SQL Selects

Lecture 13

Wednesday - Oct 18, 2023

Housekeeping

- Focus on getting everything to work!
- YES - you MUST be able render explore.qmd.

Module	Week	Date	Day	Lectures/Quizzes	Deliverables/Notes
Intro to SQL	9	10/18	Wed	MTG16: L13 (SQL Selects)	
Intro to SQL	9	10/20	Fri		No classes: Reading Day; Midterm Grades Due
SQL	10	10/23	Mon	MTG17: L14 (SQL Create/Insert/Delete/Update)	
SQL	10	10/25	Wed	MTG18: L15 (Referential Integrity 1)	
SQL	10	10/27	Fri		Last day for "W"
SQL	10	10/29	Sun		HW6 due (Getting started with SQL)
SQL	11	10/30	Mon	MTG19: L16 (Databases and APIs)	
SQL	11	11/1	Wed	MTG20: Quiz 4 today (SQL Basics)	

Homework 6 - connecting to DBMS

Homework 6 - connecting to DBMS

Overview of the assignment

- We're using mysql DBMS.
- There are multiple ways to connect
- Lots of tools to install

Each approach is handy for one aspect of development. You need to use all the ways!

Ways to connect

1. phpMyAdmin - via the web
2. ssh - to a linux terminal
3. vscode - using SQLTools extension
4. python - using a connection

Homework 6 - connecting to DBMS

1. phpmyadmin

- http://cmcs508.com/phpmyadmin
- Username: 23FA_, 23FA_jdleonard
- Password: Shout4__GOME
- for example, Shout4_jdleonard_GOME

This approach is most useful for managing the DBMS but NOT very useful for development.

2. SSH to the server

- Open a terminal window on your local machine.
- Enter these commands on your terminal.

```
ssh 23FA_jdleonard@cmsc508.com
password: <your EID goes here>

mysql -p
password: <your Shout4 GOME password>
```

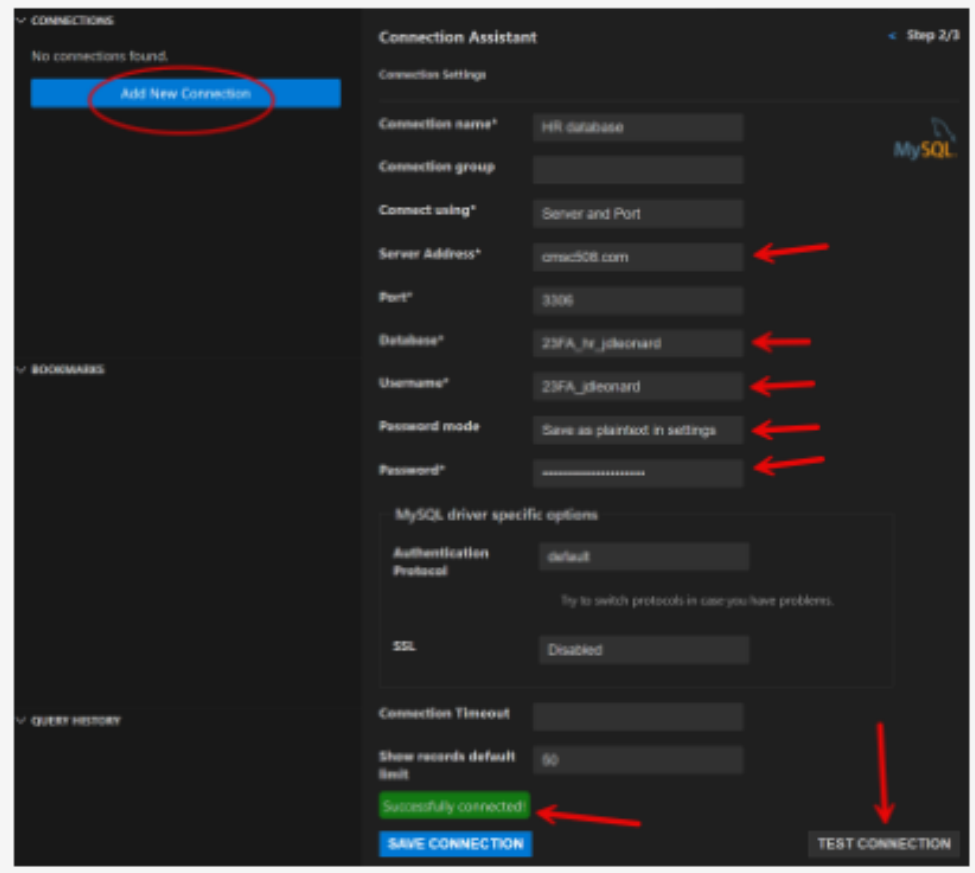
This is the quickest and most efficient approach for directly managing the DBMS but not very useful for development.

Homework 6 - connecting to DBMS

3. Visual Studio Code

- Install:
  - SQLTOOLS extension
  - SQLTOOLS mysql connector
- You should see a DB icon on the left
- See the connection screen!

This approach is most useful for directly building and coding SQL code. Lots of cutting and pasting commands through an DB window.



Homework 6 - connecting to DBMS

4. Python code

- Create your hw6 repository
- Install pyenv, poetry, quarto
- Install libraries
  - Auto: using poetry
  - Manually using pip
  - matplotlib jupyter plotly
  - python-dotenv sqlalchemy pymysql
- Create a .env file
- Create your python code

See explorer.qmd for details!

```python
import os
import pandas as pd
from dotenv import load_dotenv
from sqlalchemy import create_engine,text
# load credentials from .env file
load_dotenv()
# store these credentials a dictionary for later reference
config = {
  'user': os.getenv("HW6_USER"),
  'password': os.getenv("HW6_PASSWORD"),
  'host': os.getenv("HW6_HOST"),
  'database': os.getenv("HW6_DB_NAME")
}
engine_uri = f"mysql+pymysql://{config['user']}:{config['password']}

# create a database connection.  THIS IS THE ACTUAL CONNECTION!
cnx = create_engine(engine_uri)
```

This approach is most useful for developing the wrapper application around the SQL code.

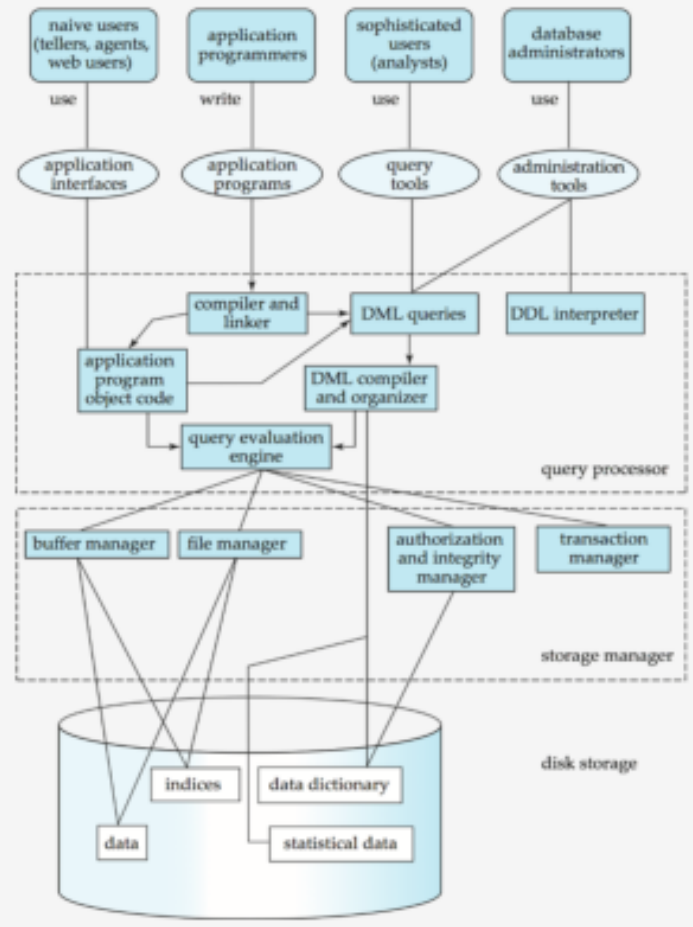
MySQL DBMS

MYSQL DBMS

Leveraging the power

What not simply use 2-D arrays or other data structures in our programs? Why use a data base at all?

- The DBMS will DO LOTS OF WORK for us AUTOMATICALLY. We simply need to know how to tell it what we want it to do.
- We can provide these management and operating rules when tables are created.
- The DBMS will enforce the rules in real-time behind the scenes.
- The DBMS maintains several databases specifically for the management of the database itself!



MySQL DBMS

MySQL 8.0 Reference Manual

- Preface and Legal Notices
- General Information
- Installing and Upgrading MySQL
- Tutorial
- MySQL Programs
- MySQL Server Administration
- Security
- Backup and Recovery
- Optimization
- Language Structure
- Character Sets, Collations, Unicode
- Data Types
- Functions and Operators
- SQL Statements
- MySQL Data Dictionary
- The InnoDB Storage Engine
- Alternative Storage Engines
- Replication
- Group Replication
- MySQL Shell
- Using MySQL as a Document Store
- InnoDB Cluster
- InnoDB ReplicaSet
- MySQL NDB Cluster 8.0
- Partitioning
- Stored Objects
- INFORMATION_SCHEMA Tables
- MySQL Performance Schema
- MySQL sys Schema

Documentation Home

MySQL 8.0 Reference Manual

- Preface and Legal Notices
- General Information
- Installing and Upgrading MySQL
- Tutorial
- MySQL Programs
- MySQL Server Administration

- Character Sets, Collations, Unicode
- Data Types
- Functions and Operators
- SQL Statements
- MySQL Data Dictionary
- The InnoDB Storage Engine
- Alternative Storage Engines

- InnoDB Cluster
- InnoDB ReplicaSet
- MySQL NDB Cluster 8.0
- Partitioning
- Stored Objects
- INFORMATION_SCHEMA Tables
- MySQL Performance Schema
- MySQL sys Schema
- Connectors and APIs
- MySQL Enterprise Edition
- MySQL Workbench
- MySQL on the OCI Marketplace
- MySQL 8.0 Frequently Asked Questions

MYSQL Documentation

Chapter 13 SQL Statements

Table of Contents

13.1 Data Definition Statements
13.2 Data Manipulation Statements
13.3 Transactional and Locking Statements
13.4 Replication Statements
13.5 Prepared Statements
13.6 Compound Statement Syntax
13.7 Database Administration Statements
13.8 Utility Statements

Two types of statements:

- Data Definition Statement (DDL)
- Data Manipulation Statements (DML)

https://dev.mysql.com/doc/refman/8.0/en/

DDL - Data Definition Language

13.1.2 ALTER DATABASE
13.1.3 ALTER EVENT
13.1.4 ALTER FUNCTION
13.1.5 ALTER INSTANCE
13.1.6 ALTER LOGFILE GROUP
13.1.7 ALTER PROCEDURE
13.1.8 ALTER SERVER
13.1.9 ALTER TABLE
13.1.10 ALTER TABLESPACE
13.1.11 ALTER VIEW
13.1.12 CREATE DATABASE
13.1.13 CREATE EVENT
13.1.14 CREATE FUNCTION
13.1.15 CREATE INDEX
13.1.16 CREATE LOGFILE GROUP
13.1.17 CREATE PROCEDURE and CREATE FUNCTIONs
13.1.18 CREATE SERVER
13.1.19 CREATE SPATIAL REFERENCE SYSTEM
13.1.20 CREATE TABLE
13.1.21 CREATE TABLESPACE
13.1.22 CREATE TRIGGER
13.1.23 CREATE VIEW
13.1.24 DROP DATABASE
13.1.25 DROP EVENT
13.1.26 DROP FUNCTION
13.1.27 DROP INDEX
13.1.28 DROP LOGFILE GROUP
13.1.29 DROP PROCEDURE and DROP FUNCTIONs
13.1.30 DROP SERVER
13.1.31 DROP SPATIAL REFERENCE SYSTEM
13.1.32 DROP TABLE
13.1.33 DROP TABLESPACE
13.1.34 DROP TRIGGER
13.1.35 DROP VIEW
13.1.36 RENAME TABLE
13.1.37 TRUNCATE TABLE

DML - Data Manipulation Language

13.2.1 CALL Statement
13.2.2 DELETE Statement
13.2.3 DO Statement
13.2.4 HANDLER Statement
13.2.5 IMPORT TABLE Statement
13.2.6 INSERT Statement
13.2.7 LOAD DATA Statement
13.2.8 LOAD XML Statement
13.2.9 REPLACE Statement
13.2.10 SELECT Statement
13.2.11 Subqueries
13.2.12 TABLE Statement
13.2.13 UPDATE Statement
13.2.14 VALUES Statement
13.2.15 WITH (Common Table Expressions)

SELECT statement

SELECT clause

A *SELECT* query is the one of the most fundamental statements in SQL.

A *SELECT* statement describe a request to perform operations within the database and return a *result set*.

This *result set* is a relation.

This *result* can be used in subsequent operations or stored back into the DB as a new table.

A SQL query is relational algebra:

$$\Pi_{A_1,A_2,\ldots A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_n))$$

SELECT A1, A2, A3

FROM R1,R2

WHERE P

ORDER BY A2,A3,A1

- A* represents an attribute (column), a literal, a function, or an operation
- R* represents a relation (table), cartesian project, or join
- P is a predicate, conditions or filters

SELECT clause and examples

- The select clause lists the attributes desired in the result of a query, corresponds to the projection operation of the relational algebra
- SQL names are case insensitive
- SQL allows duplicates in relations as well as in query results
- To force the elimination of duplicates, use the keyword distinct
- An asterisk in the select clause denotes “all attributes”
- May rename columns using alias

```
-- this is syntax
SELECT [distinct] {*, column [[as] alias], ...}
FROM table {or tables}

-- this is a real statement (and comment!)
SELECT
  *
FROM
  REGIONS

-- indents are not managed
-- comments stop processing of line
```


SELECT clause and examples

Using * to select all columns

```
result = pd.read_sql("""
select * from regions
""",cnx)
result
```

	region_id	region_name
0	1	Europe
1	2	Americas
2	3	Asia
3	4	Middle East and Africa

Selecting only one attribute

```
result = pd.read_sql("""
select
  region_name
from
  regions
""",cnx)
result
```

	region_name
0	Europe
1	Americas
2	Asia
3	Middle East and Africa

SELECT clause and examples

Gathering all last names

```
1 result = pd.read_sql("""
2 select last_name from employees
3 """,cnx)
4 result
```

	last_name
0	Abel
1	Ande
2	Atkinson
3	Austin
4	Baer
...	...
102	Vollman
103	Walsh
104	Weiss
105	Whalen
106	Zlotkey

Eliminating duplicates with DISTINCT

```
1 result = pd.read_sql("""
2 select distinct last_name from employees
3 """,cnx)
4 result
```

	last_name
0	Abel
1	Ande
2	Atkinson
3	Austin
4	Baer
...	...
97	Vollman
98	Walsh
99	Weiss
100	Whalen
101	Zlotkey

SELECT clause and examples

Combining last and first names

```
1 result = pd.read_sql("""
2 select concat(last_name,',',first_name)
3 from employees
4 """,cnx)
5 result.head(7)
```

	concat(last_name,',',first_name)
0	Abel,Ellen
1	Ande,Sundar
2	Atkinson,Mozhe
3	Austin,David
4	Baer,Hermann
5	Baida,Shelli
6	Banda,Amit

Then renaming the column

```
1 result = pd.read_sql("""
2 select concat(last_name,',',first_name) as full_name
3 from employees
4 """,cnx)
5 result.head(7)
```

	full_name
0	Abel,Ellen
1	Ande,Sundar
2	Atkinson,Mozhe
3	Austin,David
4	Baer,Hermann
5	Baida,Shelli
6	Banda,Amit

SELECT clause and examples

attribute can be literal or function

```
result = pd.read_sql("""

select
  27 as "Literal",
  now() as "Current Time"
from
  dual

""",cnx)
result.head(7)
```

	Literal	Current Time
0	27	2023-10-24 11:35:11

Note that we're using a built-in table named *dual* that returns what you put in as a table that can be used in subsequent operations.

Doing computations

The attribute can be a function or arithmetic operation

```
result = pd.read_sql("""
select concat(last_name,',',first_name) as full_name,
salary*12 as "Annual Salary"
from employees
""",cnx)
result.head(5)
```

	full_name	Annual Salary
0	King,Steven	288000.0
1	Kochhar,Neena	204000.0
2	De Haan,Lex	204000.0
3	Hunold,Alexander	108000.0
4	Ernst,Bruce	72000.0

WHERE clause and examples

- The WHERE clause specifies conditions that the results must satisfy, correspoding to the σ predicate in relational algebra.
- An individual WHERE element returns TRUE or FALSE for every row. This is applied during the query and will return only rows that evaluate TRUE in the where clause.
- Attributes in where clause can be compared using relational operators: < <= = >= >
- WHERE elements can be combined using *AND*, *OR* and *NOT* logical operators.
- WHERE also uses special operators: *BETWEEN*, *IN* and *IS NULL*

```
result = pd.read_sql("""
-- note that this is a monthly salary!
SELECT
  last_name, first_name, salary
FROM
  employees
WHERE
  salary > 10000
""",cnx)
result.head(5)
```

	last_name	first_name	salary
0	King	Steven	24000.0
1	Kochhar	Neena	17000.0
2	De Haan	Lex	17000.0
3	Greenberg	Nancy	12008.0
4	Raphaely	Den	11000.0

WHERE clause and examples

Selecting a department

```
result = pd.read_sql("""
select last_name,department_id
from employees
where department_id=110;
""",cnx)
result.head(7)
```

	last_name	department_id
0	Higgins	110
1	Gietz	110

Data types matter. You'll get an error if you set *department_id='110'*

Using BETWEEN

The attribute can be a function or arithmetic operation

```
result = pd.read_sql("""
select concat(last_name,',',first_name) as full_name, s
from employees
where salary BETWEEN 10000 and 12000;
""",cnx)
result.head(5)
```

	full_name	salary
0	Raphaely,Den	11000.0
1	Errazuriz,Alberto	12000.0
2	Cambrault,Gerald	11000.0
3	Zlotkey,Eleni	10500.0
4	Tucker,Peter	10000.0

WHERE clause and examples

Using IN

```
result = pd.read_sql("""
select last_name,department_id
from employees
where department_id in (100,145,146)
""",cnx)
result.head(7)
```

	last_name	department_id
0	Greenberg	100
1	Faviet	100
2	Chen	100
3	Sciarra	100
4	Urman	100
5	Popp	100

The long way

```
result = pd.read_sql("""
select last_name,department_id
from employees
where department_id=30 or department_id=70 or departmer
""",cnx)
result
```

	last_name	department_id
0	Raphaely	30
1	Khoo	30
2	Baida	30
3	Tobias	30
4	Himuro	30
5	Colmenares	30
6	Baer	70

WHERE clause and examples

What are all the department IDs?

```
result = pd.read_sql("""
select distinct department_id
from employees
order by department_id
""",cnx)
result
```

	department_id
0	NaN
1	10.0
2	20.0
3	30.0
4	40.0
5	50.0
6	60.0
7	70.0
8	80.0
9	90.0

Who is missing a manager?

```
result = pd.read_sql("""
select last_name,job_id,manager_id
from employees
where manager_id is NULL
""",cnx)
result.head(7)
```

	last_name	job_id	manager_id
0	King	AD_PRES	None

NULL must be all caps!!

Computations

Calculating commissions

```
result = pd.read_sql("""

select
  concat(last_name,',',first_name) as "Name",
  salary as "Monthly",
  12.0*salary+commission_pct*salary as "With/Commission
from
  employees
order by
  last_name,
  first_name

""",cnx)
result
```

What are the NaN “not a number” entries?

How do we remove the NaN (not a number) from list?

Calculating commissions

	Name	Monthly	With/Commission
0	Abel,Ellen	11000.0	135300.0
1	Ande,Sundar	6400.0	77440.0
2	Atkinson,Mozhe	2800.0	NaN
3	Austin,David	4800.0	NaN
4	Baer,Hermann	10000.0	NaN
...	...	...	...
102	Vollman,Shanta	6500.0	NaN
103	Walsh,Alana	3100.0	NaN
104	Weiss,Matthew	8000.0	NaN
105	Whalen,Jennifer	4400.0	NaN
106	Zlotkey,Eleni	10500.0	128100.0

107 rows × 3 columns

IFNULL operator

The IFNULL operator

IFNULL(expr1, expr2) replaces NULL with a value.

- IF expr1 is NOT NULL then returns expr1
- IF expr1 IS NULL, then returns expr2.

Calculating commissions

```
result = pd.read_sql("""
select
  concat(last_name,',',first_name) as "Name",
  salary as "Monthly",
  IFNULL(12.0*salary+commission_pct*salary,0.0) as "Wit
from employees
order by last_name, first_name
""",cnx)
result
```


IFNULL operator

IFNULL(expr1, 0.0)

	Name	Monthly	With/Commission
0	Abel,Ellen	11000.0	135300.0
1	Ande,Sundar	6400.0	77440.0
2	Atkinson,Mozhe	2800.0	0.0
3	Austin,David	4800.0	0.0
4	Baer,Hermann	10000.0	0.0
...	...	...	...
102	Vollman,Shanta	6500.0	0.0
103	Walsh,Alana	3100.0	0.0
104	Weiss,Matthew	8000.0	0.0
105	Whalen,Jennifer	4400.0	0.0
106	Zlotkey,Eleni	10500.0	128100.0

107 rows × 3 columns

IFNULL(expr1, 'missing')

	Name	Monthly	With/Commission
0	Abel,Ellen	11000.0	135300.0000
1	Ande,Sundar	6400.0	77440.0000
2	Atkinson,Mozhe	2800.0	missing
3	Austin,David	4800.0	missing
4	Baer,Hermann	10000.0	missing
...	...	...	...
102	Vollman,Shanta	6500.0	missing
103	Walsh,Alana	3100.0	missing
104	Weiss,Matthew	8000.0	missing
105	Whalen,Jennifer	4400.0	missing
106	Zlotkey,Eleni	10500.0	128100.0000

107 rows × 3 columns

String operations

The operator *LIKE* uses patterns (case insensitive) (use *LIKE BINARY* for case sensitive) for string-matching operations using two special characters:

* percentage (%) matches any substring (none or many characters)
* underscore (_) matches any single character

Examples:

'Intro%' matches any string beginning with "Intro"

'%Comp%' matches any string containing "Comp" as a substring

'_ _ _' matches any string of exactly three characters

'_ _ _ %' matches any string of at least three characters

'%_ a _' same as before but the second to the last letter is 'a'

String operations

DOUBLE %% when using Python

```
result = pd.read_sql("""
select last_name from employees
where last_name like 'MC%%'
""",cnx)
result
```

	last_name
0	McCain
1	McEwen

DOUBLE %% when using Python

```
result = pd.read_sql("""
SELECT phone_number FROM employees
WHERE phone_number LIKE '%%123%%';
""",cnx)
result
```

	phone_number
0	515.123.4567
1	515.123.4568
2	515.123.4569
3	650.123.1234
4	650.123.2234
5	650.123.3234
6	650.123.4234
7	650.123.5234
8	650.121.1234
9	515.123.4444

ORDER BY clause

ORDER BY is used to specify the sort order in the result set.

The result set is sorted by the first attribute listed in the *ORDER BY*.

If there is a tie, *ORDER BY* moves to the second attribute, and so on.

Attribute expression may be modified with ASC (the default) or DESC (for descending)

Attributes in *ORDER BY* can be computed!

Example

```
result = pd.read_sql("""
select
  concat(last_name,',',first_name) as "Full Name", sala
from employees
order by
  salary desc,
  last_name, first_name
""",cnx)
result
```

	Full Name	Monthly
0	King,Steven	24000.0
1	De Haan,Lex	17000.0
2	Kochhar,Neena	17000.0
3	Russell,John	14000.0
4	Partners,Karen	13500.0
...	...	...
102	Gee,Ki	2400.0
103	Landry,James	2400.0

Housekeeping

- Focus on getting everything to work!
- YES - you MUST be able render explore.qmd.

Module	Week	Date	Day	Lectures/Quizzes	Deliverables/Notes
Intro to SQL	9	10/18	Wed	MTG16: L13 (SQL Selects)	
Intro to SQL	9	10/20	Fri		No classes: Reading Day; Midterm Grades Due
SQL	10	10/23	Mon	MTG17: L14 (SQL Create/Insert/Delete/Update)	
SQL	10	10/25	Wed	MTG18: L15 (Referential Integrity 1)	
SQL	10	10/27	Fri		Last day for "W"
SQL	10	10/29	Sun		HW6 due (Getting started with SQL)
SQL	11	10/30	Mon	MTG19: L16 (Databases and APIs)	
SQL	11	11/1	Wed	MTG20: Quiz 4 today (SQL Basics)	