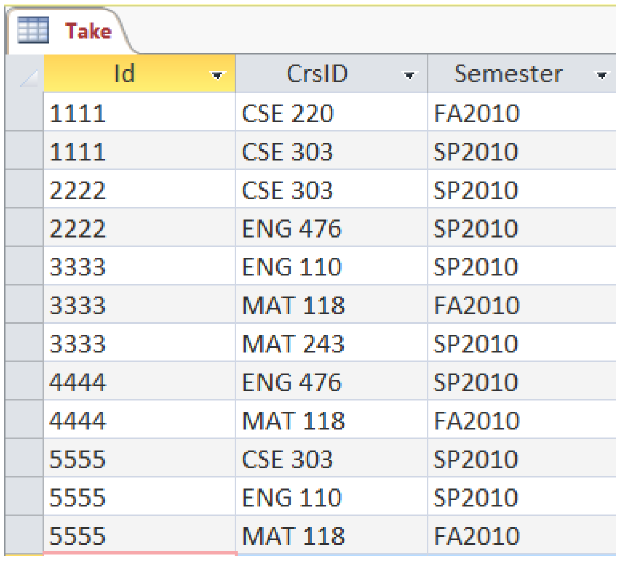
Students

Id	SName	Class	Major
1111	Jeff Carter	Junior	Computer Science
2222	Anne Penny	Senior	Computer Science
3333	Fred Hopewell	Freshman	Math
4444	Andrew Spoth	Junior	English
5555	Valerie Dunbar	Freshman	Math



Courses

CrsID	CrsTitle	Credits
CSE 220	Data Structures	2
CSE 303	Computation Theory	3
ENG 110	American Lit	2
ENG 476	Old English Lit	4
MAT 118	College Algebra	3
MAT 243	Calculus	3



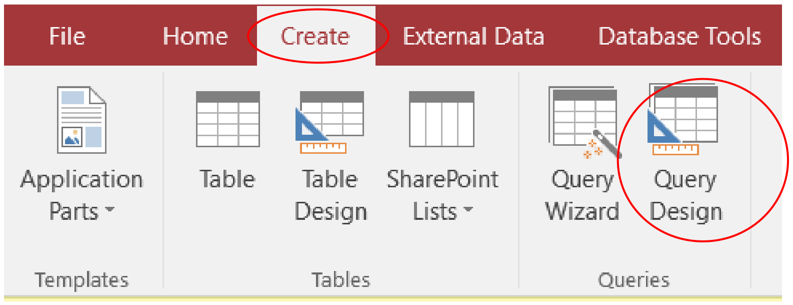
Take

Id	CrsID	Semester
1111	CSE 220	FA2010
1111	CSE 303	SP2010
2222	CSE 303	SP2010
2222	ENG 476	SP2010
3333	ENG 110	SP2010
3333	MAT 118	FA2010
3333	MAT 243	SP2010
4444	ENG 476	SP2010
4444	MAT 118	FA2010
5555	CSE 303	SP2010
5555	ENG 110	SP2010
5555	MAT 118	FA2010

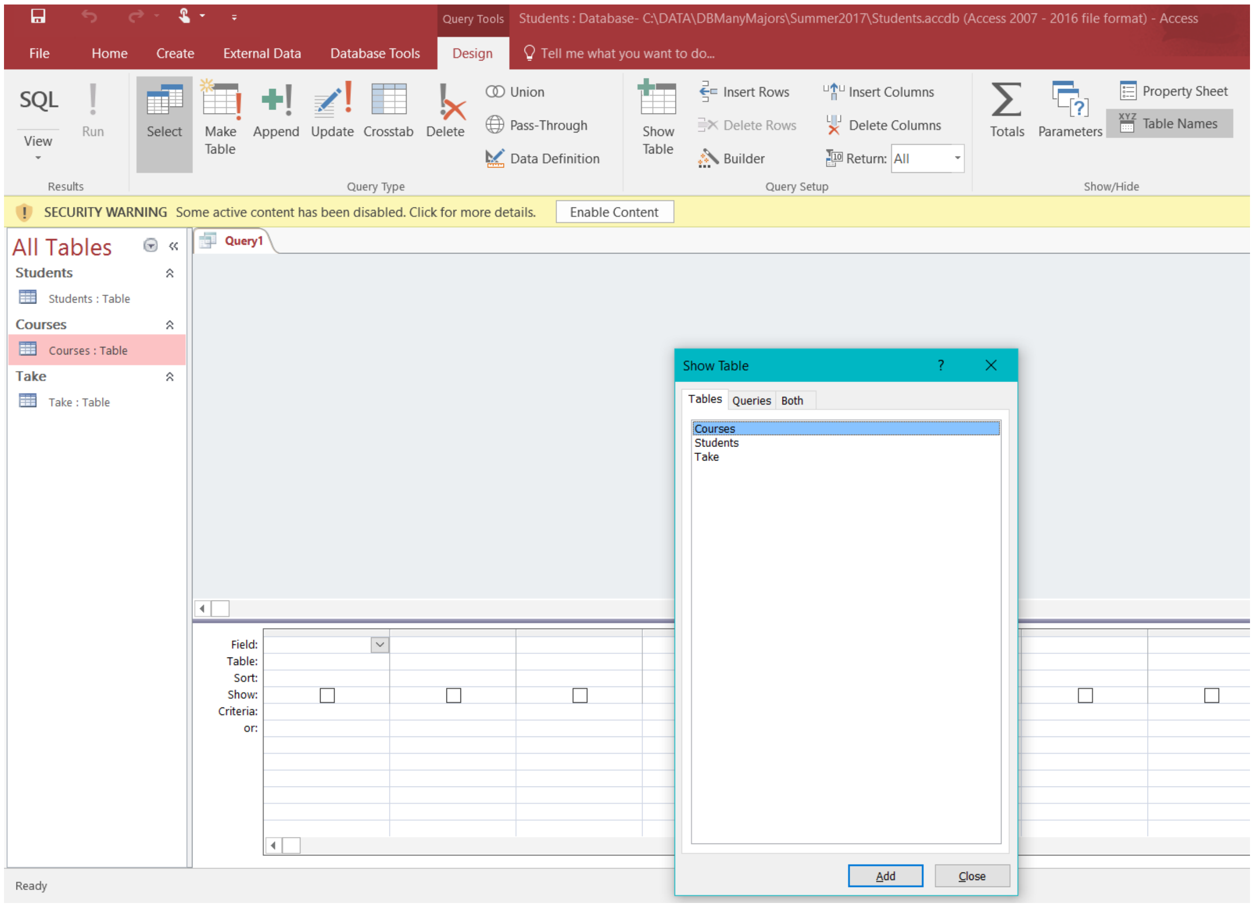
Creating a Query in Access

Example Query – Which students took "College Algebra"?

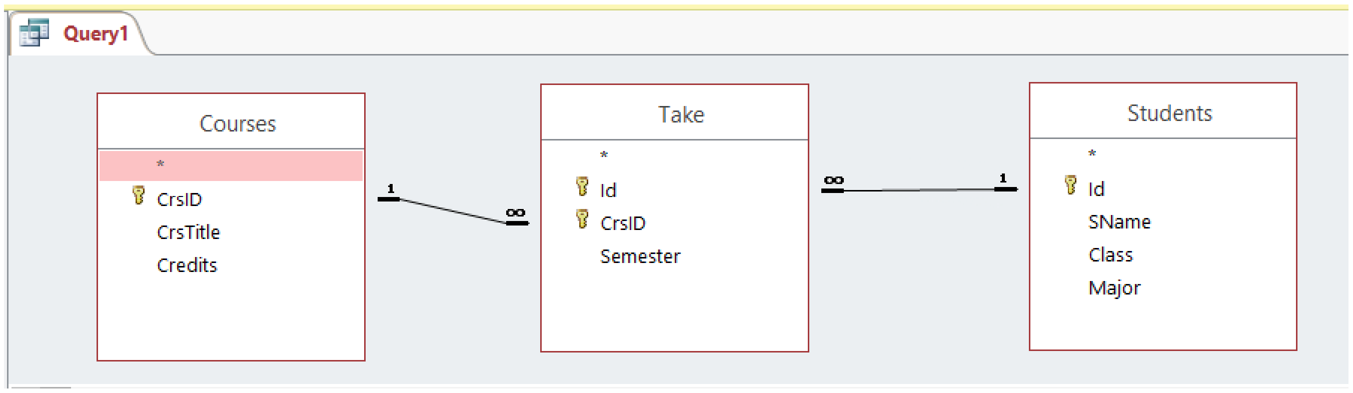
Choose the Create tab and then Query Design:



- This brings up a popup: Show Table

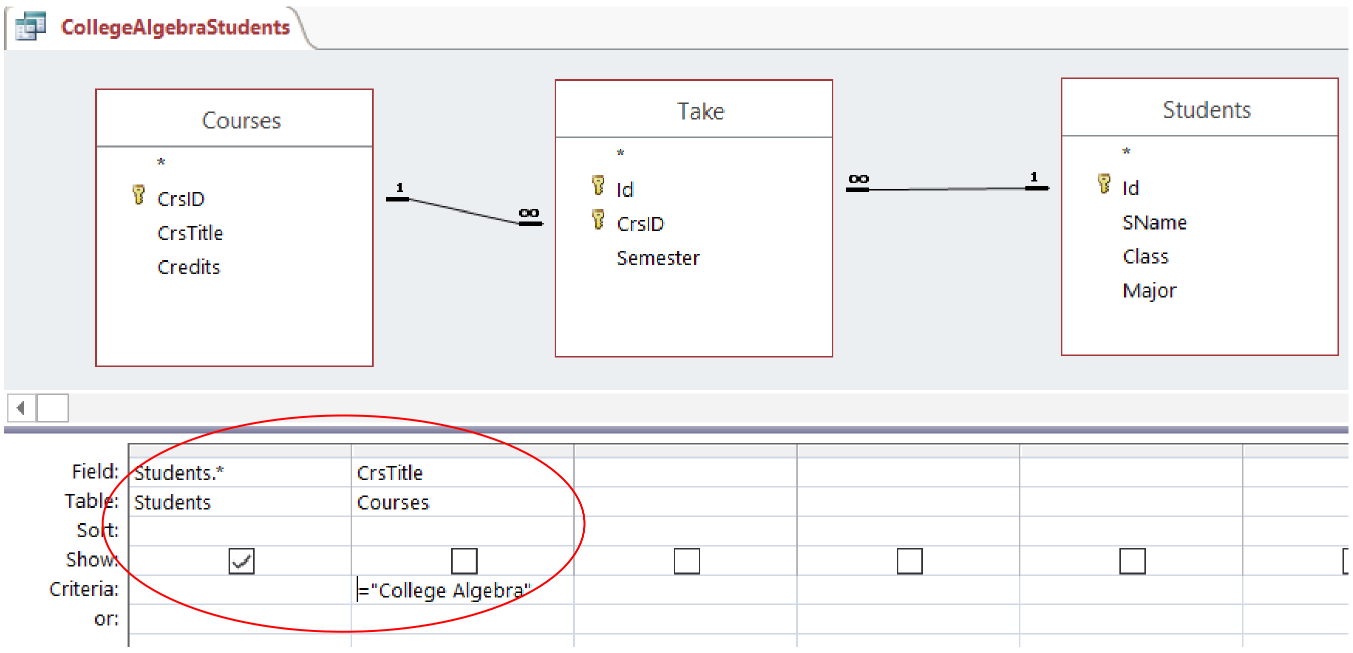


- Add the tables that you need to answer the query: *Courses, Take, Students*



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.*



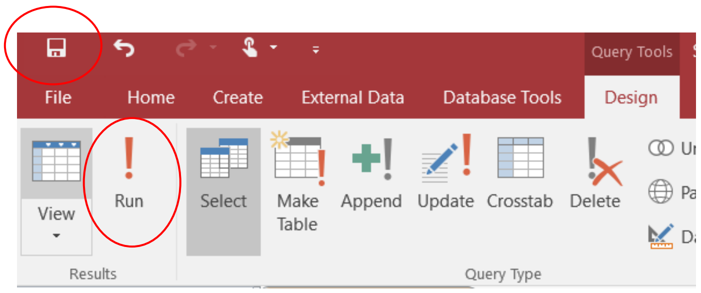
Field:	Students.*	CrsTitle
Table:	Students	Courses
Sort:		
Show:	✓	☐
Criteria:		="College Algebra"
or:		

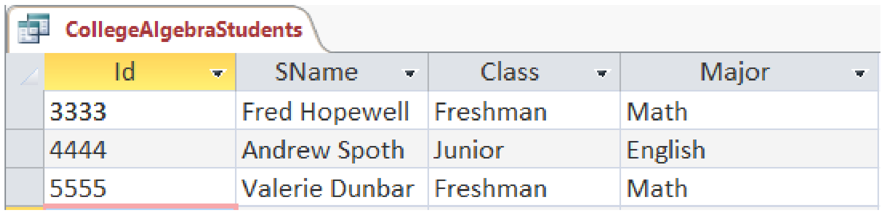
- Select the **Show** box for attributes that you want to see in the result of the query: *Students.* shows all attributes of Students – Id, SName, Class, Major*
- Specify **Criteria** for attributes to be horizontally filtered: *CrsTitle = "College Algebra"*

- Save the query with a descriptive name:

  CollegeAlgebraStudents

- Run the query to see the results





CollegeAlgebraStudents

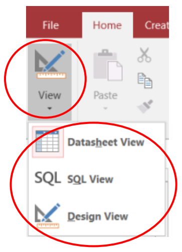
Id	SName	Class	Major
3333	Fred Hopewell	Freshman	Math
4444	Andrew Spoth	Junior	English
5555	Valerie Dunbar	Freshman	Math

SQL:

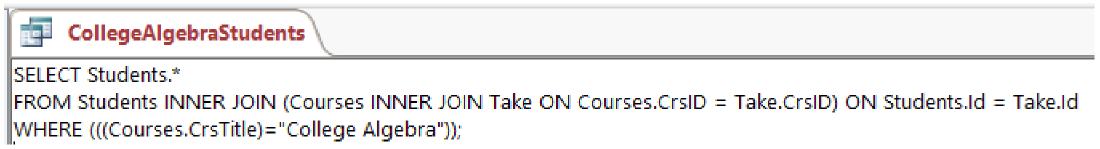
```
select   S.Id, S.Name, S.Class, S.Major
from     Courses C, Take T, Students S
where    C.CrsId = T. CrsId and
         T.Id = S.Id and
         C.CrsTitle = "College Algebra"
```

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:



CollegeAlgebraStudents

```
SELECT Students.*
FROM Students INNER JOIN (Courses INNER JOIN Take ON Courses.CrsID = Take.CrsID) ON Students.Id = Take.Id
WHERE (((Courses.CrsTitle)="College Algebra"));
```

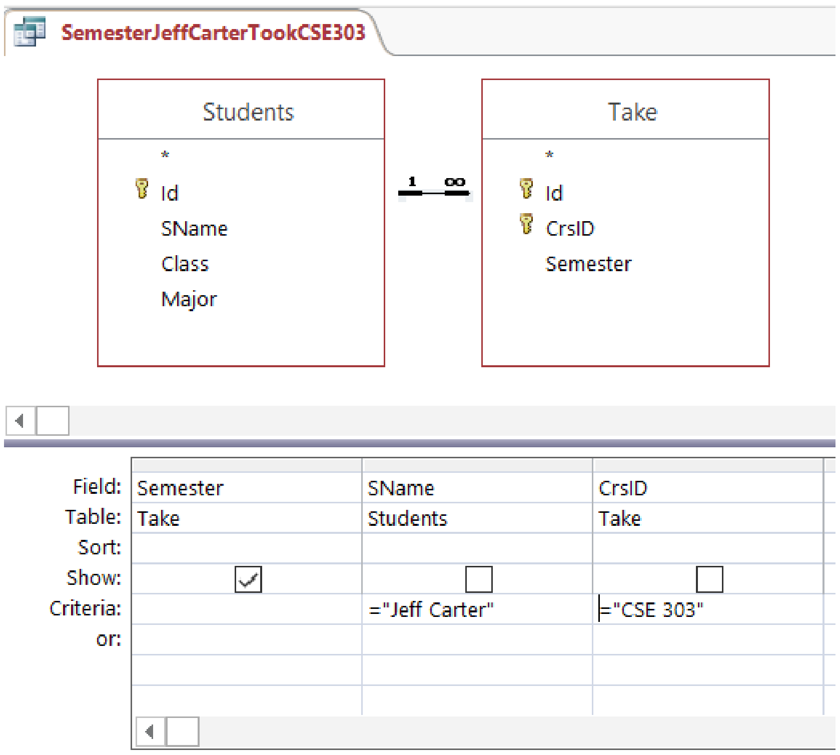
In this query, Access SQL is using the JOIN in the FROM clause. The JOIN is prefixed by the word INNER, which represents the default type of JOIN. Other types of joins are advanced SQL topics and are beyond the coverage of the visualizations. The SQL also shows extra parentheses in the WHERE clause, 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:

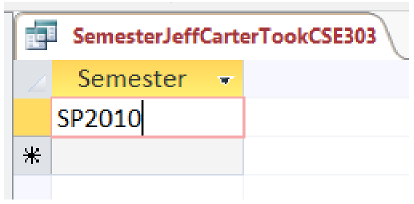


Field:	Semester	SName	CrsID
Table:	Take	Students	Take
Sort:			
Show:	✓	☐	☐
Criteria:		="Jeff Carter"	="CSE 303"
or:			

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:



Semester
SP2010

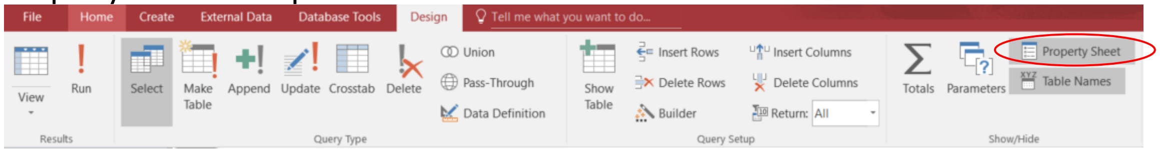
Topic: Sets

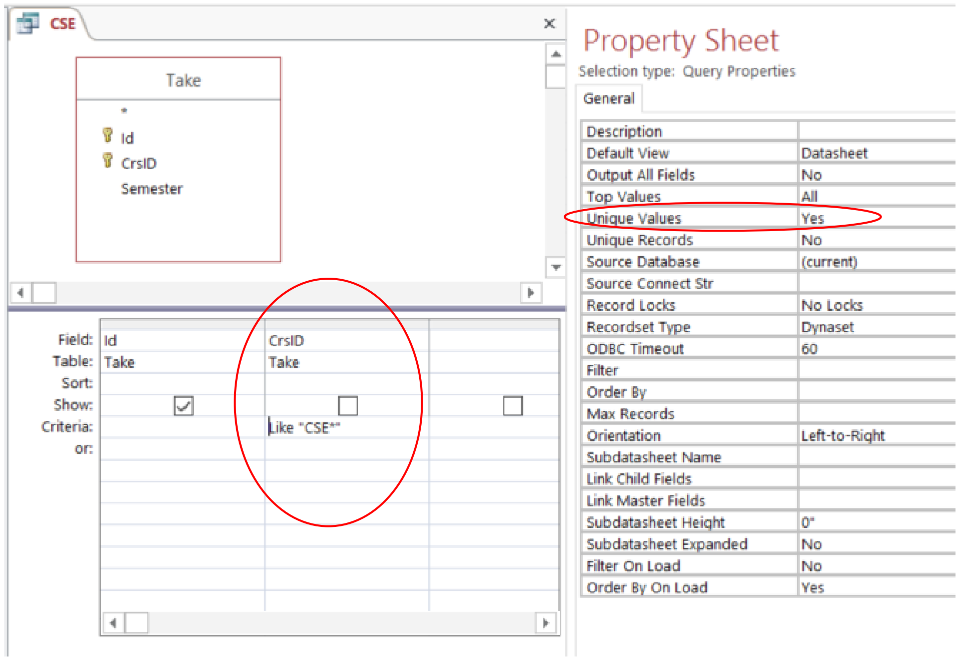
Subtopic: Sets | *Intro*

Query: CSE returns unique Id of Students taking CSE courses

Access:

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





Field:	Id	CrsID
Table:	Take	Take
Sort:		
Show:	☑	☐
Criteria:		Like "CSE*"
or:		

Property Sheet
Selection type: Query Properties
General

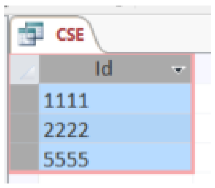
Description	
Default View	Datasheet
Output All Fields	No
Top Values	All
Unique Values	Yes
Unique Records	No
Source Database	(current)
Source Connect Str	
Record Locks	No Locks
Recordset Type	Dynaset
ODBC Timeout	60
Filter	
Order By	
Max Records	
Orientation	Left-to-Right
Subdatasheet Name	
Link Child Fields	
Link Master Fields	
Subdatasheet Height	0"
Subdatasheet Expanded	No
Filter On Load	No
Order By On Load	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.

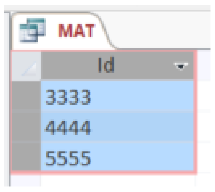
```
create view CSE as
select distinct ID
from Take
where CrsID LIKE "CSE%";
```

Results:

Similarly, for students taking MAT courses.


CSE

Id
1111
2222
5555


MAT

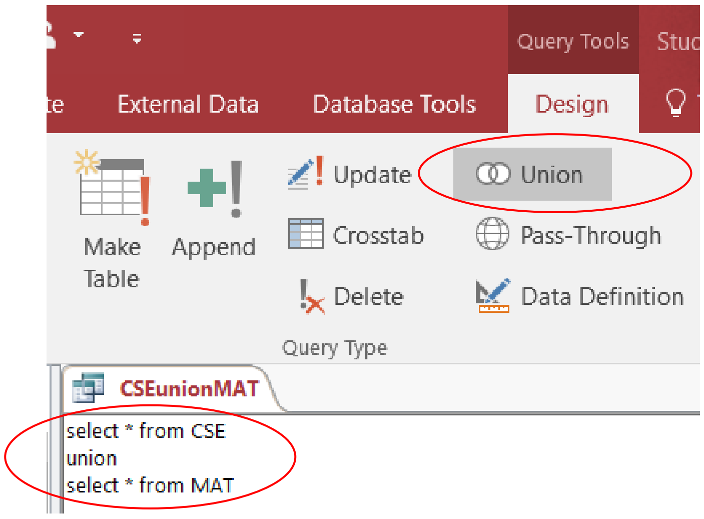
Id
3333
4444
5555

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.

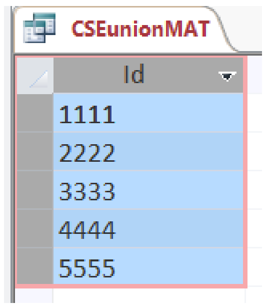


```
select * from CSE
union
select * from MAT
```

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:


CSEunionMAT

Id
1111
2222
3333
4444
5555

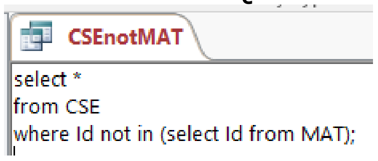
Subtopic: Sets | *Negation*

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.

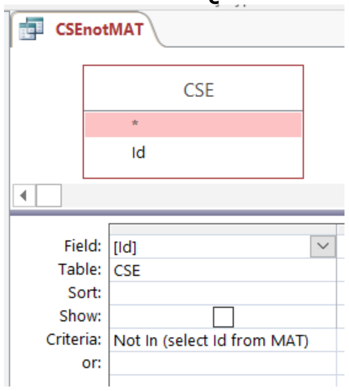
Access SQL



CSEnotMAT

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

Access QBE



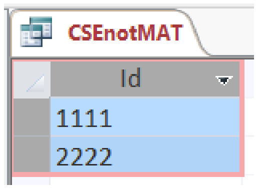
CSEnotMAT

Field: [Id]
Table: CSE
Sort:
Show: ☐
Criteria: Not In (select Id from MAT)
or:

SQL:

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

Result:



CSEnotMAT

Id
1111
2222

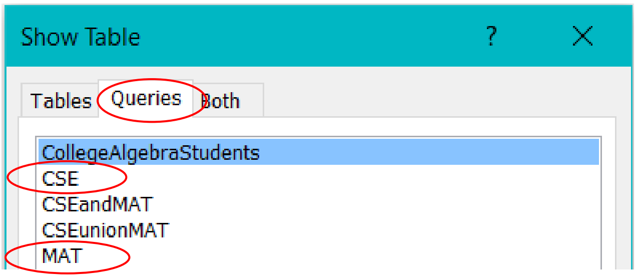
Subtopic: Sets | *Intersection*

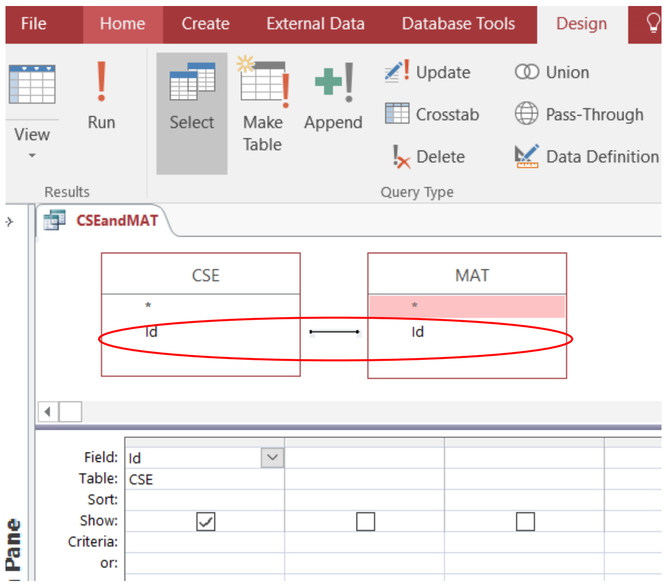
Query: Ids of students who have taken CSE courses and MAT courses

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



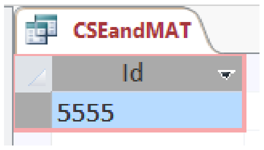


Field:	Id		
Table:	CSE		
Sort:			
Show:	✓	☐	☐
Criteria:			
or:			

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

select * from CSE
intersect
select * from MAT;
	select *
	from CSE natural join MAT;
		select *
		from CSE
		where Id in (select Id from MAT);

Result:



CSEandMAT

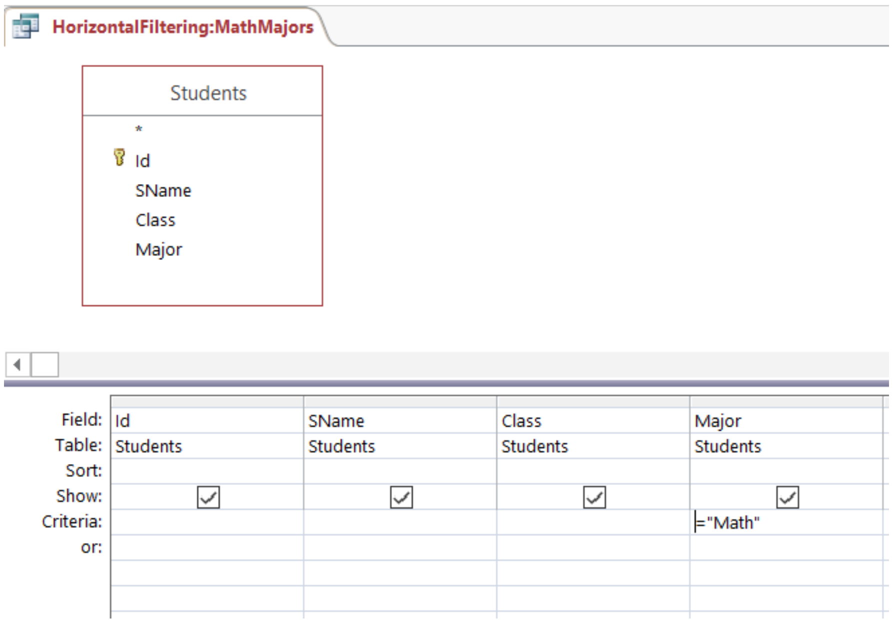
Id
5555

Topic: Filtering

Subtopic: Filtering | *Horizontal*

Query: Find the students who are "Math" majors

Access:

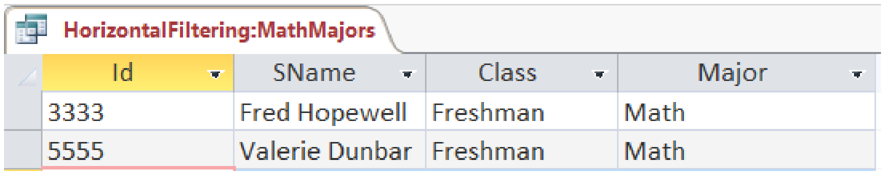


Field:	Id	SName	Class	Major
Table:	Students	Students	Students	Students
Sort:				
Show:	✓	✓	✓	✓
Criteria:				="Math"
or:				

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

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

Result:

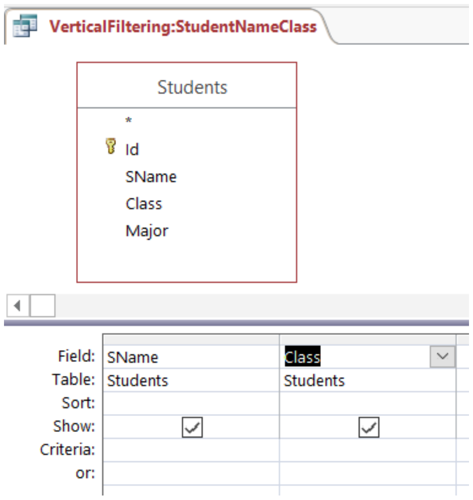


Id	SName	Class	Major
3333	Fred Hopewell	Freshman	Math
5555	Valerie Dunbar	Freshman	Math

Subtopic: Filtering | *Vertical*

Query: Retrieve the Name and Class of all students

Access:



VerticalFiltering:StudentNameClass

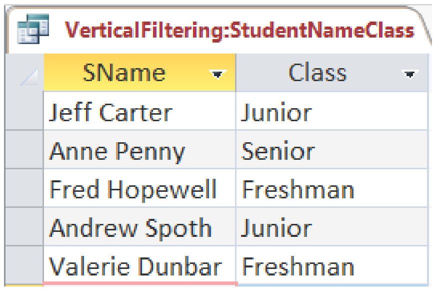
Students: *, Id, SName, Class, Major

Field:	SName	Class
Table:	Students	Students
Sort:		
Show:	✓	✓
Criteria:		
or:		

SQL:

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

Result:



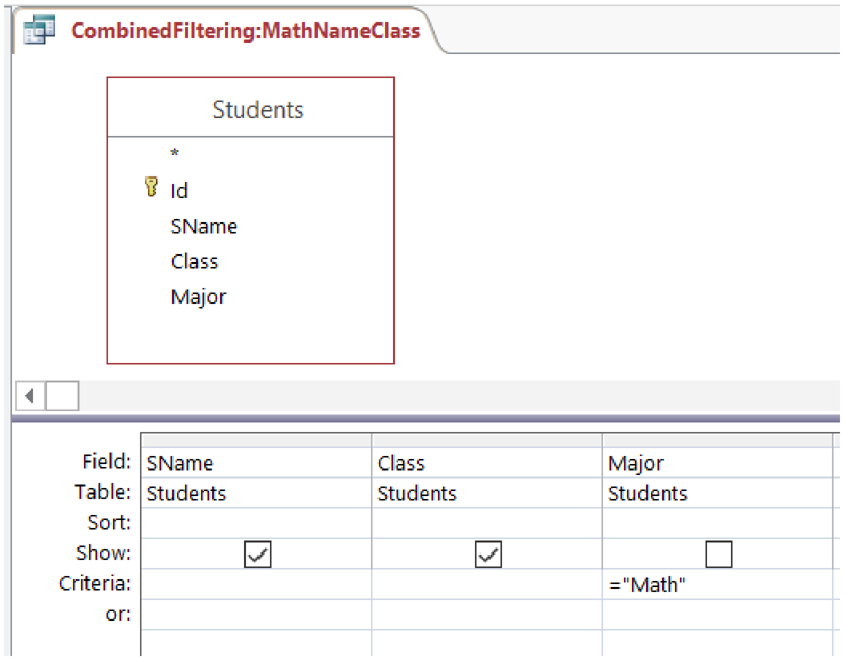
VerticalFiltering:StudentNameClass

SName	Class
Jeff Carter	Junior
Anne Penny	Senior
Fred Hopewell	Freshman
Andrew Spoth	Junior
Valerie Dunbar	Freshman

Subtopic: Filtering | *Combined*

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

Access:



CombinedFiltering:MathNameClass

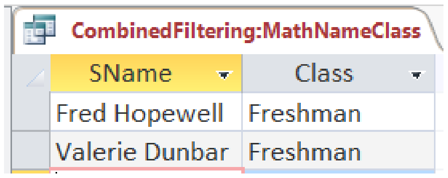
Students: *, Id, SName, Class, Major

Field:	SName	Class	Major
Table:	Students	Students	Students
Sort:			
Show:	✓	✓	☐
Criteria:			="Math"
or:			

SQL:

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

Result:



CombinedFiltering:MathNameClass

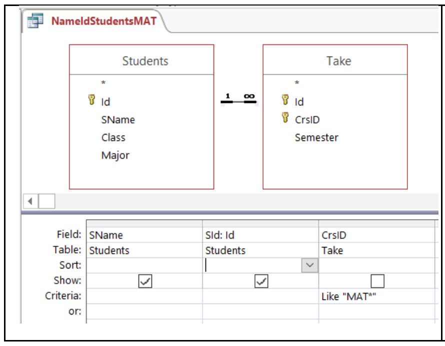
SName	Class
Fred Hopewell	Freshman
Valerie Dunbar	Freshman

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.

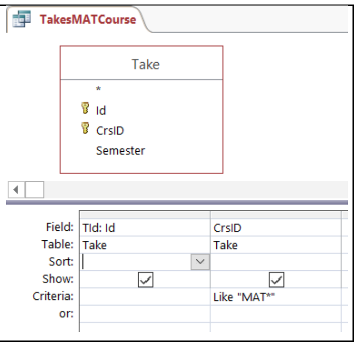
Access:


NameIdStudentsMAT

Students: *, Id, SName, Class, Major

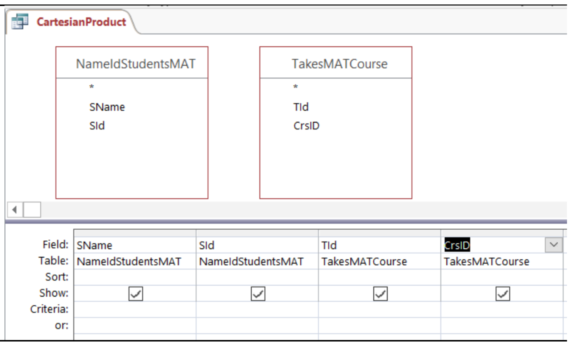
Take: *, Id, CrsID, Semester

Field:	SName	SId: Id	CrsID
Table:	Students	Students	Take
Sort:			
Show:	✓	✓	☐
Criteria:			Like "MAT*"
or:			


TakesMATCourse

Take: *, Id, CrsID, Semester

Field:	TId: Id	CrsID
Table:	Take	Take
Sort:		
Show:	✓	✓
Criteria:		Like "MAT*"
or:		


CartesianProduct

NameIdStudentsMAT: *, SName, SId

TakesMATCourse: *, TId, CrsID

Field:	SName	SId	TId	CrsID
Table:	NameIdStudentsMAT	NameIdStudentsMAT	TakesMATCourse	TakesMATCourse
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				

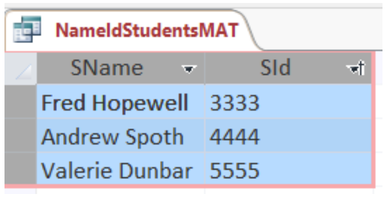
SQL:

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

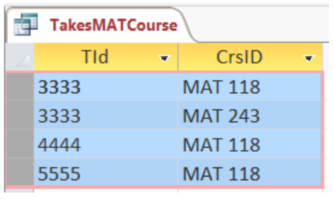
```
create view TakesMATCourse as
select Id as TId, Course
from StudentsTakingCourses
where Course LIKE "MAT%";
```

```
select *
from NameIdStudentsMAT, TakesMATCourse;
```

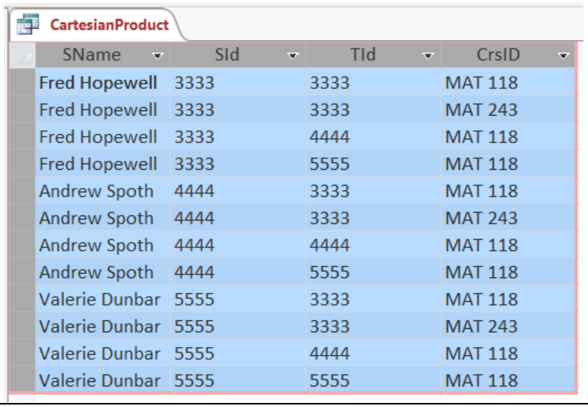
Result:


NameIdStudentsMAT

SName	SId
Fred Hopewell	3333
Andrew Spoth	4444
Valerie Dunbar	5555


TakesMATCourse

TId	CrsID
3333	MAT 118
3333	MAT 243
4444	MAT 118
5555	MAT 118

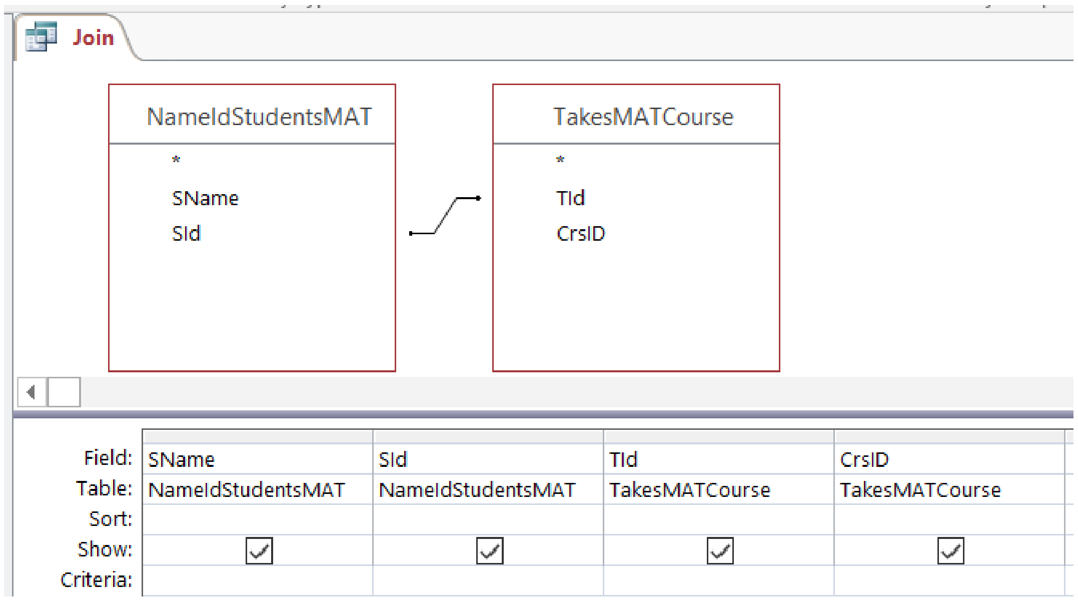

CartesianProduct

SName	SId	TId	CrsID
Fred Hopewell	3333	3333	MAT 118
Fred Hopewell	3333	3333	MAT 243
Fred Hopewell	3333	4444	MAT 118
Fred Hopewell	3333	5555	MAT 118
Andrew Spoth	4444	3333	MAT 118
Andrew Spoth	4444	3333	MAT 243
Andrew Spoth	4444	4444	MAT 118
Andrew Spoth	4444	5555	MAT 118
Valerie Dunbar	5555	3333	MAT 118
Valerie Dunbar	5555	3333	MAT 243
Valerie Dunbar	5555	4444	MAT 118
Valerie Dunbar	5555	5555	MAT 118

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.

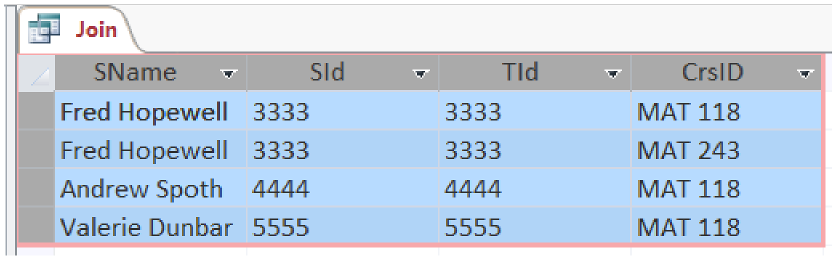


Field:	SName	SId	TId	CrsID
Table:	NameIdStudentsMAT	NameIdStudentsMAT	TakesMATCourse	TakesMATCourse
Sort:				
Show:	✓	✓	✓	✓
Criteria:				

SQL:

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

Result:



SName	SId	TId	CrsID
Fred Hopewell	3333	3333	MAT 118
Fred Hopewell	3333	3333	MAT 243
Andrew Spoth	4444	4444	MAT 118
Valerie Dunbar	5555	5555	MAT 118

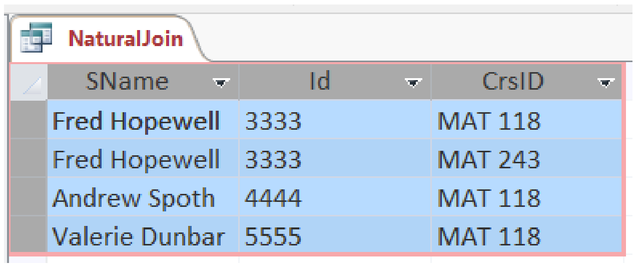
Subtopic: Joining | *NaturalJoin*

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:

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

Result:



SName	Id	CrsID
Fred Hopewell	3333	MAT 118
Fred Hopewell	3333	MAT 243
Andrew Spoth	4444	MAT 118
Valerie Dunbar	5555	MAT 118

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

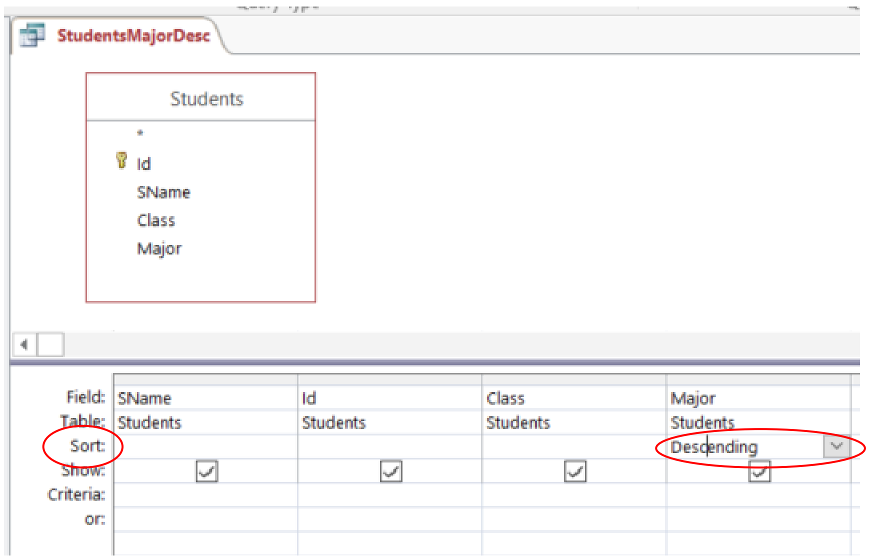
In practice, it is very important to order the results returned by a query for ease of interpretation. In SQL, there is an **order by** clause to order the results of the query. The default ordering of the listed attributes is ascending order when no keyword is specified. To change the ordering to descending,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

SQL:

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

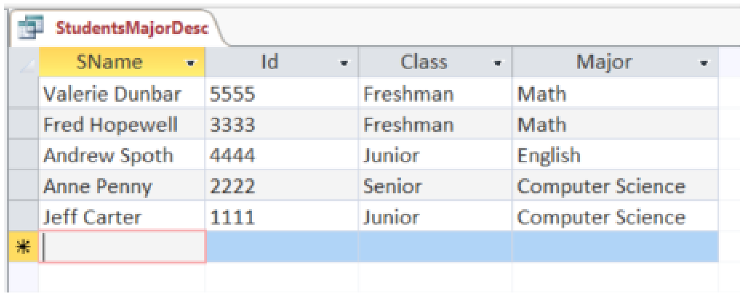
Access:

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



Field:	SName	Id	Class	Major
Table:	Students	Students	Students	Students
Sort:				Descending
Show:	✓	✓	✓	✓
Criteria:				
or:				

Results:



SName	Id	Class	Major
Valerie Dunbar	5555	Freshman	Math
Fred Hopewell	3333	Freshman	Math
Andrew Spoth	4444	Junior	English
Anne Penny	2222	Senior	Computer Science
Jeff Carter	1111	Junior	Computer Science

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:

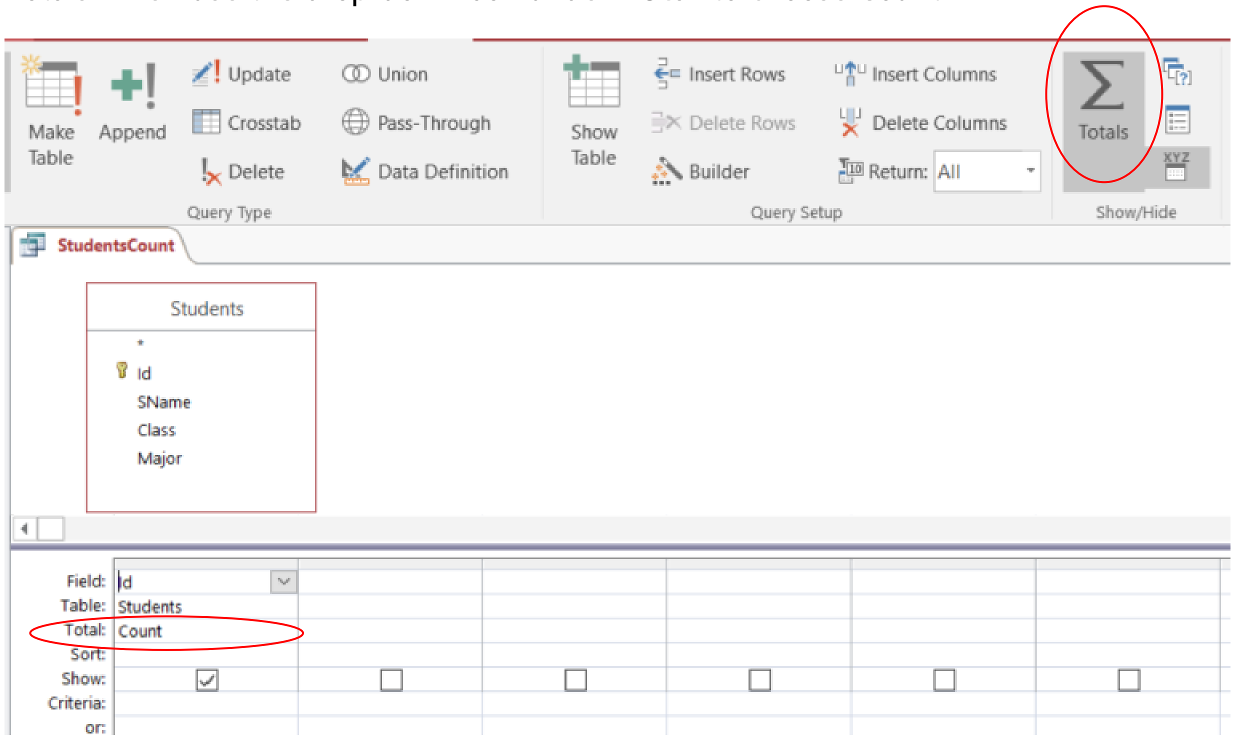
- **count(*)** indicates to count all of the tuples in the table
- **as numberOfTuples** is renaming the column to a meaningful name, since the name of the counting column is not part of the SQL standard and will vary.

SQL:

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

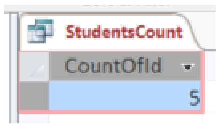
Access:

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*.



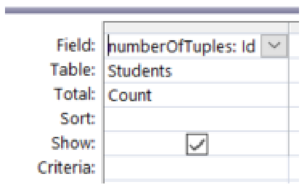
Field:	Id
Table:	Students
Total:	Count
Sort:	
Show:	✓
Criteria:	
or:	

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



StudentsCount

CountOfId
5



Field:	numberOfTuples: Id
Table:	Students
Total:	Count
Sort:	
Show:	✓
Criteria:	

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;
```

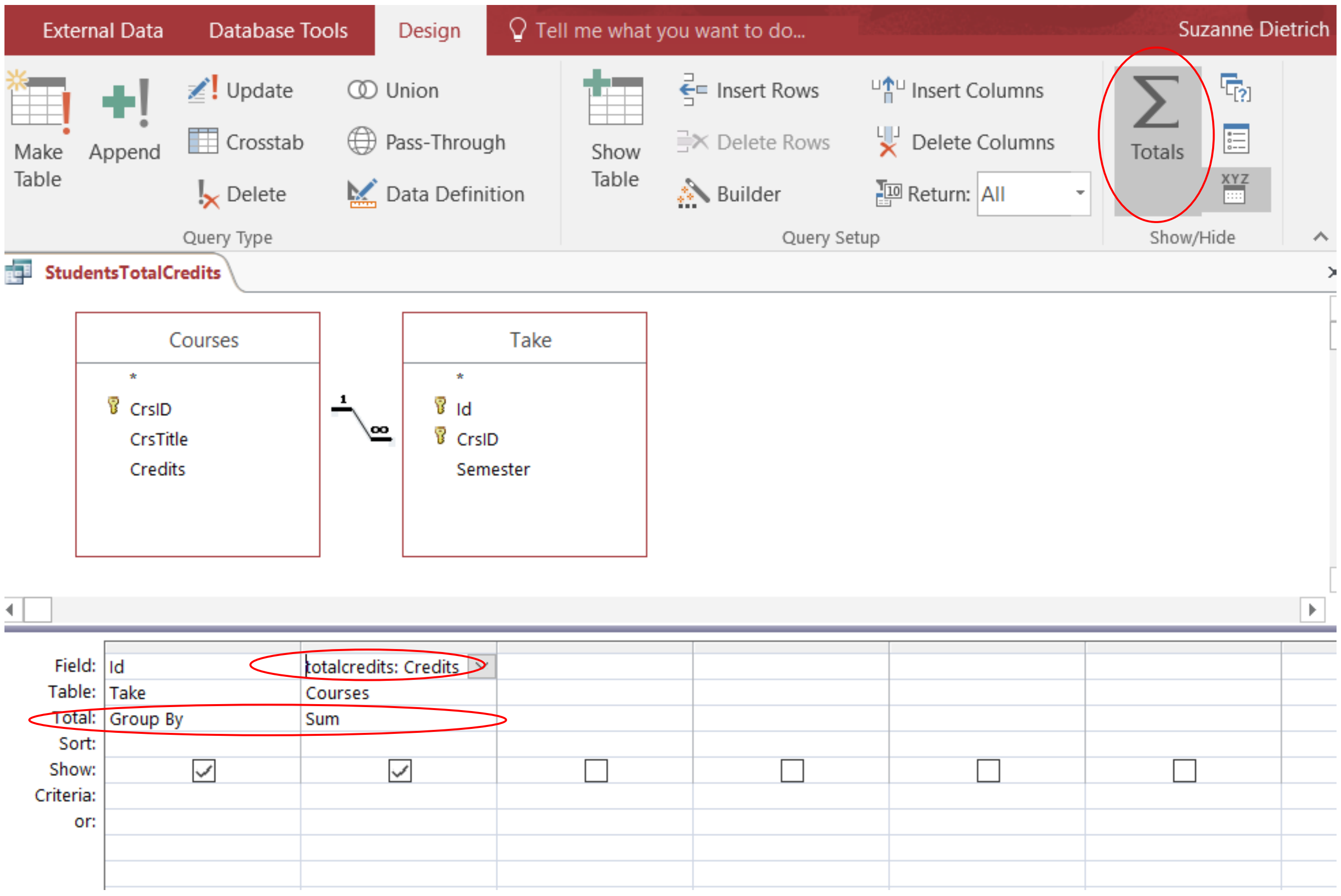
select Id, Credits from Takes natural join Courses order by Id	
Id	Credits
1111	2
1111	3
2222	3
2222	4
3333	3
3333	2
3333	3
4444	4
4444	3
5555	3
5555	2
5555	3

select Id, sum(Credits) as totalcredits from Takes natural join Courses group by Id	
Id	totalcredits
1111	5
2222	7
3333	8
4444	7
5555	8

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**:



Field:	Id	totalcredits: Credits
Table:	Take	Courses
Total:	Group By	Sum
Sort:		
Show:	✓	✓
Criteria:		
or:		

Topic: Checkpoint

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

Terminology	Symbol	Access	SQL
projection; vertical filter	π	√ in “show” box	select
selection; horizontal filter	σ	“criteria”	where
which tables are needed		Add Table	from
Cartesian product	X		commas in “from” clause
(natural) join	⋈	automatic (links shown in relationships diagram)	condition in “where” clause (may need to prefix attribute names!)
union	⋃	“or” in “criteria”	“or” in “where” clause; union
intersection	⋂	“and” in “criteria”	“and” in “where” clause; intersect
negation	-	“and not” in “criteria”	“and not” in “where”; except