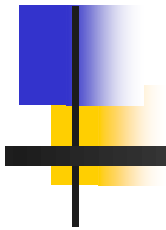
Introduction to Structured Query Language (SQL)

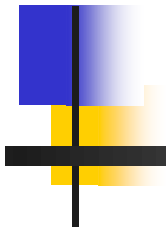
IOE 373 Lecture 04



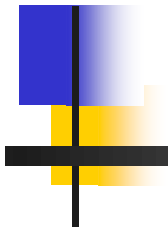
Topics

- SELECT Queries
- WHERE Clause
- ORDER BY
- GROUP BY

Intro to SQL
Getting Information Out of Database



- Now we have the database
- How do we actually use the data?
- How can we retrieve what we want from the database?



SQL

- SQL - Structured Query Language
- It can be pronounced “Sequel” or “S-Q-L”.
- It was originally developed in the early 1970s by IBM as a way to manipulate and retrieve data stored in IBM’s relational DBMS
- Now the standard query language for relational databases
  - SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What SQL Can Do

- Most commonly used to retrieve data from a database (**SELECT**)
- Also commonly used to modify data. (**INSERT, UPDATE, DELETE**)
- SQL also contains commands for creating tables and other database objects (**CREATE**)
- And for maintaining data security through granting and denying privileges to users.

SELECT Statement

- Basic Syntax:
  - **SELECT** [Fields] **FROM** [Table(s)]
  - **SELECT** [Fields] **FROM** [Table(s)] **WHERE** [Conditions] **ORDER BY** [Fields] **ASC/DESC**
- Example:
  - **SELECT** ItemID**,** ItemName **FROM** ItemList
- Use (*) to select all fields
  - SELECT * FROM Students

Using SQL

- SELECT * FROM Students

Students

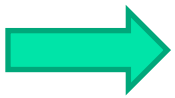
UMID	StName	Email	Gender
10000001	Steve	steve@notexist.com	M
10000002	John	john@notexist.com	M
10000003	Mary	mary@notexist.com	F
10000004	Emily	emily@notexist.com	F
10000005	Mike	mike@notexist.com	M
10000006	James	james@notexist.com	M

UMID	StName	Email	Gender
10000001	Steve	steve@notexist.com	M
10000002	John	john@notexist.com	M
10000003	Mary	mary@notexist.com	F
10000004	Emily	emily@notexist.com	F
10000005	Mike	mike@notexist.com	M
10000006	James	james@notexist.com	M

How SELECT Statements Work…

- SELECT UMID, StName FROM Students2

UMID	StName	Email	Gender
10000001	Steve	steve@notexist.com	M
10000002	John	john@notexist.com	M
10000003	Mary	mary@notexist.com	F
10000004	Emily	emily@notexist.com	F
10000005	Mike	mike@notexist.com	M
10000006	James	james@notexist.com	M



UMID	StName
10000001	Steve
10000002	John
10000003	Mary
10000004	Emily
10000005	Mike
10000006	James

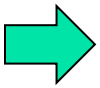
SELECT Statement

- Using alias in Fields

SELECT StName **AS** StudentName
FROM Students2

Students2

UMID	StName	StLastName	Email	Gender
10000001	Steve	Martin	steve@notexist.com	M
10000002	John	Legend	john@notexist.com	M
10000003	Mary	Smith	mary@notexist.com	F
10000004	Emily	Lake	emily@notexist.com	F
10000005	Mike	Jones	mike@notexist.com	M
10000006	James	Dee	james@notexist.com	M



StudentName
Steve
John
Mary
Emily
Mike
James

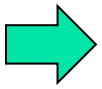
SELECT Statement

- Concatenates two fields

  SELECT StName + ' ' + StLastName **AS** Name
  FROM Students2

Students2

UMID	StName	StLastName	Email	Gender
10000001	Steve	Martin	steve@notexist.com	M
10000002	John	Legend	john@notexist.com	M
10000003	Mary	Smith	mary@notexist.com	F
10000004	Emily	Lake	emily@notexist.com	F
10000005	Mike	Jones	mike@notexist.com	M
10000006	James	Dee	james@notexist.com	M



Name
Steve Martin
John Legend
Mary Smith
Emily Lake
Mike Jones
James Dee

SELECT Statement

- Calculated fields:

  SELECT CarID, KM/1.61 **AS** Miles
  FROM CarData

CarData

CarId	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM
1	TOYOTA Corolla	13500	23	10	2002	46986
2	TOYOTA Corolla	13750	23	10	2002	72937
3	TOYOTA Corolla	13950	24	9	2002	41711
4	TOYOTA Corolla	14950	26	7	2002	48000
5	TOYOTA Corolla	13750	30	3	2002	38500
6	TOYOTA Corolla	12950	32	1	2002	61000
7	TOYOTA Corolla	16900	27	6	2002	94612
8	TOYOTA Corolla	18600	30	3	2002	75889
9	TOYOTA Corolla	21500	27	6	2002	19700
10	TOYOTA Corolla	12950	23	10	2002	71138
11	TOYOTA Corolla	20950	25	8	2002	31461
12	TOYOTA Corolla	19950	22	11	2002	43610
13	TOYOTA Corolla	19600	25	8	2002	32189
14	TOYOTA Corolla	21500	31	2	2002	23000
15	TOYOTA Corolla	22500	32	1	2002	34131
16	TOYOTA Corolla	22000	28	5	2002	18739

CarID	Miles
1	29183.850931677
2	45302.4844720497
3	25907.4534161491
4	29813.6645962733
5	23913.0434782609
6	37888.198757764
7	58765.2173913043
8	47136.0248447205
9	12236.0248447205
10	44185.0931677019
11	19540.9937888199
12	27086.9565217391
13	19993.1677018634
14	14285.7142857143
15	21199.3788819876

SELECT Statement

- SELECT * FROM table is a good way to look at the entire table
- However, it will take a long time if the database is large
- When writing a SQL query, use specific Field names rather than * whenever possible

WHERE Clause

- Use WHERE clause to only select rows that satisfy some conditions
- Syntax:
  - **SELECT** [Fields] **FROM** [Table(s)] **WHERE** [Conditions]
- Example
  - **SELECT** * **FROM** Students **WHERE** Gender = 'F'
  - **SELECT** Name **FROM** students **WHERE** Gender = 'F'

PersonID	Name	Gender	Email
1	Tom	M	tom@umich.edu
2	Amy	F	amy@umich.edu
3	Kate	F	kate@umich.edu
4	Edward	M	edward@umich.edu
5	Matt	M	matt@umich.edu

How SELECT Statements Work…

- SELECT UMID, StName FROM Students2 WHERE Gender = "F"

UMID	StName	Email	Gender
10000001	Steve	steve@notexist.com	M
10000002	John	john@notexist.com	M
10000003	Mary	mary@notexist.com	F
10000004	Emily	emily@notexist.com	F
10000005	Mike	mike@notexist.com	M
10000006	James	james@notexist.com	M

UMID	StName
10000003	Mary
10000004	Emily

SELECT Statement

1. Pick the columns you specified right after SELECT
2. For each row, check conditions in WHERE clause
3. Display results that show up in both 1 and 2

UMID	StName	Email	Gender
10000001	Steve	steve@notexist.com	M
10000002	John	john@notexist.com	M
10000003	Mary	mary@notexist.com	F
10000004	Emily	emily@notexist.com	F
10000005	Mike	mike@notexist.com	M
10000006	James	james@notexist.com	M

WHERE Clause

- WHERE Clause can also be used in other SQL commands, such as DELETE, UPDATE, INSERT
- You can use AND, OR, NOT, and parentheses to change the order of operation.
- Examples:
  - **WHERE** PlayerID = 7
  - **WHERE** TeamID=2 **AND** Age>24
  - **WHERE NOT** (TeamID=1 AND Age<=30)
  - **WHERE** Age>20 **OR** Age<30 [Will affect all records]
  - **WHERE** Age <20 **AND** Age>30 [Won't affect any record]

BETWEEN and IN in WHERE clauses

- BETWEEN and IN can be used to simplify WHERE clauses.
- BETWEEN combines two inequalities:

  WHERE Age **BETWEEN** 24 **AND** 27
  Is the same as

  WHERE Age>=24 **AND** Age<=27

  BETWEEN is always inclusive; that is, it includes the endpoints.
- **IN** combines multiple Ors:

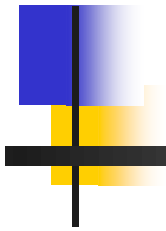
  WHERE TeamID **IN** (2,3,5,8)
  Is the same as

  WHERE TeamID=2 **OR** TeamID=3 **OR** TeamID=5 **OR** TeamID=8

ORDER BY Clause

- By default, the query result is not sorted
- To sort the result, use ORDER BY Clause:

  SELECT * **FROM** Players **ORDER BY** Age **ASC**
- **ASC** stands for "Ascending" order, **DESC** is the opposite
- Now youngest players is on top of the list, with the oldest at the bottom
- **ASC** is the default if you don't specify **ASC** or DESC



ORDER BY Clause

- You can have more than one fields in ORDER BY Clause
- **SELECT** LastName, FirstName **FROM** Players **ORDER BY** LastName, FirstName
- First sorted by LastName, then FirstName (tie-breaker)
- Players with the same last names, those with first names beginning with A will be listed before those beginning with B

ORDER BY Clause

- You can also sort two fields with different orders
- **SELECT** * **FROM** Players **ORDER BY** TeamID **ASC**, Age **DESC**
- This query will display all members of team 1 before all members of team 2, but within each team, the oldest players will be listed first

TOP Clause

- Use TOP clause to select only the first of several records selected

  SELECT TOP 5 LastName, FirstName, GPA **FROM** Students **ORDER BY** GPA **DESC**

- This will return the best 5 students in class
- Don’t use this:

  SELECT TOP 5 LastName, FirstName, GPA **FROM** Students **ORDER BY** GPA

- This will only select the worst 5 students.

UNION Operator

- The **UNION** operator combines the results of multiple **SELECT** statements into one result.
- List all employees and managers with names starting with `'Z'`

```
(SELECT Name, Salary FROM Employees
  WHERE name LIKE 'Z*')
    UNION
 (SELECT Name, Bonus FROM Managers
   WHERE name LIKE 'Z*')
```

- List all cities with names starting with `'Z'` or `'Y'`

```
(SELECT CityName, Population FROM City   WHERE CityName LIKE 'Z*')
UNION
(SELECT CityName, Population FROM City WHERE CityName LIKE 'Y*')
```

UNION Operator

- Remark: one important rule for UNION operation is that the queries' fields must match.
- Each query must return the same number of fields and corresponding fields must have compatible data types
- In previous example, the first column (of both queries) is text type, and the second column is number type, so the two queries are compatible

Aggregate Data

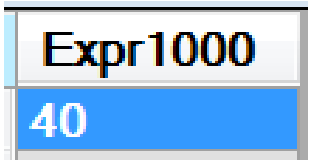
- **SELECT MAX(**Age**) AS** MaxAge **FROM** Players
  - Returns the age of oldest players (one row, one column result)
- **SELECT AVG(**Age**) AS** AvgAge **FROM** Players
  - Average age of players.
- **SELECT COUNT(*) AS** NumOldPlayer **FROM** Players **WHERE** Age **>= 30**
  - Number of players age 30 or older

Aggregate Data

- **MIN**: Find the minimum value (or the earliest date for dates) in the specified field in the rows meeting the conditions.
- **MAX**: Find the maximum value (or the latest date for dates) in the specified field in the rows meeting the conditions.
- **SUM**: Add the values in the specified field in the rows meeting the conditions.
- **AVG**: Find the mean value of the specified field in the rows meeting the conditions.
- **STDEV**: Find the standard deviation of the values of the specified field in the rows meeting the conditions.
- **COUNT**: Find out how many rows there are which meet the conditions. For this course, always use COUNT(*), not COUNT(SomeField). They generally return the same numbers, but doing COUNT on a specific field may return a smaller number if that field contains NULLs

AAA: Always Alias Aggregates

- If your query says
- SELECT MAX(Age) FROM Players
- Access will return the following:



  - Access doesn't have a name for MAX(Age), so it makes up one: "Expr1000"
  - You should always alias aggregate with a meaningful name.
  - **SELECT MAX**(Age) **AS** MaxAge **FROM** Players

GROUP BY

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s)

GROUP BY

- Consider this table called *Workers*

WorkerID	YearsOfSchool	Sex	YearsWorking	UnionMember	Wage	Age	JobType
1026	12	M	8	No	$9.36	26	Other
1053	12	F	16	No	$4.80	34	Other
1080	14	M	14	No	$13.98	34	Other
1107	5	M	44	No	$14.00	55	Other
1134	9	M	16	No	$4.80	31	Other
1161	13	M	32	No	$21.25	51	Management
1188	14	M	16	No	$16.14	36	Management
1215	14	M	2	No	$7.50	22	Sales
1242	14	M	0	No	$5.00	20	Sales
1269	12	M	20	Yes	$9.00	38	Clerical
1296	12	F	15	No	$4.10	33	Clerical
1323	12	F	26	No	$5.00	44	Clerical
1350	2	M	16	No	$3.75	24	Service
1377	12	F	6	No	$3.60	24	Service
1404	11	M	3	No	$7.50	20	Service
1431	17	M	31	No	$24.98	54	Professional
1458	18	M	7	No	$5.71	31	Professional
1485	18	F	40	No	$22.20	64	Professional
1512	18	M	23	Yes	$12.00	47	Professional

•If you want to know the maximum age of all workers, you use this query:
•**SELECT MAX(Age) AS MaxAge FROM Workers**
•which will return

MaxAge
64

GROUP BY

- But if you want to know the maximum age by JobType, use GROUP BY:
- **SELECT** JobType**, MAX(**Age**) AS** MaxAge **FROM** Workers **GROUP BY** JobType
- Which returns

JobType	MaxAge
Clerical	44
Management	51
Other	55
Professional	64
Sales	22
Service	24

Another GROUP BY Example

- If you want to know how many men and women there are in the list:
- **SELECT** Sex, **COUNT**(*) **AS** Num **FROM** Workers **GROUP BY** Sex

Sex	Num
F	5
M	14

And Another One

- **SELECT** JobType**, Sex, COUNT(*) AS** Num **FROM** Workers **GROUP BY** JobType, Sex

JobType	Sex	Num
Clerical	F	2
Clerical	M	1
Management	M	2
Other	F	1
Other	M	4
Professional	F	1
Professional	M	3
Sales	M	2
Service	F	1
Service	M	2

MAX/MIN vs. TOP

- Above, we saw the results of this query:
- **SELECT MAX(**Age**) AS** MaxAge **FROM** Workers

MaxAge
64

However, we can get the same result using TOP:

SELECT TOP 1 Age **FROM** Workers **ORDER BY** Age **DESC**

In addition, we can find out everything about the oldest worker using TOP, something we can't do with MAX:

SELECT TOP 1 * **FROM** Workers **ORDER BY** Age **DESC**

WorkerID	YearsOfSchool	Sex	YearsWorking	UnionMember	Wage	Age	JobType
1485	18	F	40	No	22.2	64	Professional

Combining TOP and Aggregates

- If you want to find one maximum or minimum and want to know something about the record that possesses that value, use TOP. In most cases, it will be preferable to MAX or MIN.
- However, if you want to calculate maximum or minimums by categories, use MAX and MIN along with **GROUP BY**. You can even combine **TOP** with **GROUP BY**:
- **SELECT TOP** 3 JobType**, AVG(**Wage**) AS** AvgWage **FROM** Workers **GROUP BY** JobType **ORDER BY AVG(**Wage**) DESC**

JobType	AvgWage
Management	18.695
Professional	16.2225
Other	9.388

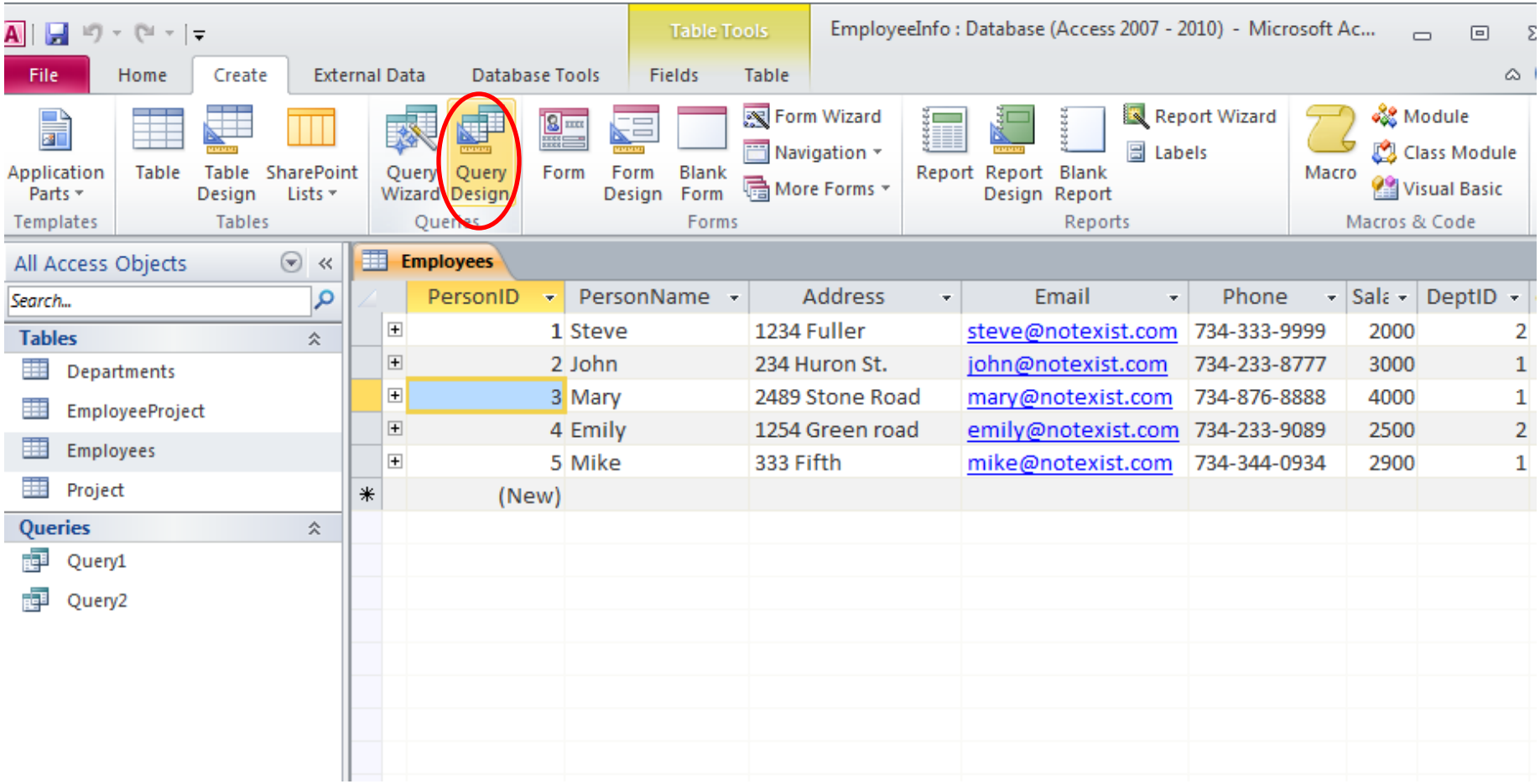
- This tells us the three highest paying job categories and the corresponding average wages.

Using SQL In Access

- Open the EmployeeInfo Database (downloadable from Canvas)

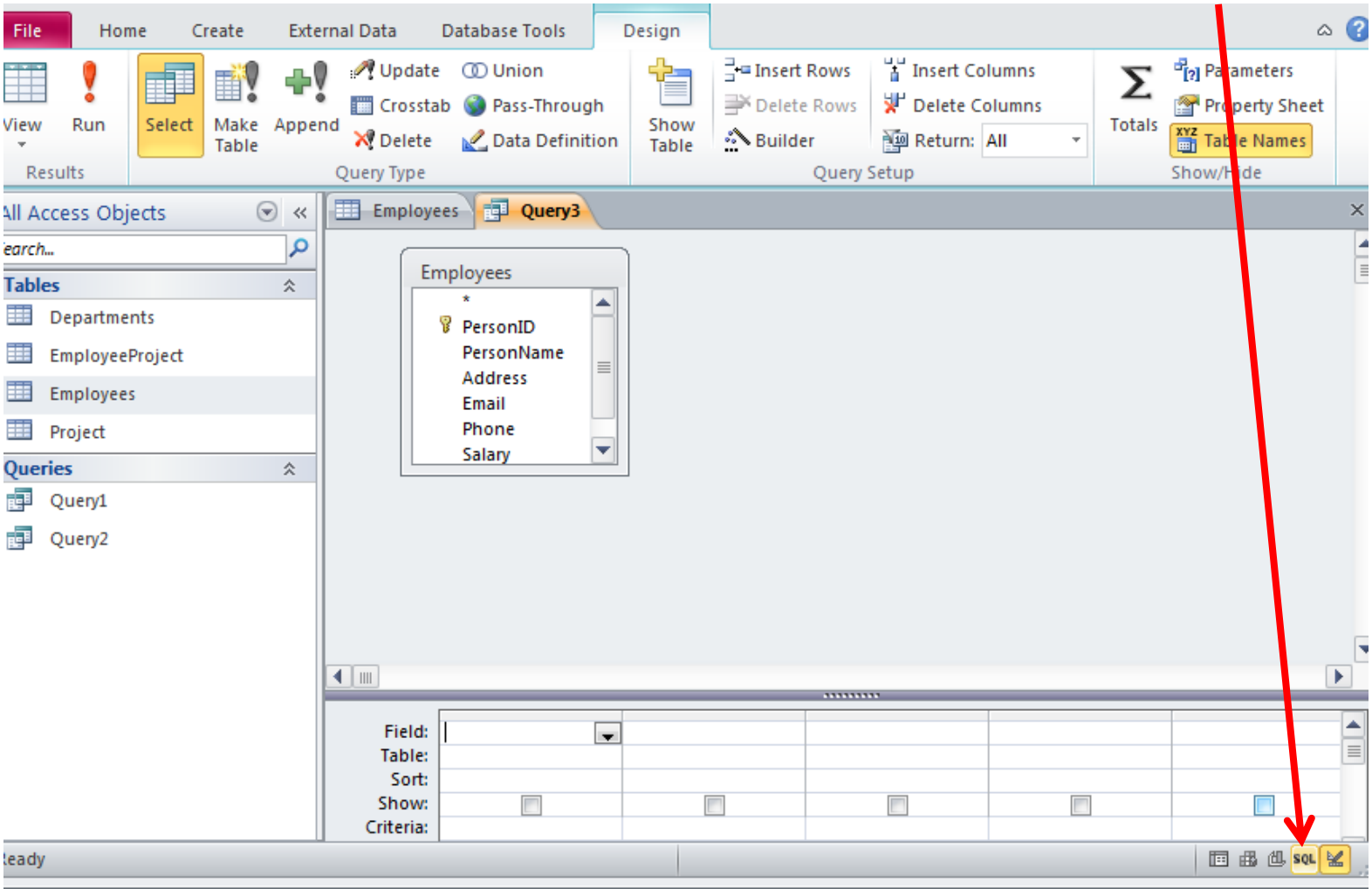
SQL In Access

- Click “Query Design” , Select tables



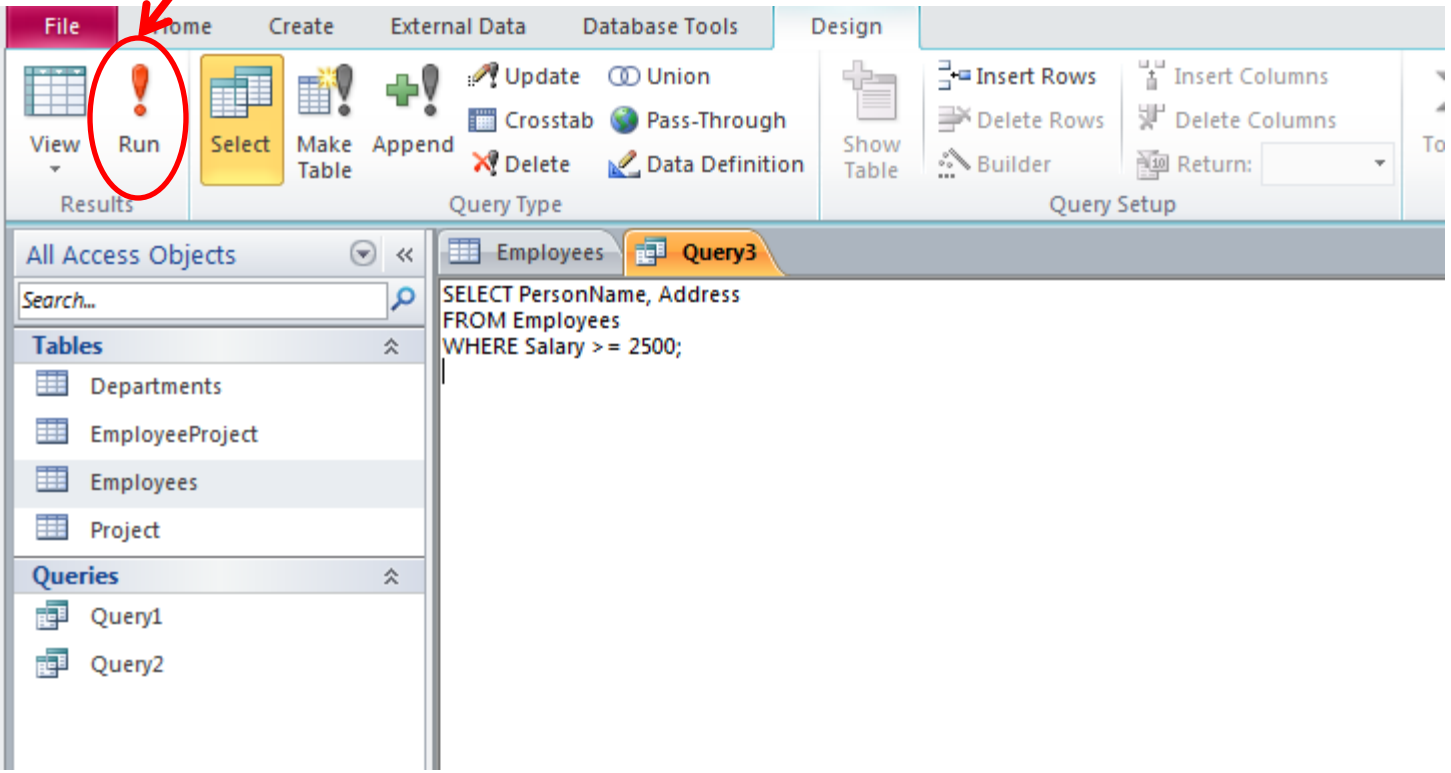
Enter SQL Mode

- Click "SQL" Button at the bottom right



Type In SQL Query

- Type in your SQL query and click Run



- You will see the result of the query