SQL

SQL

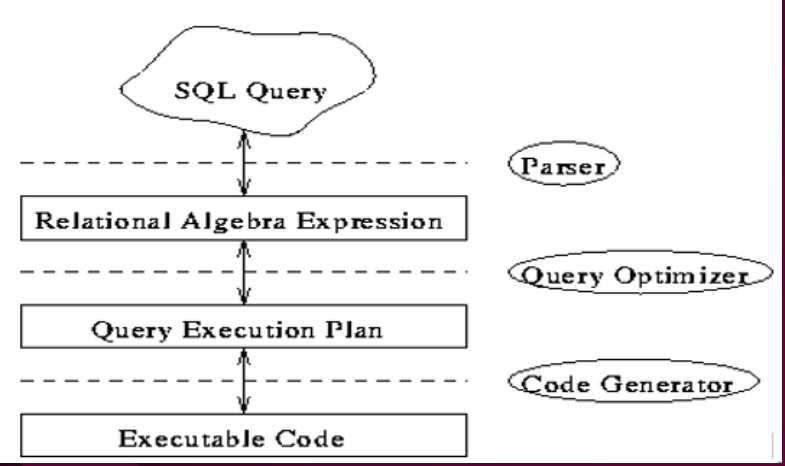
- Categories of SQL commands/functions
  - Data definition language (DDL)
  - Data manipulation language (DML)
  - Transaction control language (TCL)
  - Data control (security) language (DCL)
- Nonprocedural language with basic command vocabulary set of less than 100 words
- Differences in RDBMS Vendors dialects and capabilities

Learning SQL

- SQL requires study plus trial/error
- Understanding relational algebra provides a deeper understanding of SQL
- Going thru this lesson one time and/or reading the corresponding textbook chapter one time will likely not be enough
- You need to practice with a lot of queries to gain a solid understanding of SQL

Relational Algebra

- Relational Algebra is a procedural way to perform database queries
- SQL is non-procedural and based upon the mathematics of relational and predicate calculus



Case Study Tables [salesperson, product, sales]

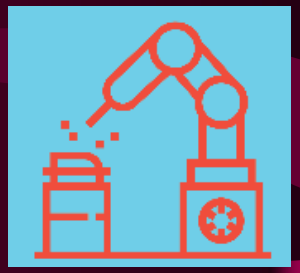
- S (SID, SName, City)
- P (PID, PName, Size, Price)
- SP (*SID, PID*, Qty)

Keys ?

Salesperson Table (S)

SID	Sname	City
S1	Peterson	Aarhus
S2	Olsen	Copenhagen
S4	Hansen	Odense
S5	Jensen	Copenhagen

Product Table (P)

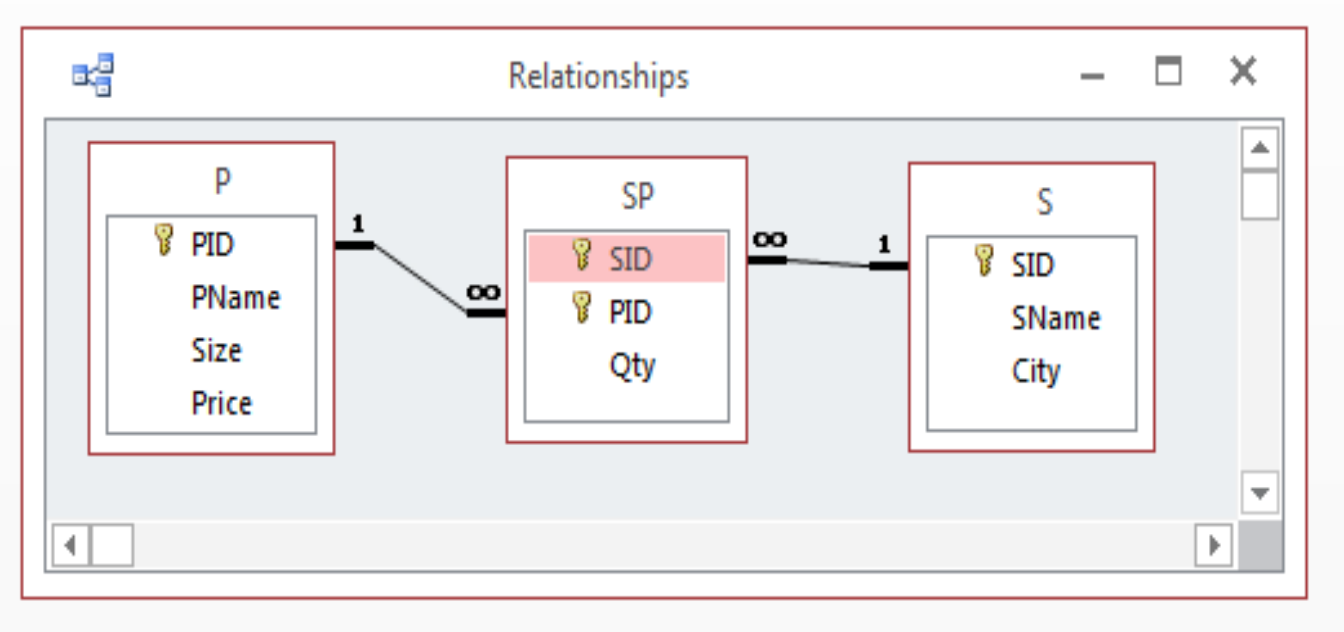


PID	PName	Size	Price
P1	Shirt	6	50
P3	Trousers	5	90
P4	Socks	7	20
P5	Blouse	6	50
P8	Blouse	8	60

SP Table (Intersection Table)

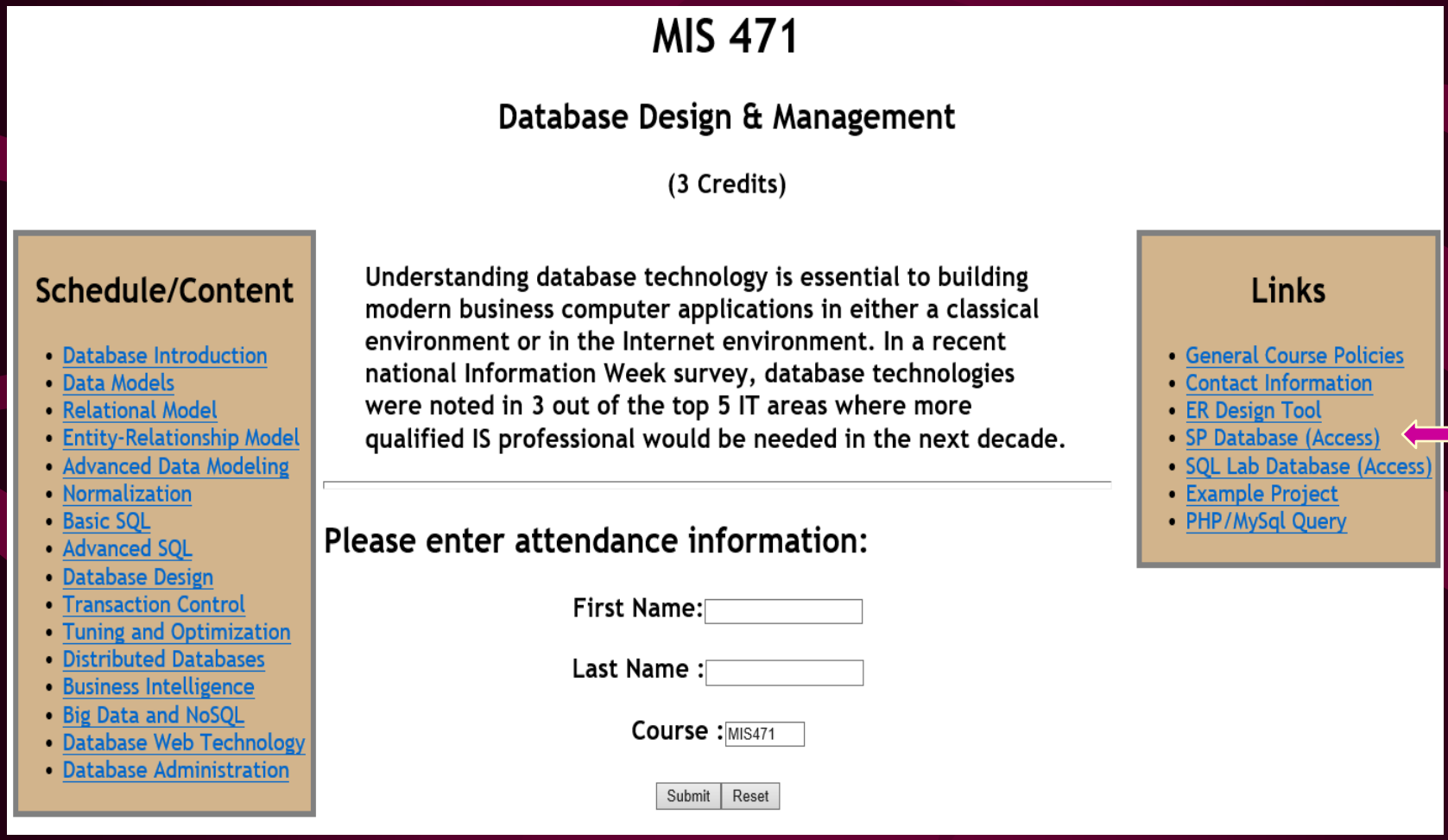
SID	PID	Qty
S2	P1	200
S2	P3	100
S4	P5	200
S4	P8	100
S5	P1	50
S5	P3	500
S5	P4	800
S5	P5	500
S5	P8	100

Access Relationship Grid



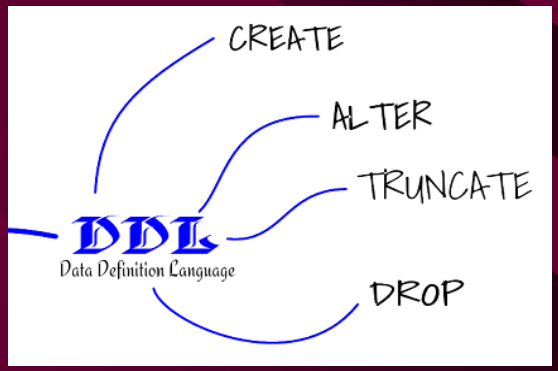
Download this Access file (SP) form the online syllabus.

SP Access File



SQL Data Definition Language

- Create Table
  - Columns (names, allow nulls, and data types)
  - Primary Key
  - Foreign Key
  - Indexes
  - Column Constraints
  - Table Constraints



S (SID, SName, City)

- CREATE TABLE S
  - (SID CHAR(2) NOT NULL,
  - SName VARCHAR (30),
  - City VARCHAR (15),
  - PRIMARY KEY (SID))

S (SID, SName, City)

- Alternative forms
  - CREATE TABLE S
    - (SID CHAR(2) Primary Key,
    - SName VARCHAR (30),
    - City VARCHAR (15))
  - CREATE TABLE S
    - (SID CHAR(2) NOT NULL,
    - SName VARCHAR (30),
    - City VARCHAR (15))
    - ALTER TABLE S ADD PRIMARY KEY (SID))
- Can also add other table features: index, unique index, constraints, foreign key, …

P (PID, PName, Size, Price)

- CREATE TABLE P
  - (PID CHAR(2) NOT NULL,
  - PName VARCHAR (20),
  - Size SMALLINT,
  - Price DECIMAL (5.2) NOT NULL,
  - PRIMARY KEY (PID))

SP (*SID, PID*, Qty)

- CREATE TABLE SP
  - (SID CHAR(2) NOT NULL,
  - PID CHAR(2) NOT NULL,
  - Qty INTEGER,
  - PRIMARY KEY (SID, PID),
  - FOREIGN KEY (SID) REFERENCES S,
  - FOREIGN KEY (PID) REFERENCES P)
- May also be able to add referential integrity actions, discussed later

Basic Data Types

(specific RDBMS may have mode/different types)

- CHARACTER(n) or CHAR (n) - fixed length string of n characters (ie STATE)
- CHARACTER VARYING or VARCHAR(n) - varying length string of up to n characters
- MEMO (text area)
- BIT(n) & VARBIT(n) & Yes/No
- INTEGER, SMALLINT, TINYINT
- FLOAT or REAL & DOUBLE
- DECIMAL(p,q) & NUMERIC (p,q) - assumed decimal point q digits from the right (0 <= q <= p)
- DATE, TIME, and TimeStamp (date/time)
- BLOB (Binary Large Object)

Data Types (con't)

Data Type	Description
BIGINT(size)	Stores an integer value in the range −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, where size specifies the maximum number of digits
BIGINT(size) UNSIGNED	Stores an integer value in the range 0 to 18,446,744,073,709,551,615, where size specifies the maximum number of digits
BLOB	Stores a binary large object up to 65,535 bytes
BOOLEAN	Stores a true or false value
CHAR(size)	Specifies a fixed-size string up to 255 characters, where size specifies the maximum number of characters
DATE()	Stores a date in the form YYYY-MM-DD
DATETIME()	Stores a date and time in the form YYY-MM-DD HH:MM:SS
DECIMAL(size, fractional_digits)	Stores a DOUBLE as a string, where size specifies the number of digits and fractional_digits specifies the number of digits to the right of the decimal point
DOUBLE(size, fractional_digits)	Stores a double-precision floating-point number, where size specifies the number of digits and fractional_digits specifies the number of digits to the right of the decimal point
ENUM(a, b, c)	Stores a set of up to 65,535 enumerated values
FLOAT(size, fractional_digits)	Stores a floating-point number, where size specifies the number of digits and fractional_digits specifies the number of digits to the right of the decimal point

Data Types (con't)

INT(size)	Stores an integer value in the range −2,147,483,648 to 2,147,483,647, where size specifies the maximum number of digits
INT(size) UNSIGNED	Stores an integer value in the range 0 to 4,294,967,295, where size specifies the maximum number of digits
LONGBLOB	Stores binary large object up to 4,294,967,295 bytes
LONGTEXT	Stores a string of up to 4,294,967,295 characters
MEDIUMBLOB	Stores a binary large object up to 16,777,215 bytes
MEDIUMTEXT	Stores a string of up to 16,277,215 characters
SET(a, b, c)	Stores a set of up to 64 enumerated values
SMALLINT(size)	Stores an integer value in the range −32,768 to 32,767, where size specifies the maximum number of digits
SMALLINT(size) UNSIGNED	Stores an integer value in the range 0 to 65,535, where size specifies the maximum number of digits
TEXT	Stores a string of up to 65,535 characters
TINYTEXT	Stores a string of up to 255 characters
TINYINT(size)	Stores an integer value in the range −128 to 127, where size specifies the maximum number of digits
TINYINT(size) UNSIGNED	Stores an integer value in the range 0 to 255, where size specifies the maximum number of digits
VARCHAR(size)	Stores a variable-length string of up to 255 characters, where size specifies the maximum number of characters

NULL

- NULL values in tables, means *unknown* value
- NULL table results, means empty table
- Installation defined implementation
- Area of incompatibility between RDBMS
- Operations: IS NULL or IS NOT NULL
- Ignored in functions (ave, max, ...)
- Arithmetic. If A or B or BOTH are NULL then all these expressions evaluate to NULL: A+B, A-B, A*B, A/B
- NULLS are equal to each other
- Logic. If A or B or BOTH are NULL then all these expressions use the *unknown* truth table (next slide): A=B, A NOT = B, A>B, A<B, A>=B, A<=B; Example: “A<>3” is not true for NULL A

Truth Table with Unknowns

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT *p*
True	False
False	True
Unknown	Unknown

Retrieving Data via SQL

- SELECT columns-in-output-tables
  - FROM input-tables
  - WHERE logical-expression
  - ORDER BY columns -in-output-tables
- Output is always a table
  - may be a table with only one row and/or column (“singleton”)
  - may be a NULL table

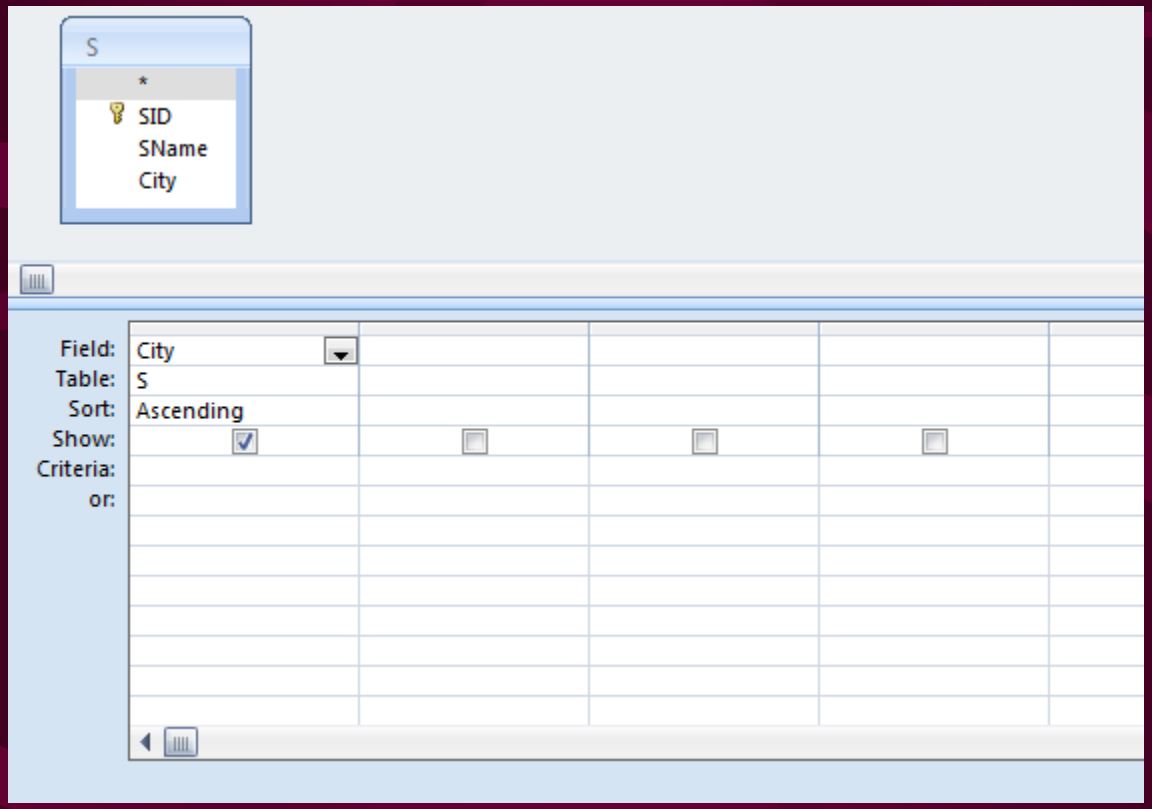
Access "Query by Example"

- Query-By-Example (QBE) is also non-procedural
- There is no standard for QBE
- Not all queries can be done in QBE
- Perform an Access QBE to answer this question (save as "Q1"):
  - "In which cities are salespersons located ?"

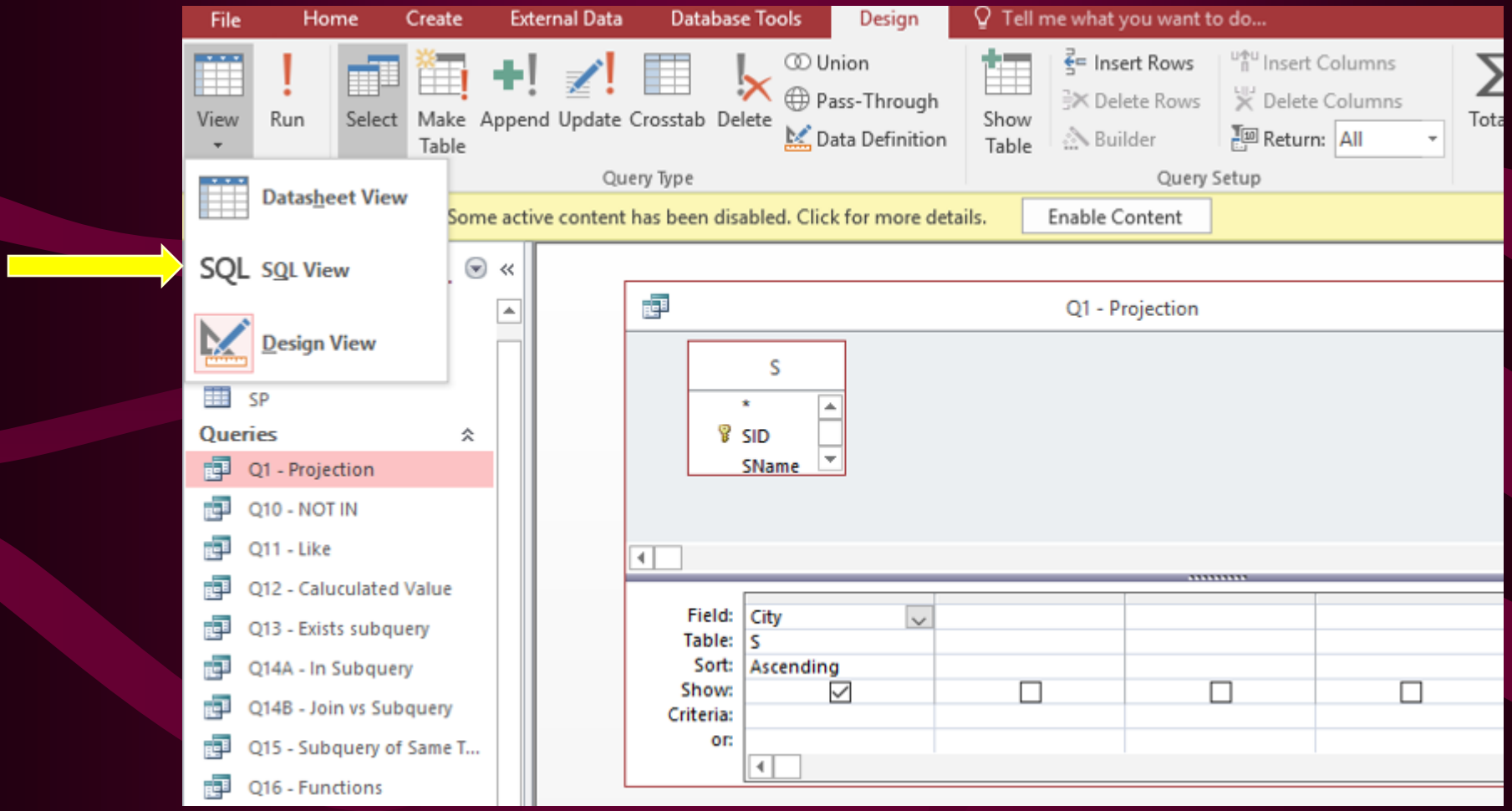
Don’t look ahead !

Christian Brothers University

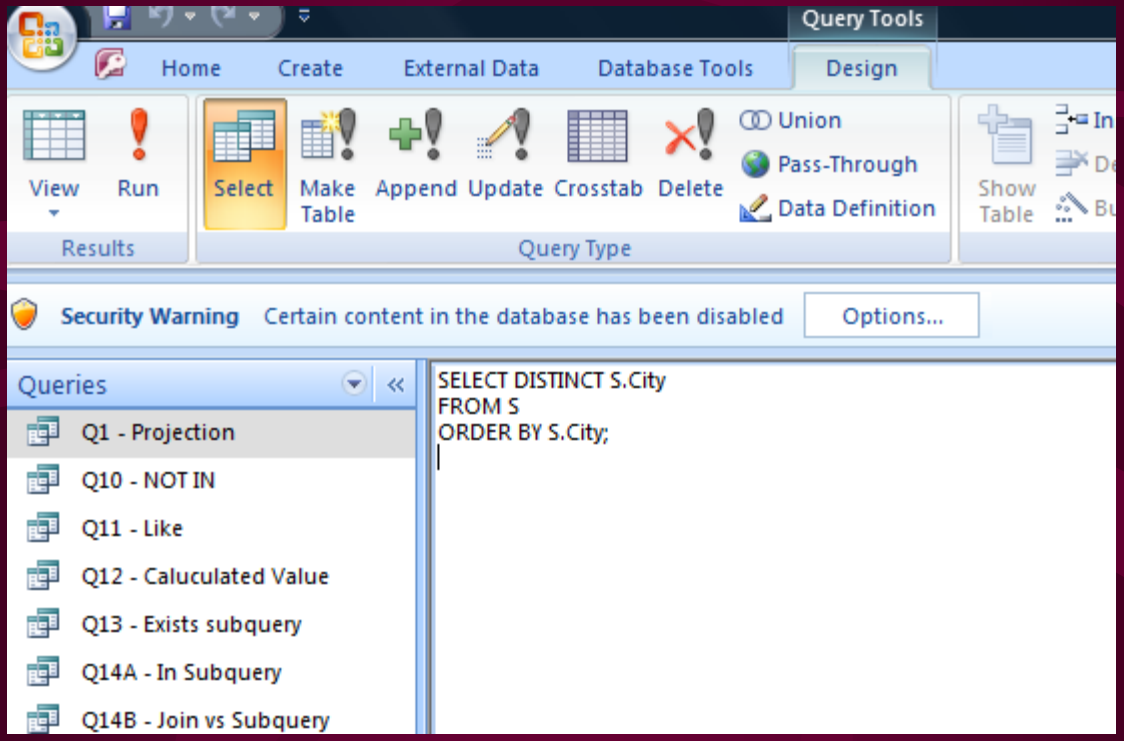
Access Query Grid
[Query by Example]



Access SQL View



Access SQL View



Relational Algebra PROJECTION
("project" certain columns)

- SELECT DISTINCT City
  - –FROM S
  - –ORDER BY City
- *DISTINCT removes redundant columns**
- "In which cities are salespersons located ?"
- * In Access, select "View" then "Properties" to set query properties; select "unique values" to "yes"

Q1

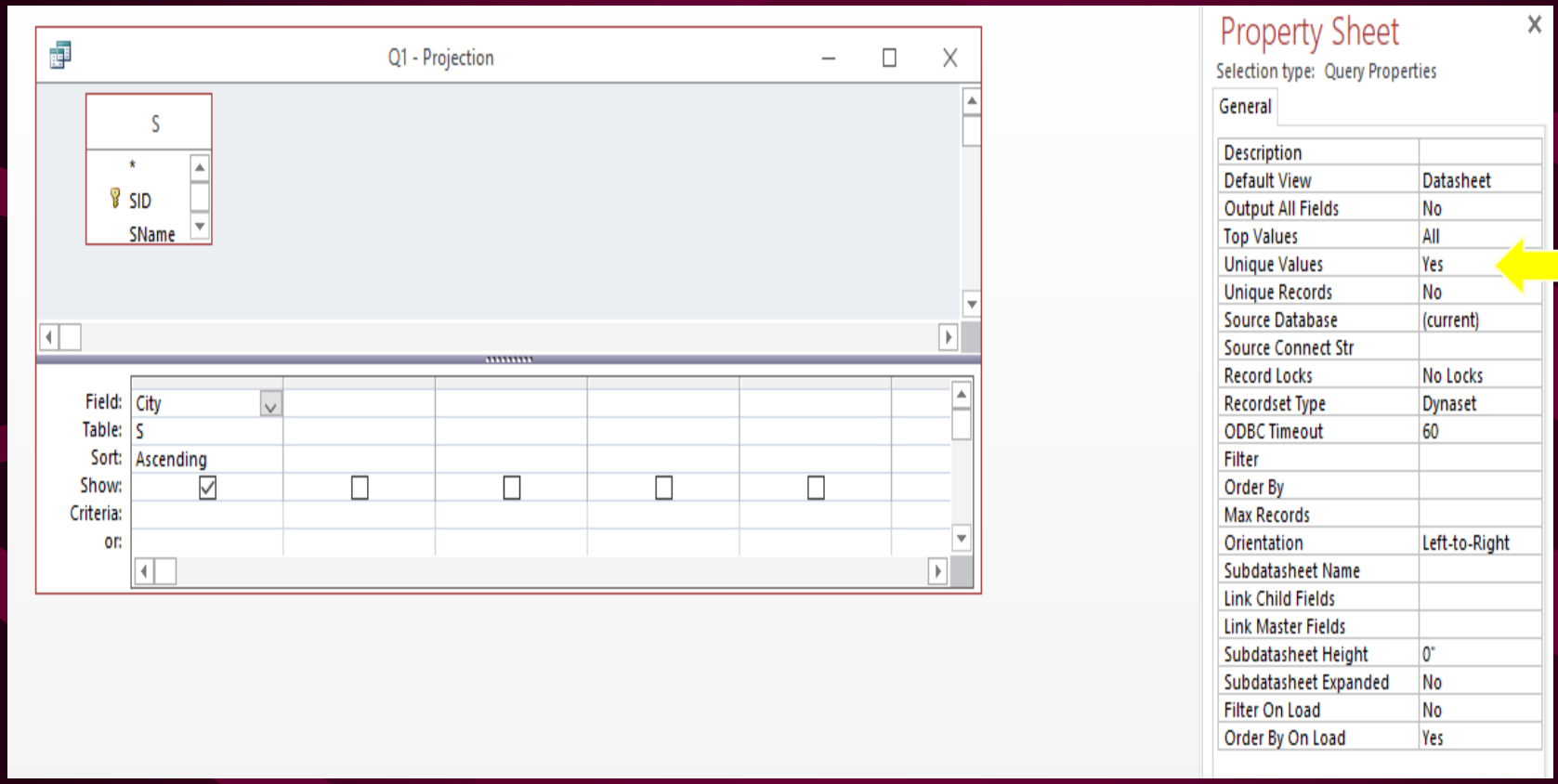
City

Aarhus

Copenhagen

Odense

Q1 (con't)



Distinct & Distinctrow

- In Access:
  - DISTINCT - Shows rows if selected columns are unique
  - DISTINCTROW - Shows rows if entire row from underlying table(s) are unique

Access Exercise

- Perform an Access query via the query grid to answer this question:
  - "List all info for salespersons in Copenhagen"
- What is the SQL for this query ?

Don’t look ahead !

Christian Brothers University

Relational Algebra SELECTION (WHERE) ("select" certain rows)

- SELECT *
  - FROM S
  - WHERE City = 'Copenhagen'
  - ORDER BY SName DESC
- "List info for salespersons in Copenhagen"
- * selects all columns
- DESC sorts in descending order

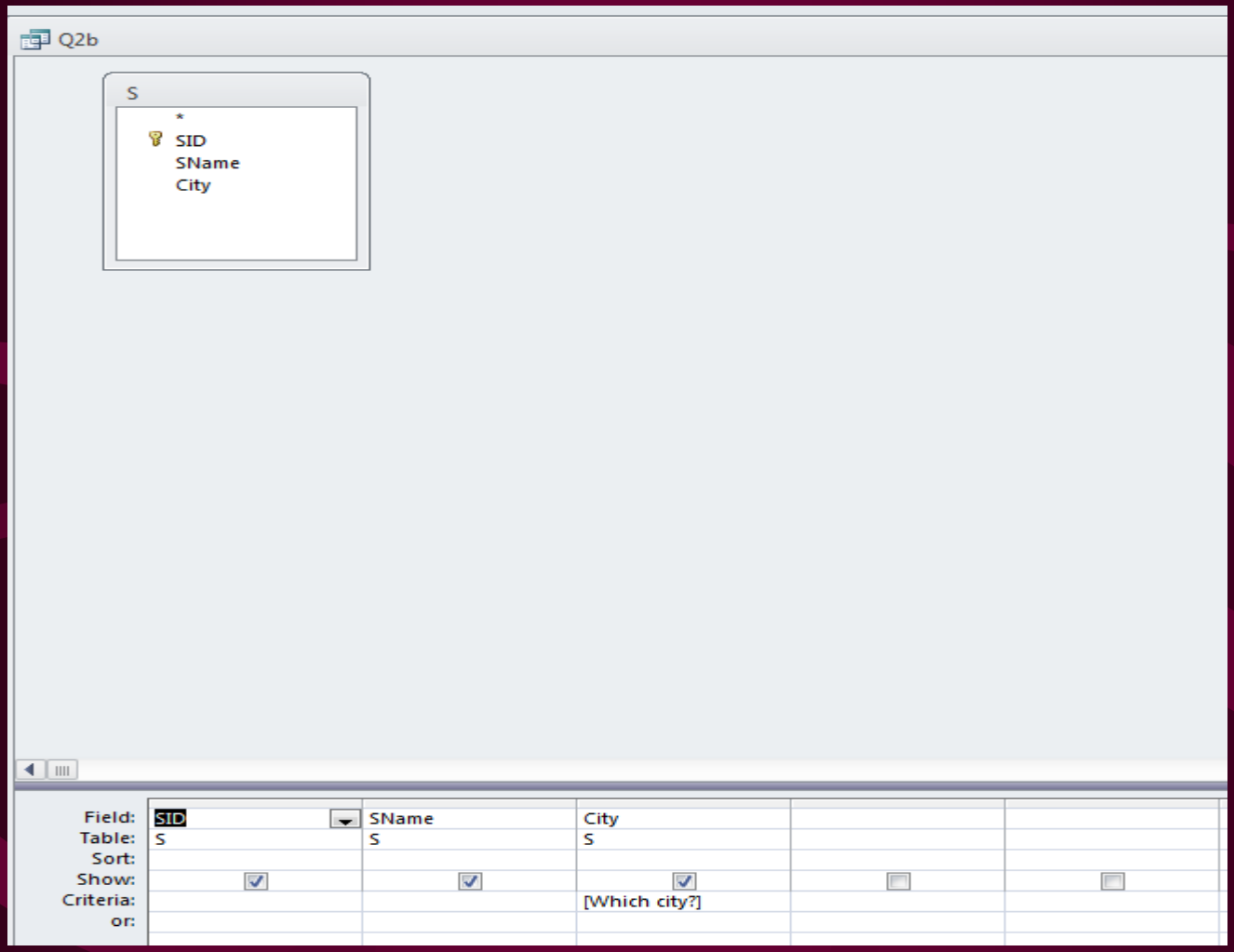
Q2 [Q2a, SQL design alternative (without checking any “show” boxes, “show all” in properties)]

SID	Sname	City
S2	Olsen	Copenhagen
S5	Jensen	Copenhagen

Parameter Query

- Which salespersons live in _____ ?
- Standard SQL:
  - SELECT S.SID, S.SName, S.City
  - FROM S
  - WHERE (S.City=?);

Parameter Query in Access (Q2b)



Access Exercise

- Perform an Access query via the query grid to answer this question:
  - Which salespersons (SID's) are either in Copenhagen or have sold some P8 ?
- What is the SQL for this query ?

Don’t look ahead !

Christian Brothers University

UNION [Q3, via Join]

[Q3a, SQL Union, no Access QBE design] (members in either set)

- Which salespersons (SID's) are either in Copenhagen or have sold some P8 ?
- SELECT SID
  - FROM S
  - WHERE City = 'Copenhagen'
  - UNION
  - SELECT SID
  - FROM SP WHERE PID = 'P8'
- The result is the table with the column containing S2, S4, S5 (the UNION removes duplicate rows, there is a UNION ALL to retain any duplicates)

- Access, like many RDBMS, cannot do Unions, Intersection, and Differences via their GUI QBE
- In fact, Access (and many others) cannot do Intersection or Difference at all
- For Query Optimization (necessary for Enterprise Applications) needs to be done in SQL not QBE

Access Exercise

- Perform an Access query via the query grid to answer this question:
  - Which salespersons (SID's) are both in Copenhagen and have sold some P8
- What is the SQL for this query ?

Don’t look ahead !

Christian Brothers University

INTERSECTION [Q4]

[no Access Intersection, need to use join]

(members in both sets)

- Which salespersons (SID's) are both in Copenhagen and have sold some P8
- SELECT SID
  - FROM S
  - WHERE City = 'Copenhagen'
  - INTERSECT
  - SELECT SID
  - FROM SP
  - WHERE PID = 'P8'

SQL Join

- The "join process" (inner join) involves multiplying two (or more) tables together and then removing the rows that do not meet a "join criteria"
- Multiplying two tables together involves taking all possible combinations of the rows from the first table with the rows from the second table

Product of Two Tables

Student
Smith
Jones

Courses
Mgt282
Mgt284
Mgt287

Create A Table With All Possible Combinations of Students and Courses

Possible Enrollments	
Smith	Mgt282
Smith	Mgt284
Smith	Mgt287
Jones	Mgt282
Jones	Mgt284
Jones	Mgt287

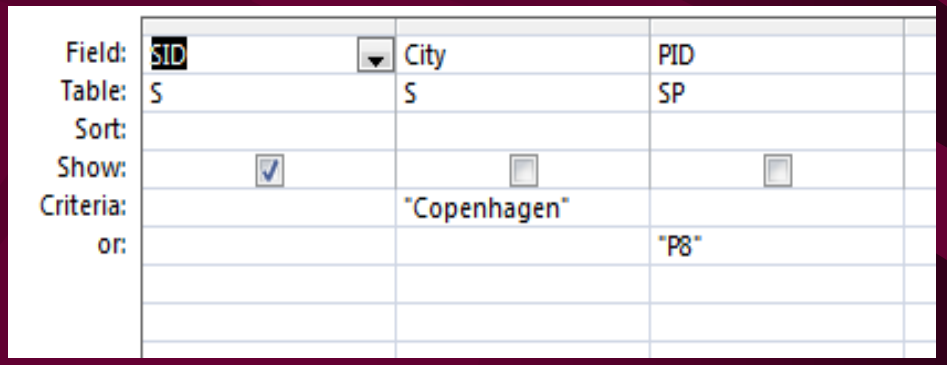
Inner Join

- SELECT DISTINCT S.SID
- FROM S INNER JOIN SP ON S.SID = SP.SID

- The join criteria is “S.SID = SP.SID” which is the typical join criteria (matching the primary key of the first table with the corresponding foreign key in the second table)

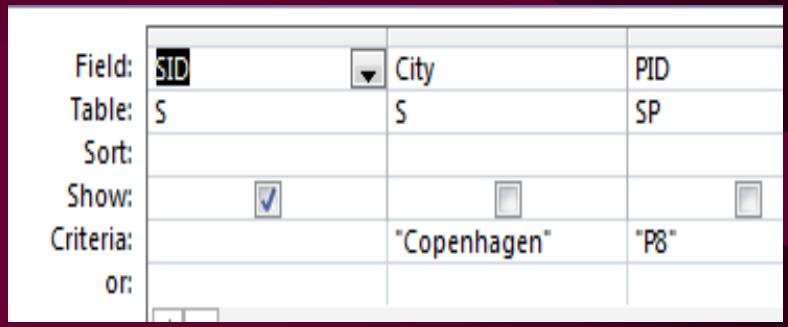
Union via Join

- SELECT DISTINCT S.SID
- FROM S INNER JOIN SP ON S.SID = SP.SID
- WHERE S.City = ‘Copenhagen’ OR SP.PID = ‘P8’



Intersection via Join

- SELECT DISTINCT S.SID
- FROM S INNER JOIN SP ON S.SID = SP.SID
- WHERE S.City = ‘Copenhagen’ AND SP.PID = ‘P8’;



Access Exercise

SID	Sname	City
S1	Peterson	Aarhus
S2	Olsen	Copenhagen
S4	Hansen	Odense
S5	Jensen	Copenhagen

SID	PID	Qty
S2	P1	200
S2	P3	100
S4	P5	200
S4	P8	100
S5	P1	50
S5	P3	500
S5	P4	800
S5	P5	500
S5	P8	100

- Perform an Access query via the query grid to answer this question:
  - Display the salespersons (SID's) in Copenhagen who have not sold any P8
- What is the SQL for this query ?

Don’t look ahead !

Christian Brothers University

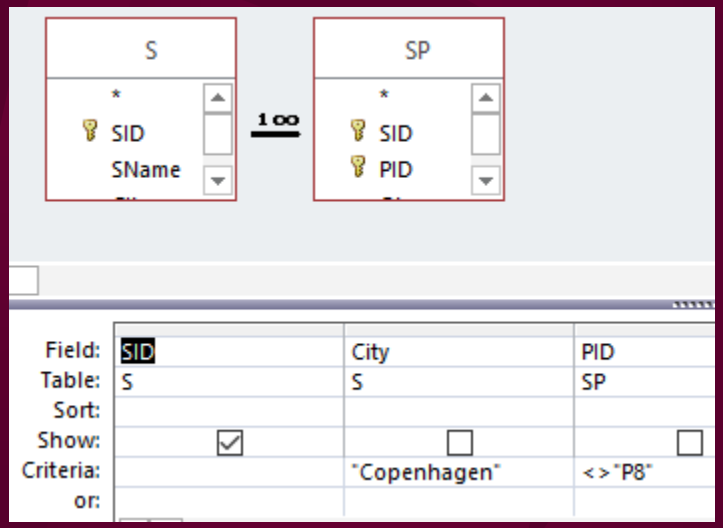
DIFFERENCE [Q5, no Access difference] (*EXCEPT* or *MINUS*, varies by vendor)

- Display the salespersons (SID's) in Copenhagen who have not sold any P8
- SELECT SID
  - FROM S
  - WHERE City = 'Copenhagen'
  - EXCEPT
  - SELECT SID
  - FROM SP
  - WHERE PID = 'P8'

Need to use sub queries in Access !

What's wrong with this; why does this not produce a difference ?

- SELECT DISTINCT S.SID
- FROM S INNER JOIN SP ON S.SID = SP.SID
- WHERE S.City='Copenhagen' AND SP.PID not = "P8"; use "<>" in Access



T = T1 - T2 = [S2]

- T1 (in Copenhagen)
  - S2
  - S5
- T2 (sold P8)
  - S4
  - S5

The join will yield 14 rows (two for S [Copenhagen] and seven for SP [not P8]), and the projection of SID will yield two rows for S2 and S5, because S5 also has a row(s) in SP that are for products other that P8 ! But only S2 is the answer .

SQL/92 "Standard" **Join**

- TABLE_REF NATURAL {JOIN_TYPE} JOIN TABLE_REF
  - Where:
    - TABLE_REF is the name (or alias) of a table
    - {JOIN_TYPE} (INNER is the default):
      - INNER
      - LEFT {OUTER}
      - RIGHT {OUTER}
      - FULL {OUTER}

If the join condition is the equality between the columns in common, the join is called an equijoin.

If one of the two common columns in an equijoin is eliminated, then it is called a natural join (the most common kind of join which removes the redundancy)

OR

- TABLE_REF {JOIN_TYPE} JOIN TABLE_REF ON EXP
  - Where:
    - TABLE_REF is the name (or alias) of a table
    - EXP is an expression
    - {JOIN_TYPE} is (INNER is the default):
      - INNER
      - LEFT {OUTER}
      - RIGHT {OUTER}
      - FULL {OUTER}
- Alternative form (“legacy SQL”):
  - TABLE_REF, TABLE_REF WHERE EXP

OR

- TABLE_REF {JOIN_TYPE} JOIN TABLE_REF USING COLS
  - Where:
    - TABLE_REF is the name (or alias) of a table
    - COLS is a column list (same names in both tables)
    - {JOIN_TYPE} is (INNER is the default):
      - INNER
      - LEFT {OUTER}
      - RIGHT {OUTER}
      - FULL {OUTER}

Acceptable Forms of the Join

- SELECT SName, Qty
  - FROM S NATURAL JOIN SP [PK and FK match]
- SELECT SName, Qty
  - FROM S JOIN SP ON S.SID = SP.SID
  - Or "legacy form":
    - SELECT SName, QTY
      - FROM S, SP
        - » WHERE S.SID = SP.SID
- SELECT SName, Qty
  - FROM S JOIN SP USING (SID) [called 'SID' in both tables]

SQL JOIN Support

- Some (currently most) products only support “legacy form”
- Some products (Access) do not support (or partially support) legacy form
- Some products support both (to some degree)
- Inner joins cannot be nested within outer joins, but outer joins can be nested within inner joins

JOIN - “Show names of salespersons with product sold info”

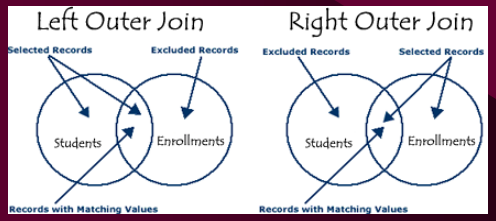
- SELECT SName, S.SID, PID, Qty
  - FROM S, SP
  - WHERE S.SID = SP.SID
- Common column is SID; if they have the same name then specify the table:
  - TableName.ColumnName
- Access SQL syntax:
  - FROM S INNER JOIN SP ON S.SID=SP.SID

Q6

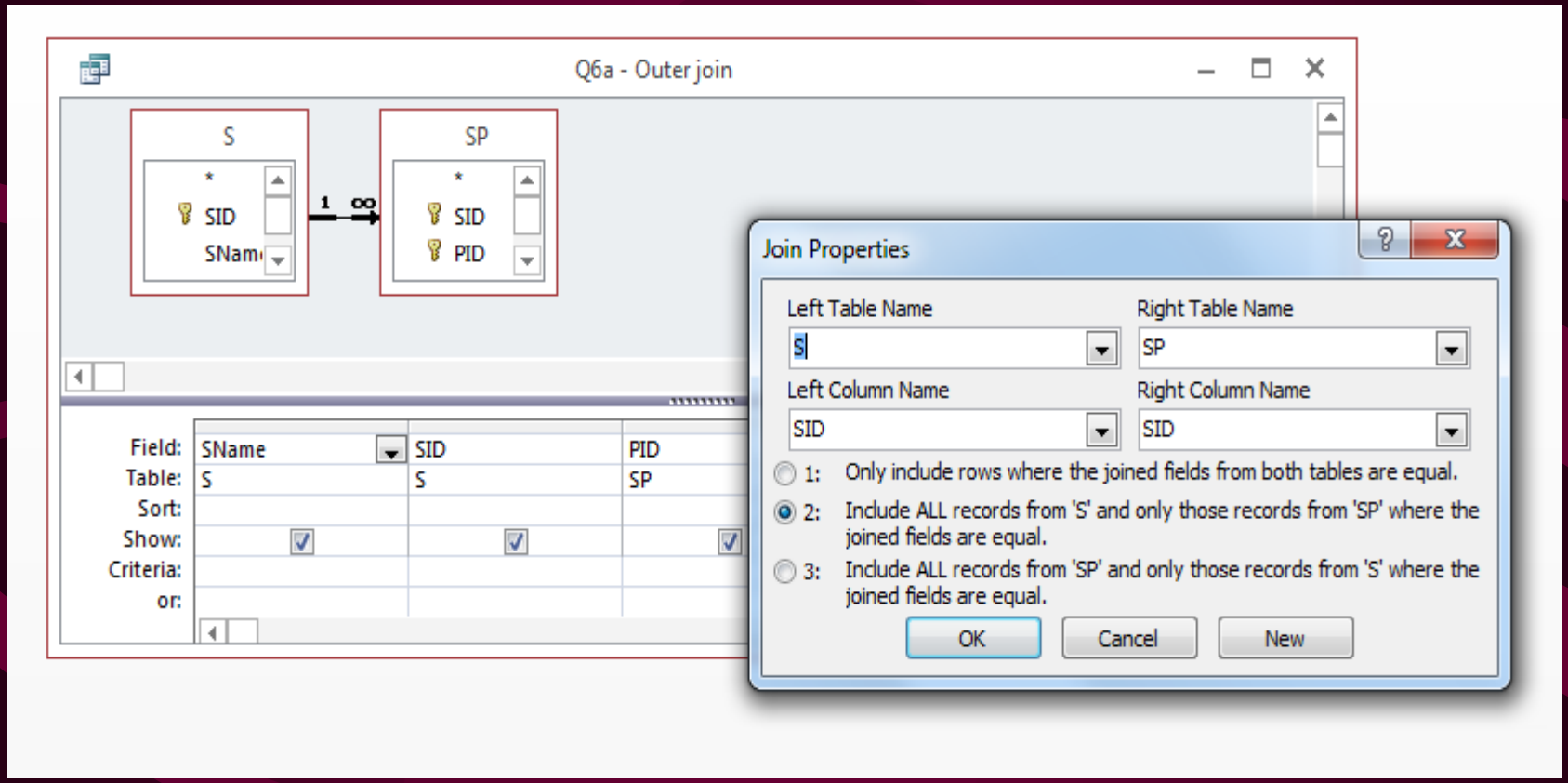
SName	SID	PID	Qty
Olsen	S2	P1	200
Olsen	S2	P3	100
Hansen	S4	P5	200
Hansen	S4	P8	100
Jensen	S5	P1	50
Jensen	S5	P3	500
Jensen	S5	P4	800
Jensen	S5	P5	500
Jensen	S5	P8	100

Include info for all salespersons

- SELECT SName, S.SID, PID, Qty
  - FROM S LEFT OUTER JOIN SP
  - WHERE S.SID = SP.SID
- OR:
  - SELECT SName, S.SID, PID, Qty
  - FROM S LEFT OUTER JOIN SP
  - ON S.SID = SP.SID
- Q6a



Access Outer Join in QBE
[right click join arrow]



Other Outer Join SQL Syntax

- In Transact-SQL (SQLServer):
  - SELECT SName, S.SID, PID, Qty
  - FROM S LEFT OUTER JOIN SP
  - WHERE S.SID *= SP.SID
- In PL/SQL (Oracle):
  - SELECT SName, S.SID, PID, Qty
  - FROM S LEFT OUTER JOIN SP
  - WHERE S.SID = SP.SID+

SQL Query Approach
[thus far]

- 1. Understand problem
- 2. What tables are involved ?
- 3. If more than one table, then what operation is need between the tables:
  - Join (inner or outer), Union, Intersection, Difference
- 4. What are the selection criterion ?
- 5. What columns need to be projected ?
- 6. Any sorting ?

Class Exercise

- Write the SQL to display the product names (Pname) and quantities (Qty) of products sold for SID ‘s
- Check your answer in Access

Don’t look ahead !

Christian Brothers University

Q7

- SELECT SID, PName, Qty
  - FROM P, SP
  - WHERE P.PID = SP.PID

Include Salesperson names:

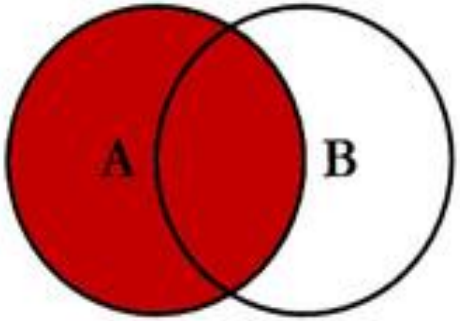
- SELECT SID, SName, PName, Qty
  - FROM P, SP, S
  - WHERE P.PID = SP.PID
  - AND SP.SID = S.SID

- When N tables are joined, N-1 join criterion are needed !

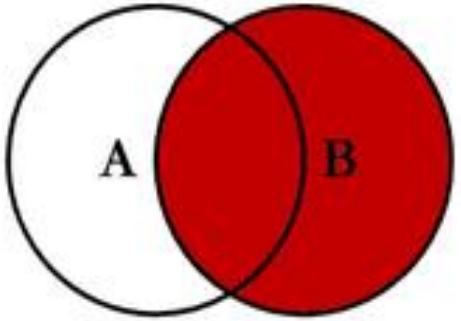
Cross Join

- A “cross join” (full outer join) is the same as the Cartesian Product in relational algebra (the combination of all rows of the first table with all rows of the second table):
  - SELECT SID, PID
  - FROM S, P

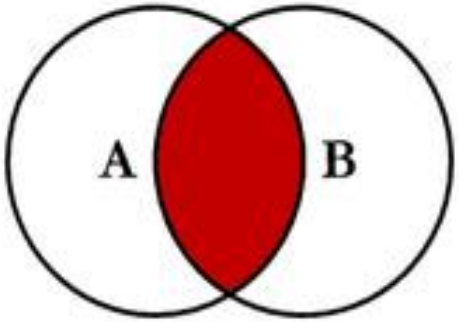
SQL JOINS



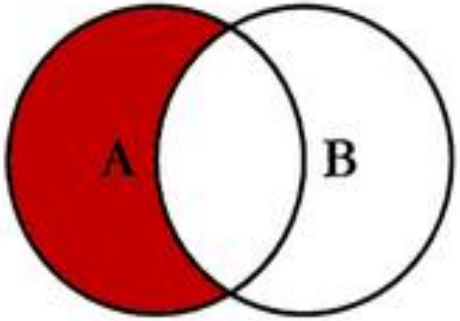

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



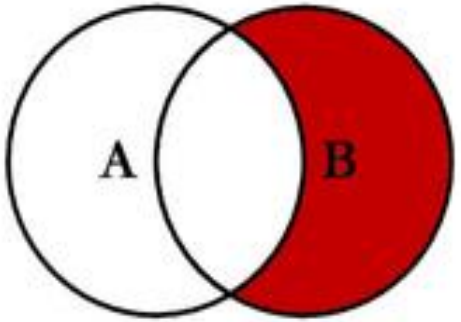

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



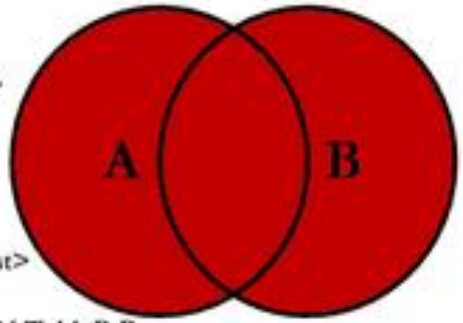

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



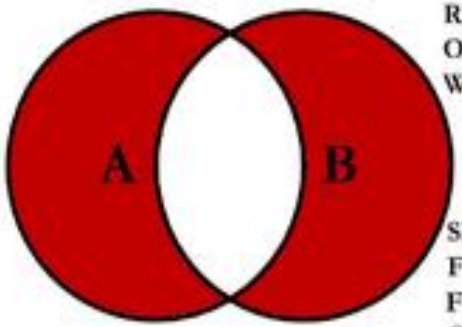

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Self Joins

- Tables may be joined with themselves
- Need to give the table(s) *alias* names
- Display a list of products that can be supplied in a size that is two sizes larger (product names have to be the same)
- What tables ae involved?
- How would you do this manually ??
  - See next slide…

Product Table (P)

PID	PName	Size	Price
P1	Shirt	6	50
P3	Trousers	5	90
P4	Socks	7	20
P5	Blouse	6	50
P8	Blouse	8	60

Display a list of products that can be supplied in a size that is two sizes larger…

- Two ways:
  - Multiply the tables together to find all combinations of one product matched with another product; then strike out all the rows that do not match the join criteria
  - For each row
    - Look at all the other rows that match the criteria (subqueries - discussed later)

SQL Self Join

- SELECT X.PID
  - FROM P X, P Y (P as X, P as Y)
  - WHERE (X.PName = Y.PName)
    AND (X.Size + 2 = Y.SIZE)
- The answer is the single value P5
  [Q8]

Self Join Applications

- Self Joins are also common in auditing queries:
  - Find missing (skipped) control numbers (ie invoice numbers)
  - Find duplicating fields (ie duplicate payments, etc.)

SQL Logical Operators

Operator	Description	Negation (not)
=	Equal to	Yes
<	Less than	Yes
<=	Less than or equal to	No
>	Greater than	Yes
>=	Greater than or equal to	No
OR	logical expression is true if at least one of the arguments of OR is a true expression	No
AND	logical expression is true if both argumnts are true expressions	No

Can also use <> or != instead of “not =“

SQL Logical Operators

Operator	Description	Negation
BETWEEN	Expression is true if the operand lies within a spepcified interval, including the limits	Yes
IN	Expression is true if the operand is included in a stated table with **one column**; this operator corresponds to the set operator 'is a member of'	Yes
EXISTS	Expression is true if there are rows in the table serving as the argument; corresponds to the set existential operator	Yes
LIKE	Expression is true if a character column contains certain selected combinations of columns	Yes

Display product numbers of blouses in the price range of $30 to $60: [Q9]

- SELECT PID
  - FROM P
  - WHERE (PName = ‘Blouse’) AND (Price BETWEEN 30 AND 60)

- The output is the table with two rows: P5 and P8
- Output would include both $30 and $60 blouses
- Note that Blouse is in quotes and 30 (or 60) is not

Quotes

- Standard SQL uses the single quote for literal text strings
- Access uses the double quote for text strings
- Some products can use either
- For standard SQL, only use the double quotes for the AS clause with keywords as titles:
  - Select Name as “select” from …
- With ODBC (or JDBC) and even if you are using ODBC for an Access File, use the single quote
- For literal text with single quotes in them (Al’s Hardware), you can do this as: ‘Al’’s Hardware’ or some products allow use of “escape characters” (‘Al^{esc}’s Hardware’)
  - Be careful with “magic quotes” in some products
  - **Magic quotes** was a feature of the PHP scripting language, wherein strings are automatically escaped—special characters are prefixed with a backslash—before being passed on. It was introduced to help newcomers write functioning SQL commands without requiring manual escaping

Display all information for salespersons who are not in Copenhagen or Odense: [Q10]

- SELECT *
  - FROM S
  - WHERE City NOT IN ('Copenhagen', 'Odense')
- Note that "IN" essentially requires a one column table to look thru

Display salesperson ID's and names for those whose name starts with a 'J' and ends with 'SEN' and the first character of their 2 character ID is 'S': [Q11]

- SELECT *
  - FROM S
  - WHERE (SName LIKE 'J%SEN')
  - AND (SID LIKE 'S_')
- % means zero, one or more characters
- _ means exactly one character
- Wildcard symbols vary with product

ANSI Wildcard Characters

Character	Description	Example
%	Matches any number of characters. It can be used as the first or last character in the character string.	**wh%** finds what, white, and why, but not awhile or watch.
_	Matches any single alphabetic character.	**B_ll** finds ball, bell, and bill.
[ ]	Matches any single character within the brackets.	**B[ae]ll** finds ball and bell, but not bill.
^	Matches any character not in the brackets.	**b[^ae]ll** finds bill and bull, but not ball or bell.
-	Matches any one of a range of characters. You must specify the range in ascending order (A to Z, not Z to A).	**b[a-c]d** finds bad, bbd, and bcd.

Access Wildcard Characters

Character	Description	Example
*	Matches any number of characters. You can use the asterisk (*) anywhere in a character string.	**wh*** finds what, white, and why, but not awhile or watch.
?	Matches any single alphabetic character.	**B?ll** finds ball, bell, and bill.
[ ]	Matches any single character within the brackets.	**B[ae]ll** finds ball and bell, but not bill.
!	Matches any character not in the brackets.	**b[!ae]ll** finds bill and bull, but not ball or bell.
-	Matches any one of a range of characters. You must specify the range in ascending order (A to Z, not Z to A).	**b[a-c]d** finds bad, bbd, and bcd.
#	Matches any single numeric character.	**1#3S** finds 103, 113, and 123.

Further Use of Expressions

- A column in the output table may consists of an arithmetic expression with the operators +, -, *, /, ^
- It is also possible to sort the output table on such a calculated column
- The logical expression after WHERE may also contain arithmetic operators

Display a price list in which the products are sorted descending according to the price of the product including a 22% sales tax: [Q12]

- SELECT PID, PName, Price * 1.22
  - FROM P
  - ORDER BY 3 DESC, PID ASC

- Note the use of the ‘3’ to refer to the third output column in the ORDER BY clause
- Could also give ‘Price*1.22’ a name with AS clause

SQL Arithmetic Functions

Function	Operation
ABS	Returns a value's absolute value
ACOS	Returns a value's arc cosine
ASIN	Returns a value's arc sine
ATAN	Returns a value's arc tangent
ATAN2	Returns the arc tangent of two values
CEIL	Returns the smallest integer value that is greater than or equal to the specified number
CEILING	Returns the smallest integer value that is greater than or equal to the specified number
COS	Returns a value's cosine
COT	Returns a value's cotangent
DEGREES	Returns the degree equivalent of a value in radians
DIV	Returns the integer result of a division operation (not the remainder, as does MOD)
EXP	Returns the value of *e* raised to the power of the specified number
FLOOR	Returns the largest integer value that is less than or equal to a value
GREATEST	Returns the greatest value in a list of values
LEAST	Returns the smallest value in a list of values
LN	Returns a value's natural logarithm
LOG	Returns a value's natural logarithm or the value's logarithm to a specified base

SQL Arithmetic Functions (con't)

LOG10	Returns a value's natural logarithm to base 10
LOG2	Returns a value's natural logarithm to base 2
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
MOD	Returns the remainder (modulo) of a value divided by another number
PI	Returns the value of pi
POW	Returns the value of a number raised to the power of the specified number
POWER	Returns the value raised to the power of the specified number
RADIANS	Returns the radian equivalent of a value specified in degrees
RAND	Returns a random number
ROUND	Returns the value rounded to the specified number of decimal places
SIGN	Returns a value's sign
SIN	Returns a value's sine
SQRT	Returns a value's square root
SUM	Returns the sum of a set of values
TAN	Returns a value's tangent
TRUNCATE	Returns a value truncated to the specified number of decimal places

Subqueries

- Queries can be nested, and connected by SQL logical operators
- SELECT *
  - FROM ...
  - WHERE operator
    - (subquery)

Subqueries (con't)

- Subqueries are enclosed in parathesis
- Subqueries can also involve the same tables, and if so *alias* names have to be used (just like in the self join)
- Think of subqueries like nested "for loops" in a program; the inner query may be using variables from the outer loop

Display product number and name of products sold: [Q13]

- SELECT PID, PName
  - FROM P
  - WHERE EXISTS
  - (SELECT *
    - FROM SP
    - WHERE SP.PID = P.PID)
- Remember that EXIST is true if there are any rows in the argument table

Information Processing Methods

- Joins are like combining all the data together and then sorting thru what you need !
- Subqueries are like the process you would normally go thru manually to find information in tables

Information Processing Methods (con't)

- One method can be much faster than the other depending on the database product and the problem at hand – try both !
- One cannot always use joins in the place of subqueries and vice-versa !

Display product number and name of products sold - Manually

- Look at each product in the product table (look at P table in Access)
- For each product in that table, look over in the SP table and see if it occurs in that table also
- If so, then it has been sold (by somebody)

Find the names of salespersons who have sold something: [Q14 a&b]

- SELECT SName
  - FROM S
  - WHERE SID IN [note "IN" needs a one column table]
    - (SELECT DISTINCT SID
    - FROM SP)
- *or with a join*
- SELECT DISTINCT SName
  - FROM S, SP
  - WHERE S.SID = SP.SID
- Some queries can be expressed with joins or subqueries, which is more efficient depends upon the size of the tables, and indexes involved

Exists and IN Methods

- SELECT PID, PName
  - FROM P
  - WHERE EXISTS
  - (SELECT *
    - FROM SP
    - WHERE SP.PID = P.PID)
- SELECT PID, PName
  - FROM P
  - WHERE PID IN
  - (SELECT DISTINCT PID [need a table with one column]
    - FROM SP)
- Note that the IN version is (should be)faster here since, the subquery produces the same table for each row of the outer loop; but this depends upon the optimization capability of the data base product used !

Any or All Conditions [Q15]

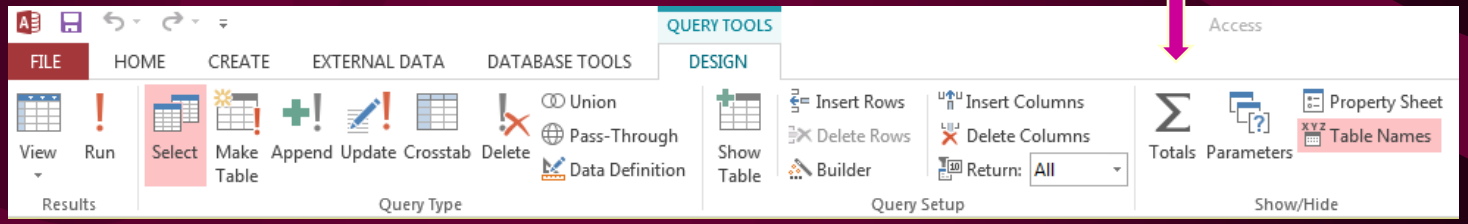
- Get product names for products whose price is greater than the price of every blouse
- SELECT DISTINCT A.PName
  - FROM P A [need alias]
  - WHERE A.Price > ALL
  - (SELECT DISTINCT B.Price
    - FROM P B
    - WHERE B.PName = ‘Blouse’)
- Can also use ANY

SQL Aggregate Functions

- SQL also has built-in functions (aggregate functions) which can be used as output:
  - SUM - sums a column
  - MIN - finds minimum of column
  - MAX - finds maximum of column
  - AVG - calculates average value for a column
  - COUNT (*) - counts rows
  - COUNT (DISTINCT column-name) - counts the number of different values in a column

SQL Aggregate Functions (con't)

- NULL values not included in calculations
- In Access, need to click on the aggregate total button (the button as a "sum" sign on it) to get a "total" line on query grid



Display the number of salespersons who have been selling product P3, along with statistics:

- SELECT COUNT (*), SUM (Qty), MAX (Qty), MIN (Qty), AVG(Qty)
  - FROM SP
  - WHERE PID = ‘P3’
- The output is the single row:
  - 2, 600, 500, 100, 300

GROUPS (subtotals)

- The SELECT statement can be altered so that it works with groups of rows
- The parameter GROUP BY causes the selected rows to be grouped together thereby outputting a table consisting of statistical information about each group
- The additional parameter HAVING is used if the output table is to have a subset of the total number of groups

GROUPS (con't)

- Remember:
  - –WHERE qualifies individual rows !
  - –HAVING qualifies groupings (subtotals) !

Display a list of cities with more than one salesperson (with a count of salespersons in that city): [Q17]

- SELECT City, COUNT (*)
  - FROM S
  - GROUP BY City
  - HAVING COUNT (*) > 1
  - ORDER BY City
- The output is the single row:
  - Copenhagen, 2

Class Exercise - Part 1

- Write the SQL for showing the total of units that have been sold ?
- Try in Access also (click on sum button)

Don’t look ahead !

Christian Brothers University

Q18a

- SELECT SUM(SP.Qty) AS SumOfQty
- FROM SP;

Class Exercise - Part 2

- Now what are the total sales in dollars ? [in SQL]
- What tables do you need ?

Don’t look ahead !

Christian Brothers University

Q18

- SELECT SUM (SP.Qty * P.Price)
  - FROM SP, P
  - WHERE SP.PID = P.PID

Class Exercise - Part 3

- Now, in SQL, display a list of salesperson's total sales in dollars (subtotals by salesperson id)

Don’t look ahead !

Christian Brothers University

Q19

- SELECT SP.SID, SUM (SP.Qty * P.Price)
  - FROM SP, P
  - WHERE SP.PID = P.PID
  - GROUP BY SP.SID

Class Exercise - Part 4

- Finally, in SQL, display a list of salesperson's total sales in dollars where the total is greater than $ 20,000

Don’t look ahead !

Christian Brothers University

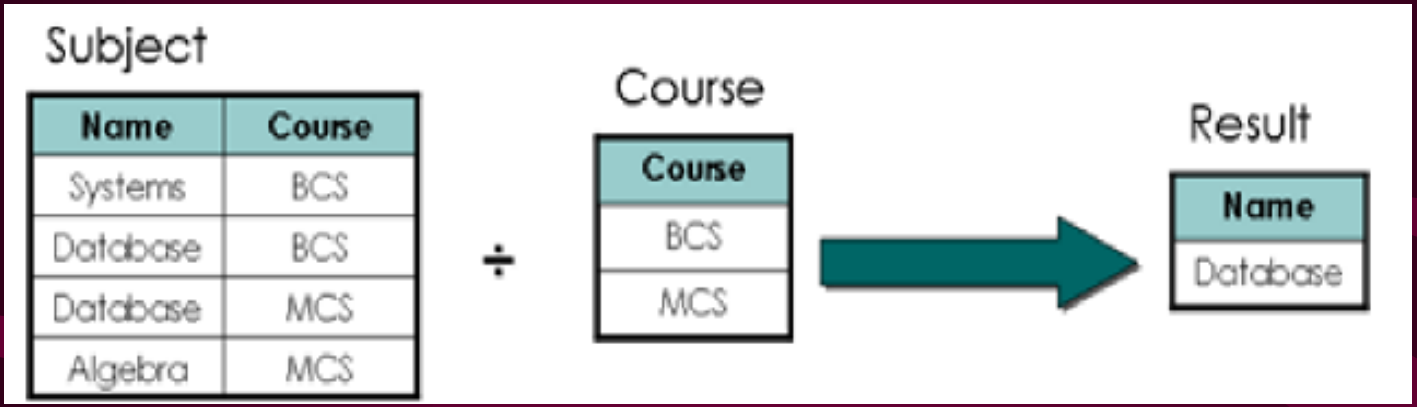
Q20

- SELECT SP.SID, SUM (SP.Qty * P.Price)
  - FROM SP, P
  - WHERE SP.PID = P.PID
  - GROUP BY SP.SID
  - HAVING SUM (SP.Qty * P.Price) > 20000

Which salesperson has sold some of all products?

- Relational algebra division operation
- T = DIVIDE SP BY P [PID] ;
- Not directly implemented in most SQL products
- The result (quotient) is only S5
- The interpretation of division is: Which sets of salesman rows (M=1) of the dividend (SP) have the attributes of the divisor (ie which S's in SP are related to all rows in the P table)

Relational Algebra Division



- Which unique row(s) in the dividend have all of the attributes of the divisor
- Note that this is the inverse of multiplication: The result (quotient) multiplied by the divisor plus the remainder yield the dividend

In SQL, which salesperson has sold some of all products?

- SELECT DISTINCT SP.SID
- FROM SP
- WHERE NOT EXISTS
  - (SELECT *
  - FROM P
  - WHERE NOT EXISTS
    - (SELECT *
    - FROM SP AS X
    - WHERE X.SID = SP.SID AND X.PID = P.PID));

"Where there does not exists a product he has not sold"

SQL Query Approach

- 1. Understand problem !
- 2. What tables are involved ?
- 3. If more than one table (or comparing within the same table), then what operation is needed between the tables:
  - Join (inner or outer), Union, Intersection, Difference
  - Sub-queries
- 4. What are the selection criterion ?
- 5. Any grouping or group selection (“having”) ?
- 5. What columns or totals need to be projected ?
- 6. Any sorting ?

References

- Introductory:
  - SQL for DUMMIES; Taylor,A; 1-56884-336-4
  - The Essence of SQL: A Guide to Learning Most of SQL in the Least Amount of Time; Rozenshtein; 0-9649812-1-1
- Advanced:
  - SQL for SMARTIES; Celko, J.; 1-55860-323-9

Courses ▾ Search Khan Academy

Computing

Computer programming

Intro to SQL: Querying and managing data

SQL basics

More advanced SQL queries

Relational queries in SQL

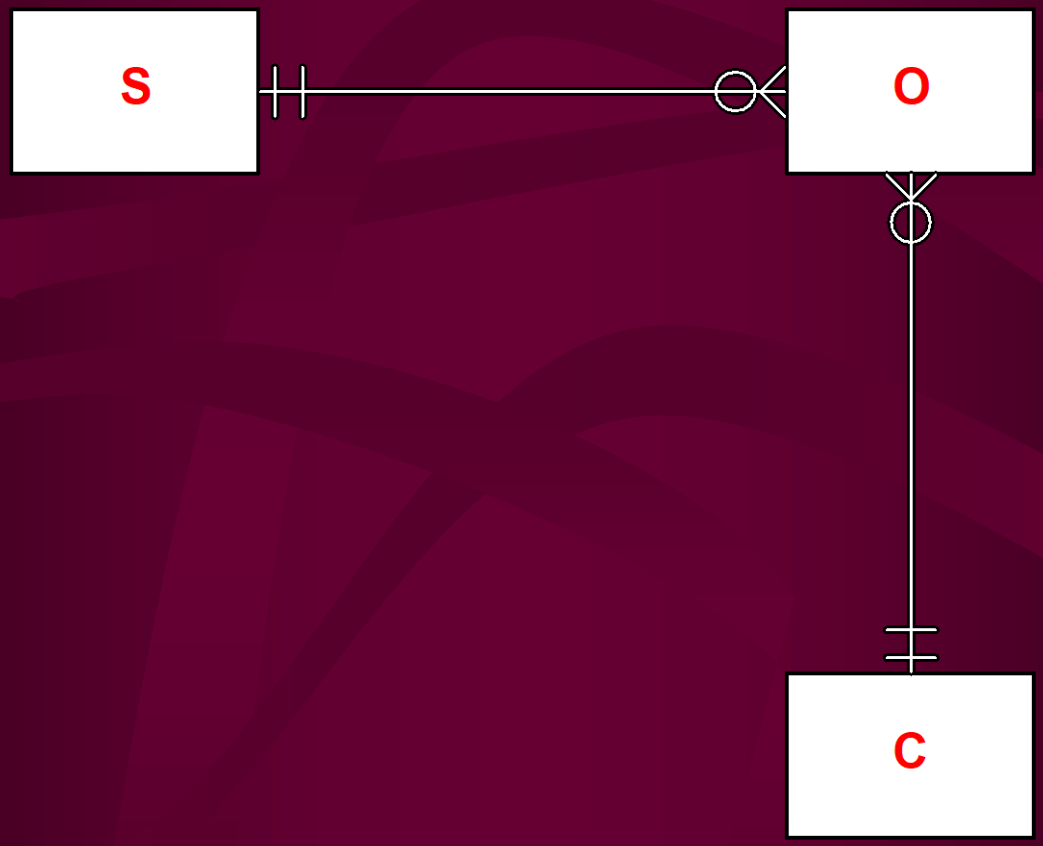
Modifying databases with SQL

Further learning in SQL

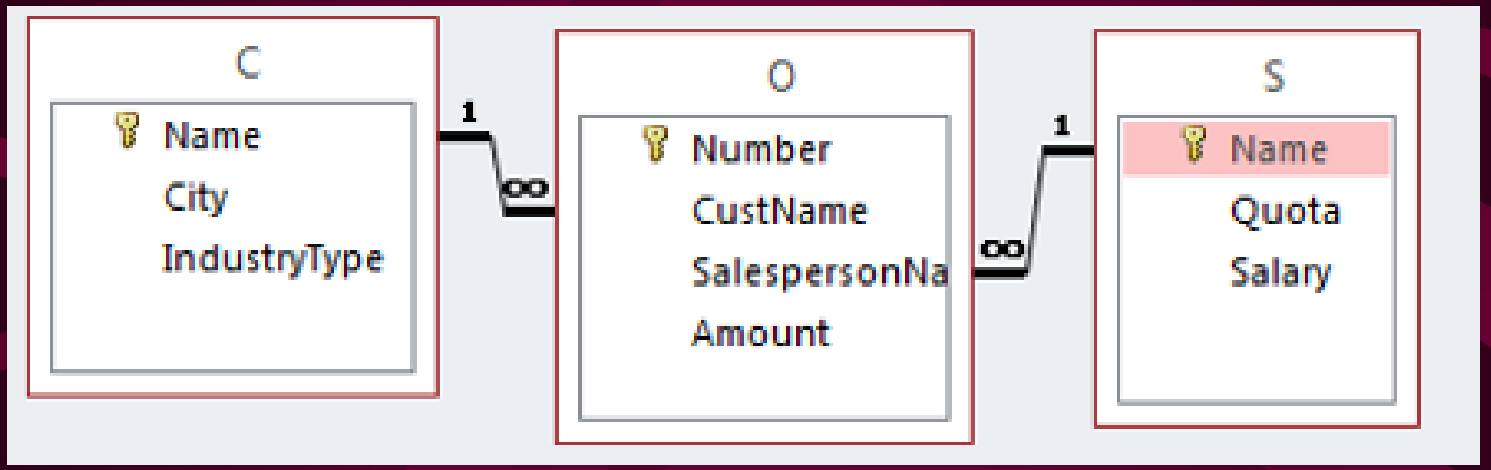
Homework

- Textbook – Chapter 7
- Access model for SOC queries
  - SOC.mdb - download from course web site- see next slide
    - Do “save as” first
- Using the Salesperson, Order, Customer Tables, build the SQL for the following queries:
  - Show standard SQL for each query
    - Submit Word file
  - Set up queries in Access
    - Submit Access file

SOC E-R Diagram



SOC Access Model



S (Salesperson Table)

Name	Quota	Salary
Abel	63	120000
Baker	38	42000
Jones	26	36000
Kobad	27	34000
Murphy	42	50000
Zenith	59	118000

C (Customer Table)

Name	City	IndustryType
Abernathy Construction	Willow	B
Amalgamated Housing	Memphis	B
Manchester Lumber	Manchester	F
Tri-City Builders	Memphis	B

O (Order Table)

Number	CustName	SalespersonNar	Amount
100	Abernathy Construction	Zenith	560
200	Abernathy Construction	Jones	1800
300	Manchester Lumber	Abel	480
400	Amalgamated Housing	Abel	2500
500	Abernathy Construction	Murphy	6000
600	Tri-City Builders	Abel	700
700	Manchester Lumber	Jones	150

SQL Homework Queries

- 1. Show the salaries of all salespeople.
- 2. Show the salaries of all salespeople but omit duplicates.
- 3. Show the names of all salespeople under 30 percent of quota.
- 4. Show the names of all salespeople who have an order with Abernathy Construction.
- 5. Show the names of all salespeople who earn more than $49,999 and less than 100,000.
- 6. Show the names of all salespeople with PercentOfQuota greater than 49 and less than 60. Use the BETWEEN keyword.
- 7. Show the names of all salespeople with PercentOf Quota greater than 49 and less than 60. Use the LIKE keyword.
- 8. Show the names of customers who are located in a City ending with S.
- 9. Show the names and salary of all salespeople who do not have an order with Abernathy Construction, in ascending order of salary.

SQL Homework Queries (con't)

- 10. Compute the number of orders.
- 11. Compute the number of different customers who have an order.
- 12. Compute the average percent of quota for salespeople.
- 13. Show the name of the salesperson with highest percent of quota.
- 14. Compute the number of orders for each salesperson.
- 15. Compute the number of orders for each salesperson, considering only orders for an amount exceeding 500.