

# CS 61: Database Systems

Access via programming languages

# Agenda



1. Direct database access
2. Web APIs
3. Node.js

# Python can directly query the database

## Steps to access MySQL from Python

1. Install connector to MySQL:

```
sudo pip install mysql-connector-python
```

2. Get a connection to the database

```
cnx=mysql.connector.connect(user=<username>, password=<pwd>,  
                             host="sunapee.cs.dartmouth.edu",  
                             database="nyc_inspections")
```

Create a database user with  
minimal necessary rights

3. Query the database

```
cursor = cnx.cursor()  
query = ("SELECT RestaurantID, RestaurantName, Boro "  
        +"FROM Restaurants r JOIN Cuisine c USING (CuisineID) "  
        +"WHERE RestaurantName LIKE %s") #%s is a parameter  
cursor.execute(query, ('%'+restaurant_name+'%',))#query,params
```

Cursors are like an iterator

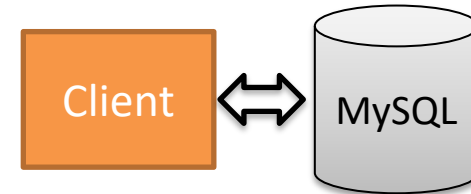
- Read only
- Non-scrollable

4. Loop over results

```
for row in cursor:  
    print str(row)
```

# get\_restaurants.py is example of client-side Python code directly querying the database

## get\_restaurants.py



1. Download `get_restaurants.py` and `db.json` from course web page
2. Edit `db.json` with your credentials
3. Run:  

```
python get_restaurants.py restaurant_name <localhost|sunapee>
```

  - Replace `restaurant_name` with your own choice (e.g., Nobu or 'Rosa Mexicano')
  - Provide either `localhost` or `sunapee` (`localhost` default)

```
python get_restaurants.py nobu sunapee
```

```
python get_restaurants.py 'rosa mexicano' localhost
```

Fetches data about one restaurant

**Problem: business logic is hidden inside python code**  
**We can do better!**

# Agenda

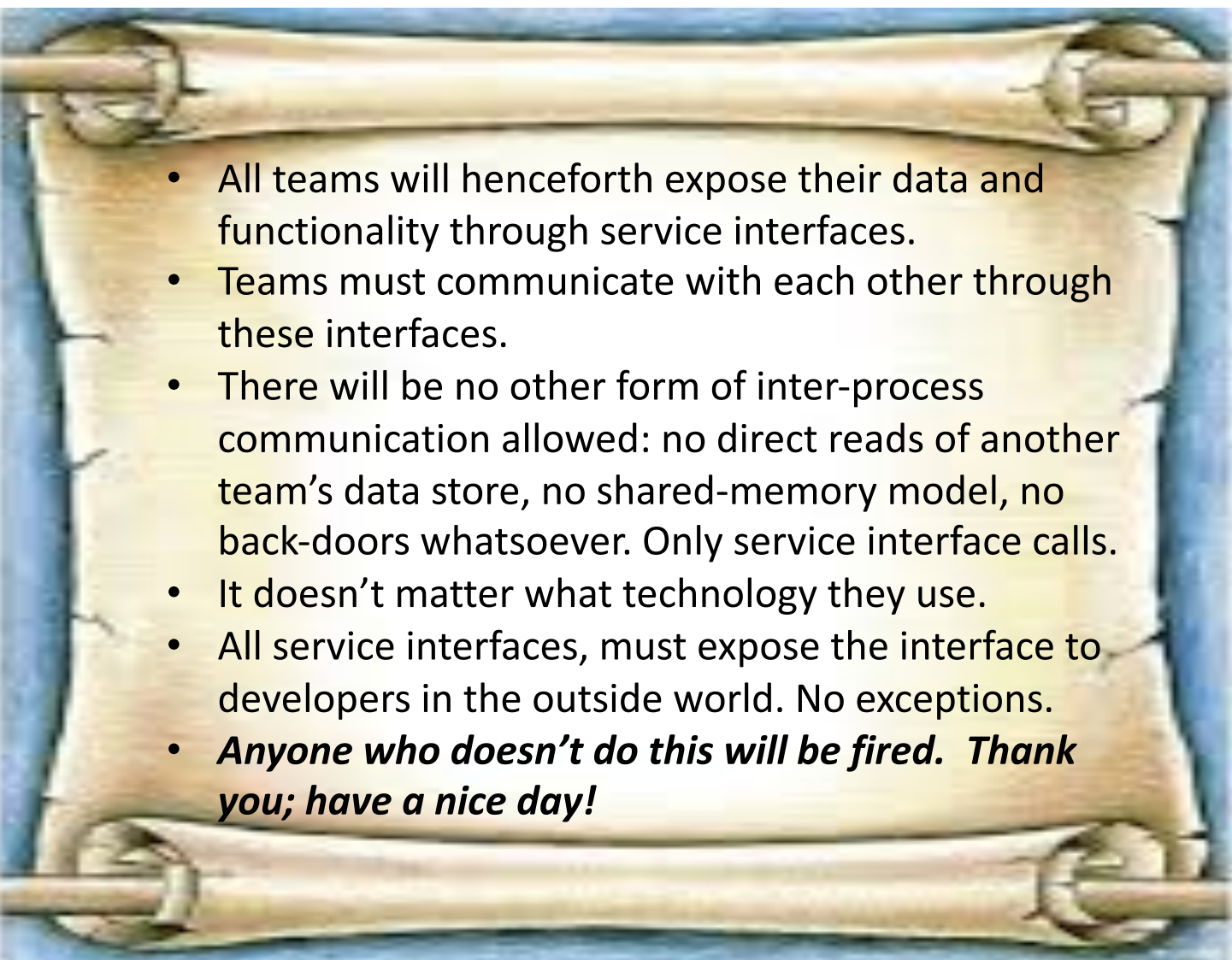
1. Direct database access



2. Web APIs

3. Node.js

# A (possibly apocryphal) letter from Jeff Bezos to Amazon developers

- 
- A scroll with text, representing a letter from Jeff Bezos to Amazon developers. The scroll is unrolled and shows a list of instructions.
- All teams will henceforth expose their data and functionality through service interfaces.
  - Teams must communicate with each other through these interfaces.
  - There will be no other form of inter-process communication allowed: no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. Only service interface calls.
  - It doesn't matter what technology they use.
  - All service interfaces, must expose the interface to developers in the outside world. No exceptions.
  - ***Anyone who doesn't do this will be fired. Thank you; have a nice day!***

**In short: don't do what we just did with Python!**

**Create an API instead**

# RESTful APIs rely on four HTTP “verbs” to implement CRUD operations

**Client side**

**Server side**



**Smart phone apps**



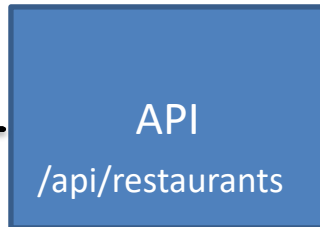
**Web browser**

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						

Parts Orders Data Entry

Item	<input type="text" value="Insulator"/>
Location	<input type="text" value="Store1"/>
Qty	<input type="text" value="100"/>
Price	<input type="text" value="\$4.00"/>
Total	<input type="text" value="\$400.00"/>

**“Thick client” apps**



**Clients make RESTful calls over network to API listening on server**

# RESTful APIs rely on four HTTP “verbs” to implement CRUD operations

## Client side



Smart phone apps



Web browser

	A	B	C	D	E	F
1						
2						
3						
4						
5	Item			Insulator		
6	Location			Store1		
7						
8	Qty			100		
9						
10	Price			\$4.00		
11						
12	Total			\$400.00		
13						
14						
15						
16						

Add to Database View Database

“Thick client” apps

## Server side

API  
/api/restaurants

Clients make RESTful calls over network to API listening on server

POST  
/api/restaurants

**Create: use POST**  
Include params to create a new restaurant

GET  
/api/restaurants  
/api/restaurants/:id

**Read: use GET**  
If no id passed in URL, get all restaurants, otherwise get data for RestaurantID = :id

PUT  
/api/restaurants/:id

**Update: use PUT**  
Update restaurant with RestaurantID = :id

DELETE  
/api/restaurants/:id

**Delete: use DELETE**  
Delete restaurant with RestaurantID = :id

**By convention HTTP verb tells API what CRUD operation to perform**

**Calls are stateless (all information needed is provided in each call)**



# RESTful APIs rely on four HTTP “verbs” to implement CRUD operations

## Client side



Smart phone apps



Web browser

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						

Parts Orders Data Entry

Item:

Location:

Qty:

Price:

Total:

“Thick client” apps

## Server side

API  
/api/restaurants

Clients make RESTful calls over network to API listening on server

POST  
/api/restaurants

**Create: use POST**  
Include params to create a new restaurant

GET  
/api/restaurants  
/api/restaurants/:id

**Read: use GET**  
If no id passed in URL, get all restaurants, otherwise get data for RestaurantID = :id

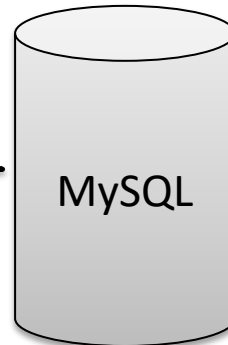
PUT  
/api/restaurants/:id

**Update: use PUT**  
Update restaurant with RestaurantID = :id

DELETE  
/api/restaurants/:id

**Delete: use DELETE**  
Delete restaurant with RestaurantID = :id

APIs access database and return results to client side



APIs access database as user with minimal required rights



# DOGFOODING

*The Religion of Successful Products*

# Agenda

1. Direct database access

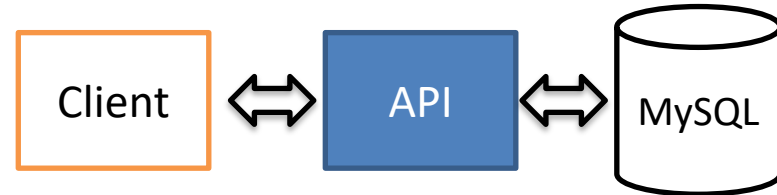
2. Web APIs



3. Node.js

# Server-side API written in JavaScript running on Node.js

## Node.js example



1. Install Node.js on your machine:  
<https://nodejs.org/en/download/>
2. Create a folder for this project (e.g., Documents/cs61/nodeExample)
3. Download example server-side code (api.js, config.js, package.json) from course web page for today into that folder
4. Edit the config.js with your database credentials: replace username and password
5. From command line:
  - Change directory into folder: `cd Documents/cs61/nodeExample`
  - Get all needed libraries (listed in package.json): `npm install`
  - Start server running: `nodemon api.js`
  - Start browser and enter URL: `localhost:3000/api/restaurants`
  - Should see a list of 10 restaurants in JSON format
  - Try connecting to sunapee: `nodemon api.js sunapee`
6. Download Postman ([www.postman.com](http://www.postman.com)) to try other verbs

# Client-side code written in Python calls server-side API written in JavaScript

## Python example

Client side:

Call web API from Python:

```
python call_api.py
```

Data normally returned from API in JSON format

Client-side Python code would then process the JSON data

