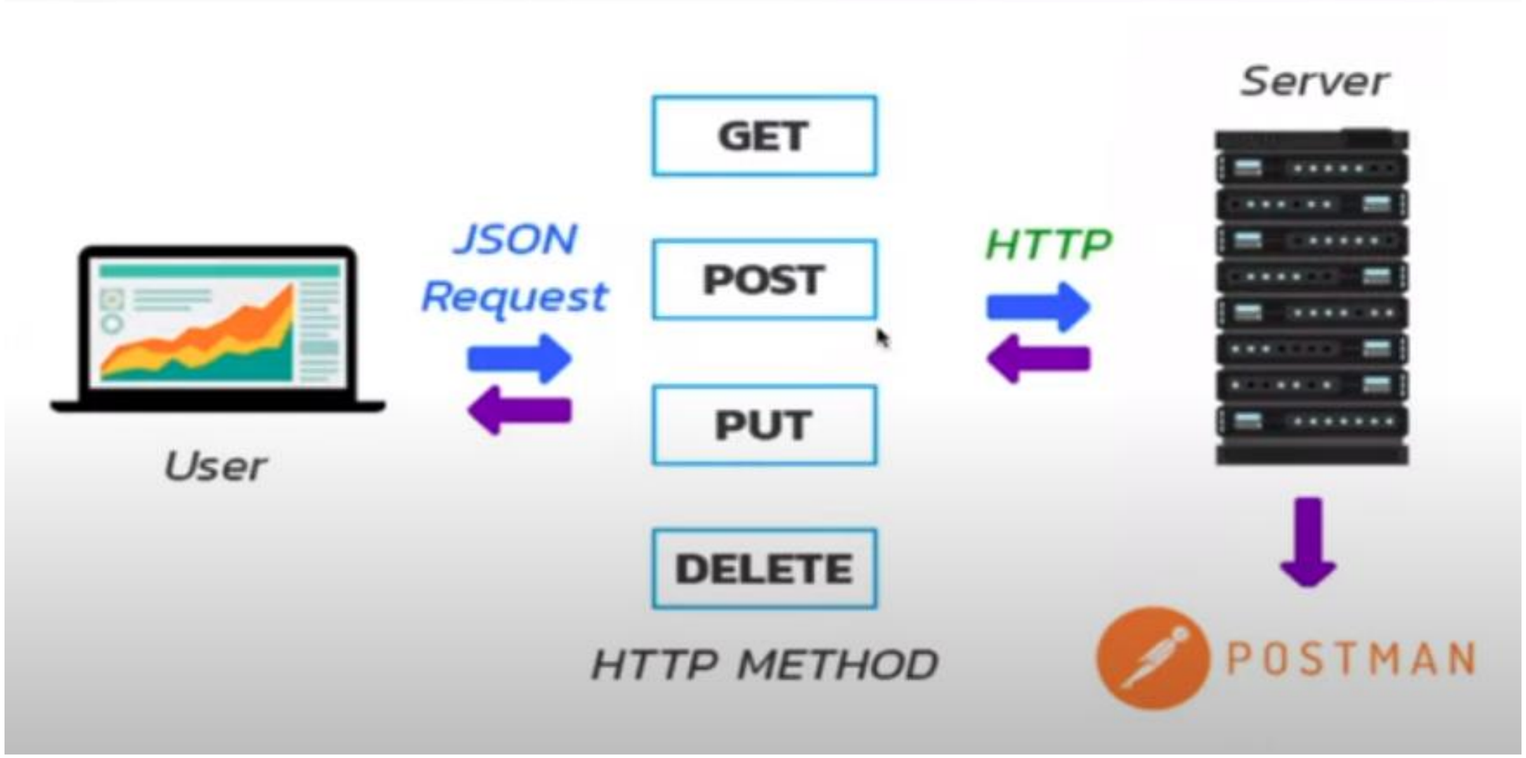
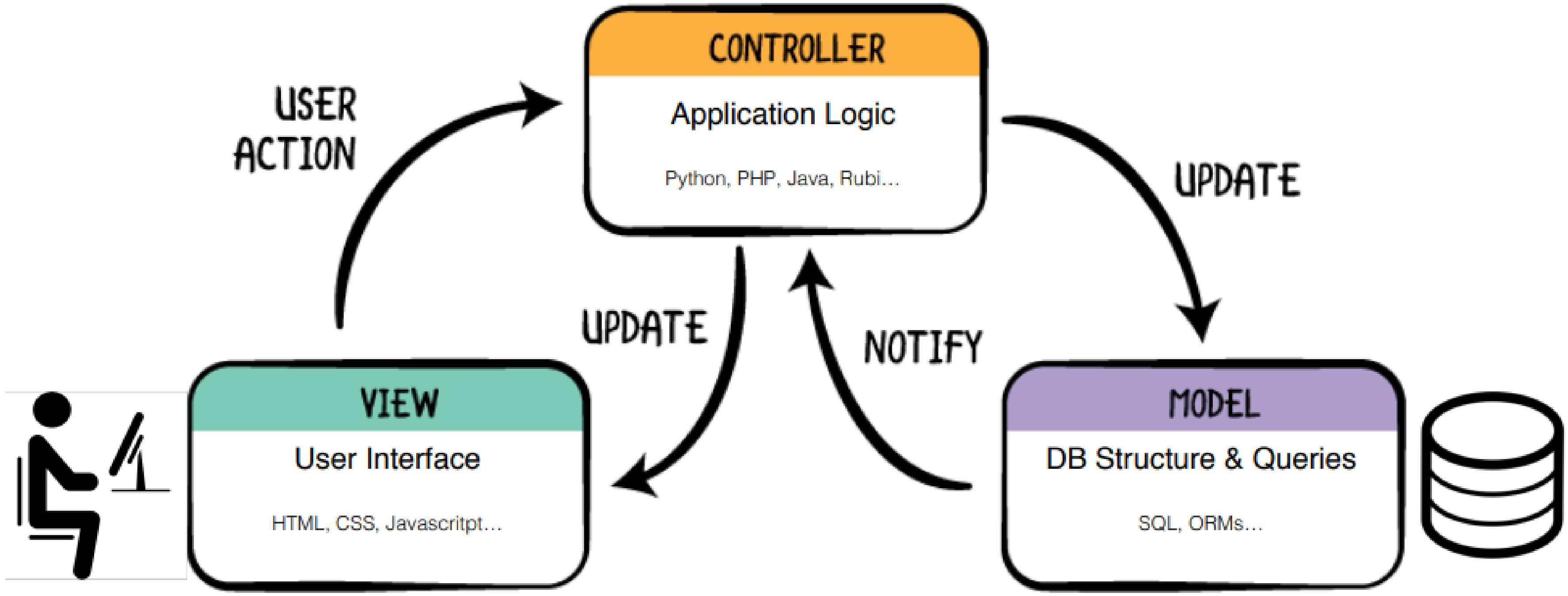
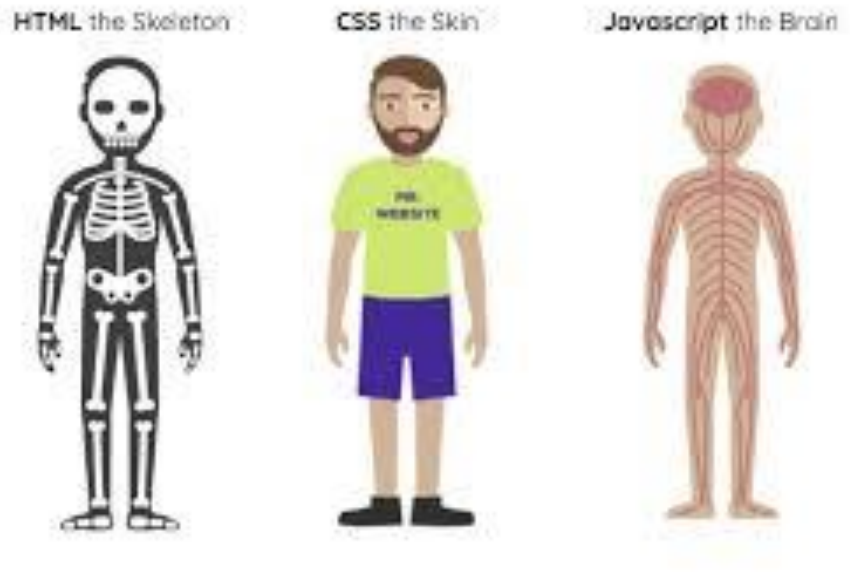
PROCESS


User
JSON Request
GET
POST
PUT
DELETE
HTTP METHOD
HTTP
Server
POSTMAN

MVC: Model View Controller




HTML the Skeleton
CSS the Skin
MR. WEBSITE
Javascript the Brain

python

Flask

web development,
one drop at a time

Project Structure

```
hangman
 |── hangman.db
 |── hangman.py
 |── static
 |     |── bootstrap.min.css
 |     |── bootstrap.min.js
 |     |── jquery.min.js
 |     |── main.css
 |     |── main.js
 |── templates
 |     |── home.html
 |     |── index.html
 |     |── play.html
 |── words.txt
```

Project Structure: The View

```
hangman
 |── hangman.db
 |── hangman.py
 |── static
 |     |── bootstrap.min.css
 |     |── bootstrap.min.js
 |     |── jquery.min.js
 |     |── main.css
 |     |── main.js
 |── templates
 |     |── home.html
 |     |── index.html
 |     |── play.html
 |── words.txt
```

Project Structure: The Model

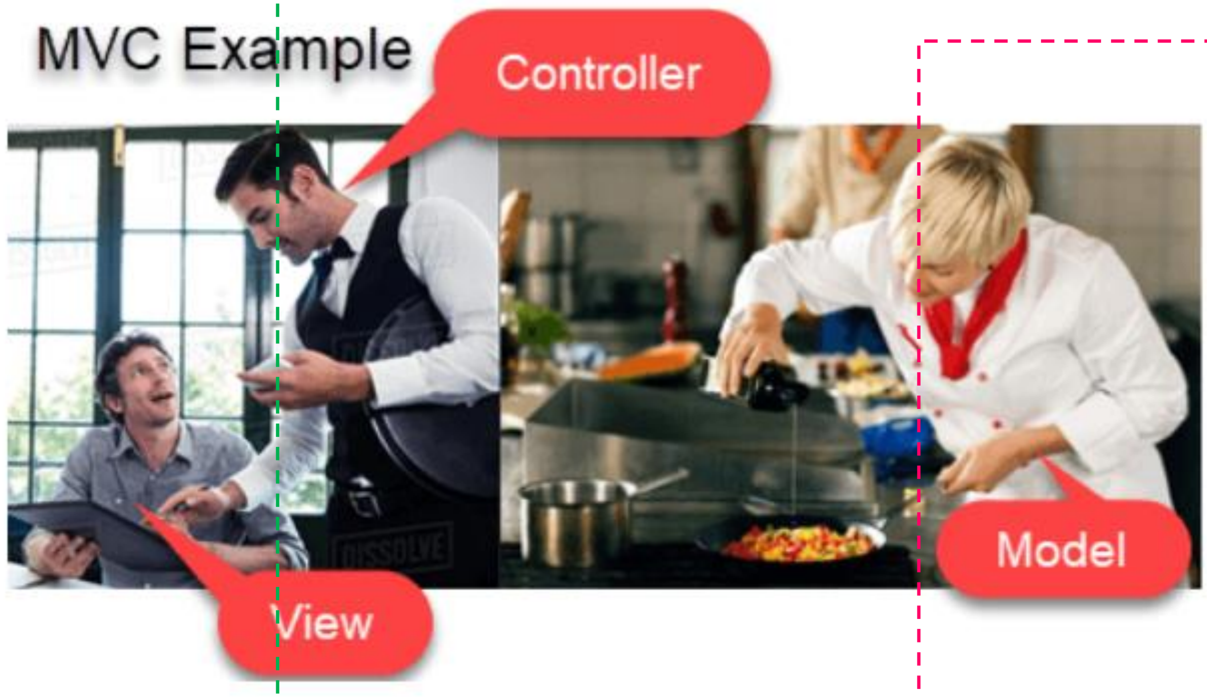
```
hangman
 |── hangman.db
 |── hangman.py
 |── static
 |     |── bootstrap.min.css
 |     |── bootstrap.min.js
 |     |── jquery.min.js
 |     |── main.css
 |     |── main.js
 |── templates
 |     |── home.html
 |     |── index.html
 |     |── play.html
 |── words.txt
```

Project Structure: The Controller

```
hangman
 |— hangman.db
 |— hangman.py
 |— static
 |      |— bootstrap.min.css
 |      |— bootstrap.min.js
 |      |— jquery.min.js
 |      |— main.css
 |      |— main.js
 |— templates
 |      |— home.html
 |      |— index.html
 |      |— play.html
 |— words.txt
```

MVC Examples

TODAY



Refer: https://www.guru99.com/mvc-tutorial.html

MVC Examples

Refer: https://www.guru99.com/mvc-tutorial.html

MVC Examples

- Let’s assume you go to a restaurant. You will not go to the kitchen and prepare food which you can surely do at your home. Instead, you go there and wait for the waiter to come on.
- Now the waiter comes to you, and you order the food. The waiter doesn’t know who you are and what you want he just written down the detail of your food order.

MVC Examples

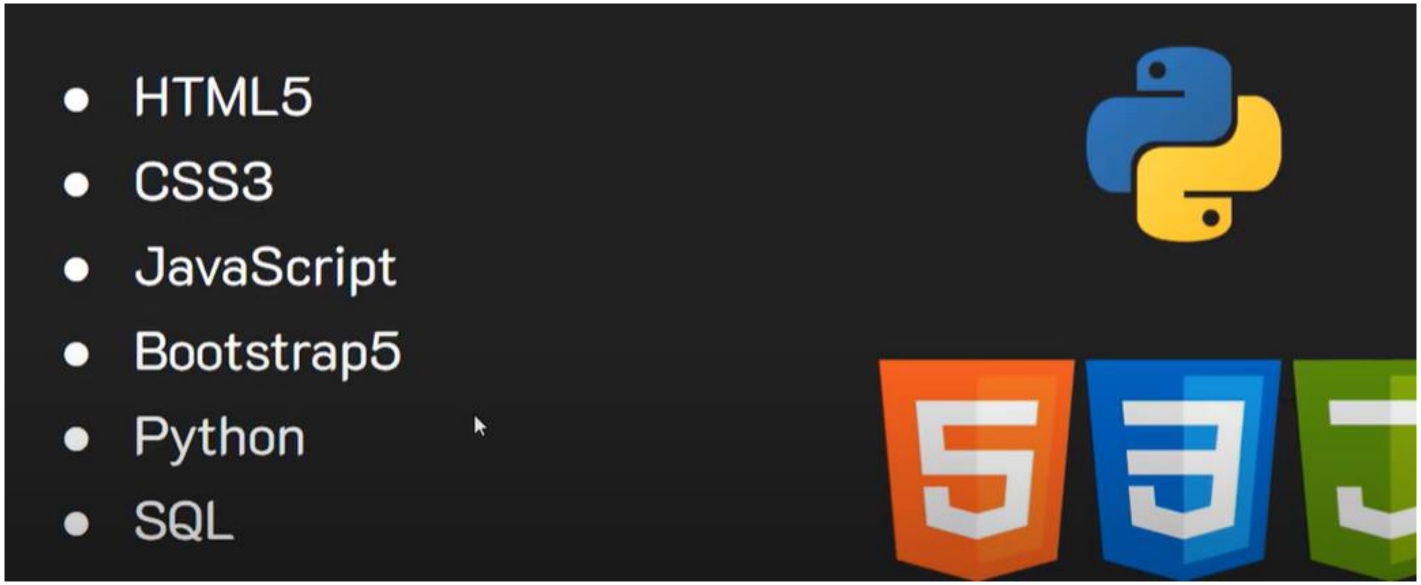
- Then, the waiter moves to the kitchen. In the kitchen, waiter does not prepare your food.
- The cook prepares your food. The waiter is given your order to him along with your table number.

MVC Examples

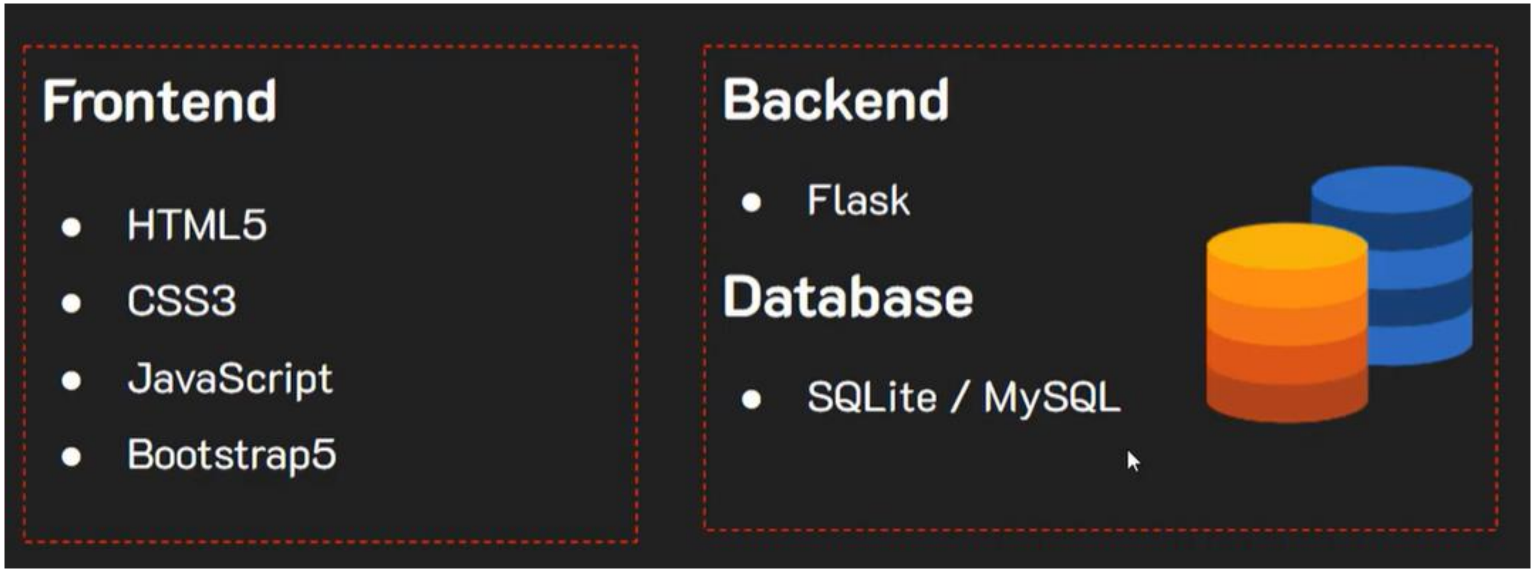
- Cook then prepared food for you. He uses ingredients to cooks the food. Let’s assume that your order a vegetable sandwich. Then he needs bread, tomato, potato, capsicum, onion, bit, cheese, etc. which he sources from the refrigerator

MVC Examples

- Cook final hand over the food to the waiter. Now it is the job of the waiter to moves this food outside the kitchen.
- Now waiter knows which food you have ordered and how they are served.



- HTML5
- CSS3
- JavaScript
- Bootstrap5
- Python
- SQL



Frontend

- HTML5
- CSS3
- JavaScript
- Bootstrap5

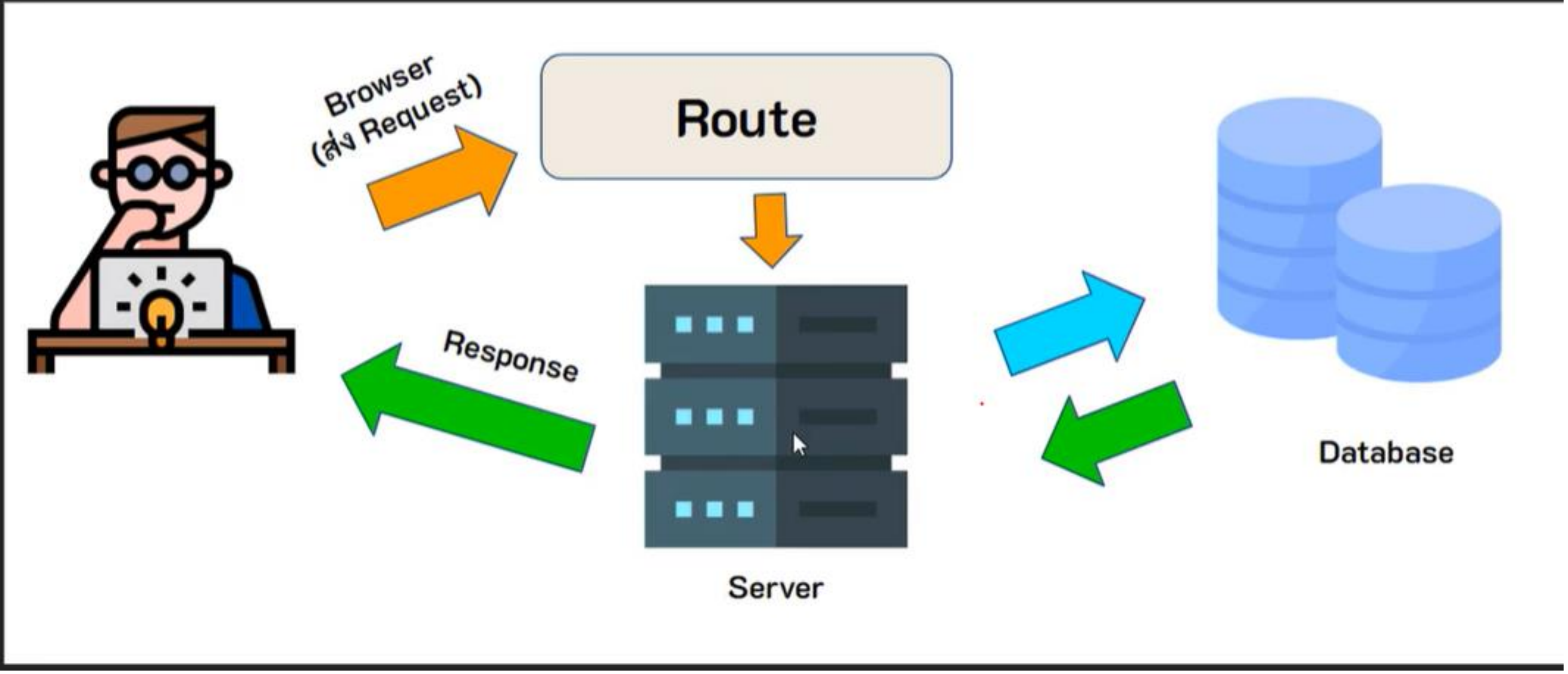
Backend

- Flask

Database

- SQLite / MySQL

Routing



Google

400. That's an error.

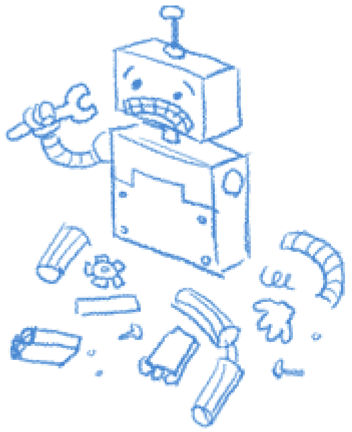
Error: invalid_request

Missing required parameter: client_id

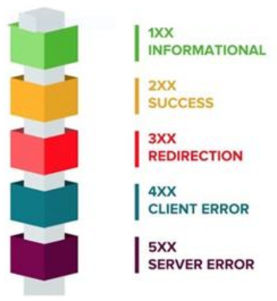
Learn more

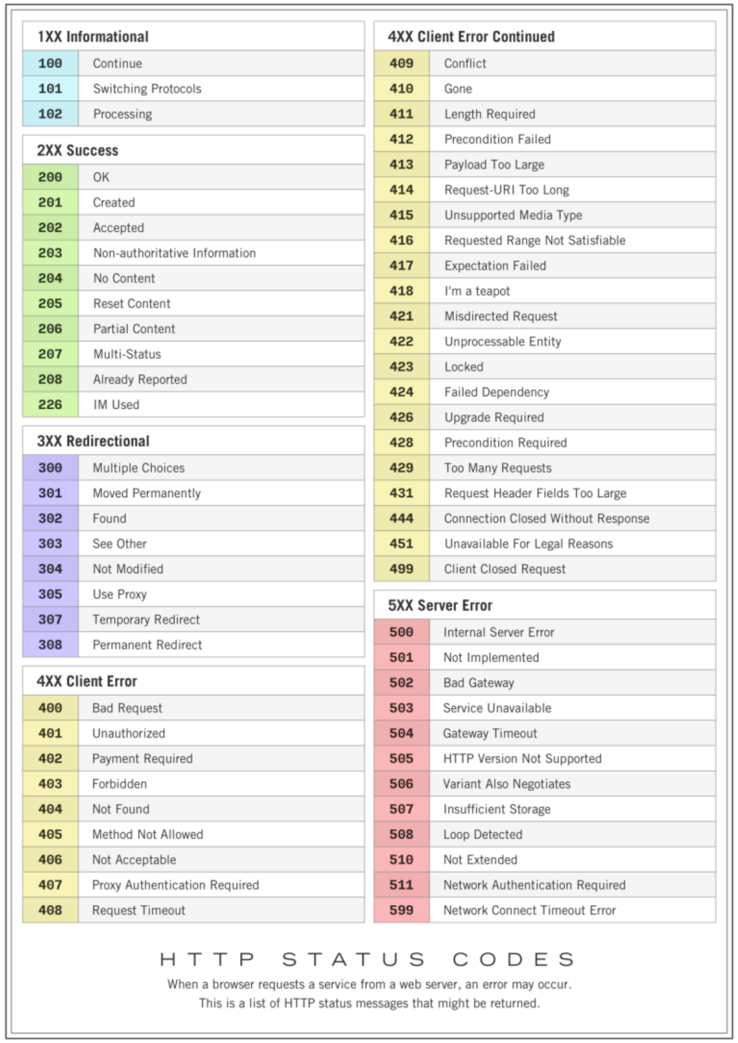
‣ Request Details

That's all we know.



HTTP Status Codes





1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

HTTP STATUS CODES

When a browser requests a service from a web server, an error may occur.
This is a list of HTTP status messages that might be returned.

SQL

- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes
  - Following lectures...
- Data Manipulation Language (DML)
  - Query one or more tables – discussed next !
  - Insert/delete/modify tuples in tables

Tables in SQL

Table name

Attribute names

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Tuples or rows

Tables Explained

- The *schema* of a table is the table name and its attributes:

Product(PName, Price, Category, Manfacturer)

- A *key* is an attribute whose values are unique; we underline a key

Product(<u>PName</u>, Price, Category, Manfacturer)

Data Types in SQL

- Atomic types:
  - Characters: CHAR(20), VARCHAR(50)
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, …
- Every attribute must have an atomic type
  - Hence tables are flat
  - Why ?

Tables Explained

- A tuple = a record
  - Restriction: all attributes are of atomic type

- A table = a set of tuples
  - Like a list…
  - …but it is unorderd:
    no **first()**, no **next()**, no **last()**.

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   *
FROM     Product
WHERE    category='Gadgets'
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

"selection"

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

“selection” and “projection”

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Notation

Input Schema

Product(PName, Price, Category, Manfacturer)

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

Answer(PName, Price, Manfacturer)

Output Schema

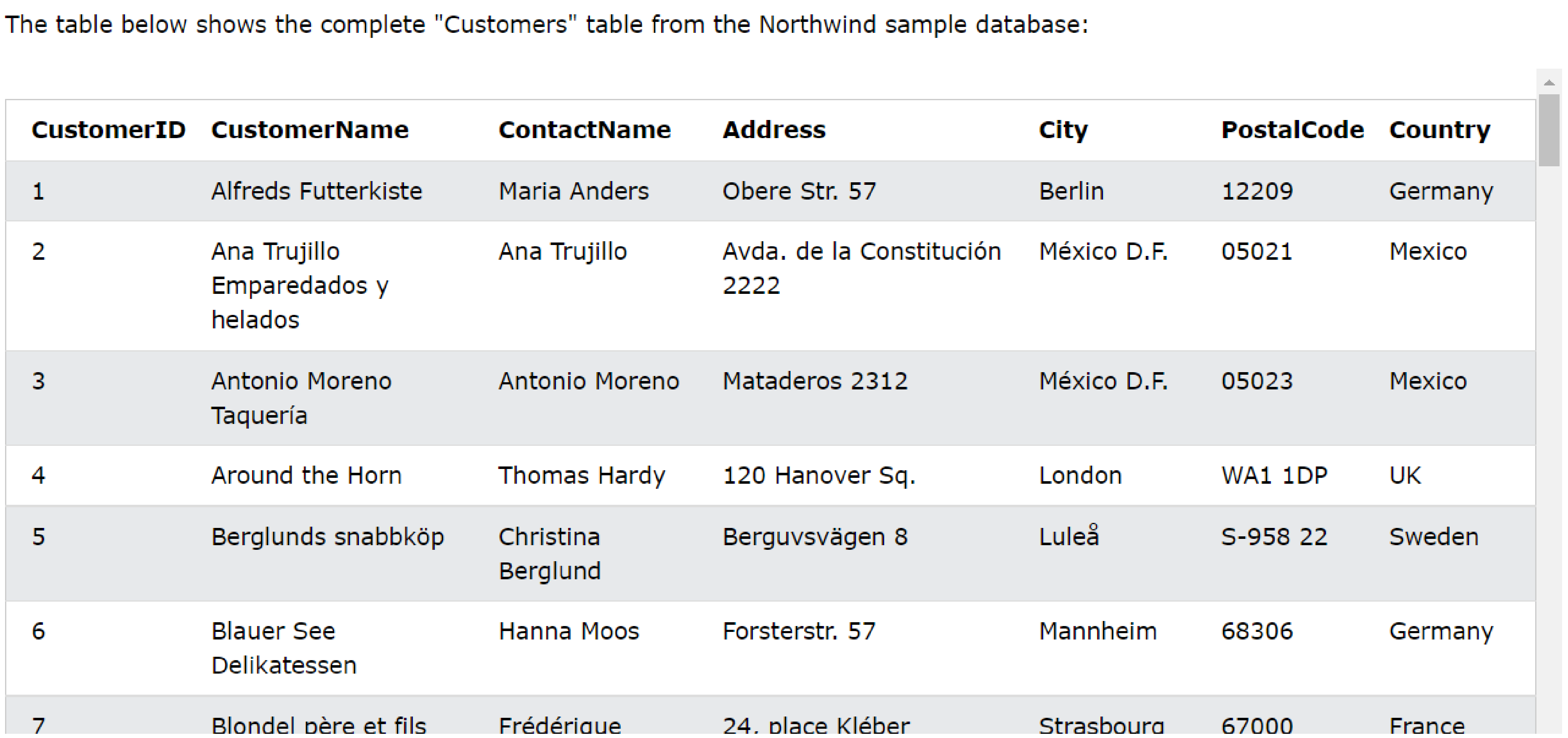
The **LIKE** operator

```
SELECT   *
FROM     Products
WHERE    PName LIKE '%gizmo%'
```

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

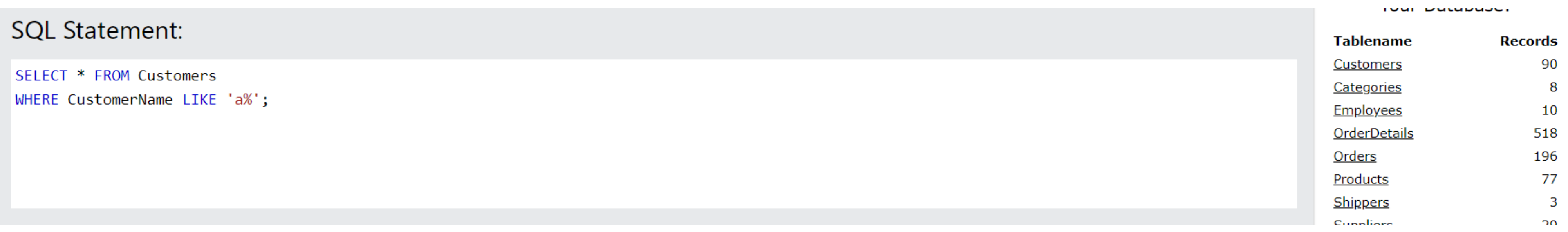
Here are some examples showing different `LIKE` operators with '%' and '_' wildcards:

LIKE Operator	**Description**
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"



The table below shows the complete "Customers" table from the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique	24, place Kléber	Strasbourg	67000	France



SQL Statement:

```sql
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Your Database:

Tablename	Records
Customers	90
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Eliminating Duplicates

```
SELECT   DISTINCT category
FROM     Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT   category
FROM     Product
```

Category
Gadgets
Gadgets
Photography
Household

Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category='gizmo' AND price > 50
ORDER BY price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

$\Rightarrow$?

```
SELECT   Category
FROM     Product
ORDER BY PName
```

$\Rightarrow$?

```
SELECT   DISTINCT category
FROM     Product
ORDER BY PName
```

$\Rightarrow$?

Keys and Foreign Keys

Company

Key

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Foreign key

Joins

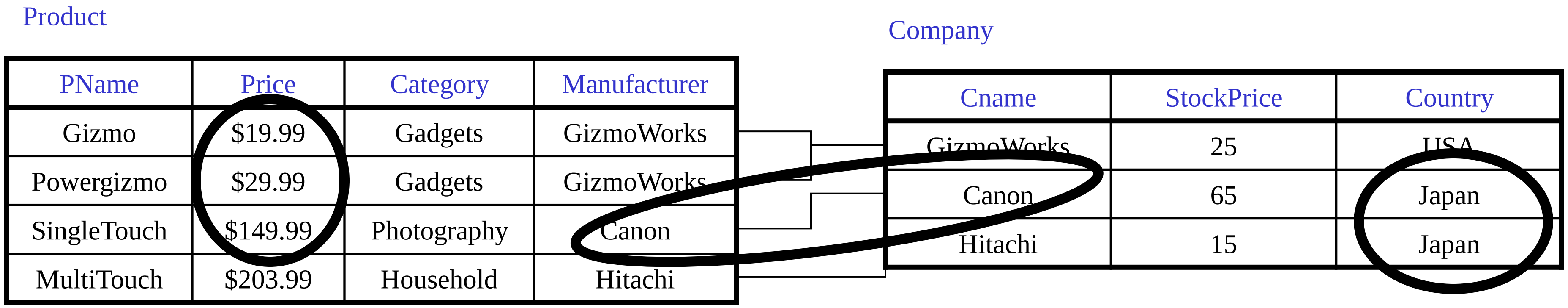
Product (<u>pname</u>, price, category, manufacturer)
Company (<u>cname</u>, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

Join between Product and Company

Joins



Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country=‘Japan’
         AND Price <= 200
```

PName	Price
SingleTouch	$149.99

Tuple Variables

Person(<u>pname</u>, address, worksfor)
Company(<u>cname</u>, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address ?

⇒

```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company
WHERE    Person.worksfor = Company.cname
```

⇒

```
SELECT   DISTINCT x.pname, y.address
FROM     Person AS x, Company AS y
WHERE    x.worksfor = y.cname
```

Subqueries Returning Relations

Company(<u>name</u>, city)
Product(<u>pname</u>, maker)
Purchase(<u>id</u>, product, buyer)

Return cities where one can find companies that manufacture products bought by Joe Blow

```
SELECT  Company.city
FROM    Company
WHERE  Company.name  IN
              (SELECT Product.maker
                FROM   Purchase , Product
                WHERE Product.pname=Purchase.product
                     AND Purchase .buyer = ‘Joe Blow‘);
```

Subqueries Returning Relations

Is it equivalent to this ?

```
SELECT  Company.city
FROM     Company, Product, Purchase
WHERE   Company.name= Product.maker
          AND  Product.pname  = Purchase.product
          AND  Purchase.buyer = 'Joe Blow'
```

Beware of duplicates !

Removing Duplicates

```
SELECT DISTINCT Company.city
FROM     Company
WHERE  Company.name  IN
                (SELECT Product.maker
                  FROM   Purchase , Product
                  WHERE Product.pname=Purchase.product
                       AND Purchase .buyer = ‘Joe Blow‘);
```

```
SELECT DISTINCT Company.city
FROM      Company, Product, Purchase
WHERE   Company.name= Product.maker
        AND  Product.pname  = Purchase.product
        AND  Purchase.buyer = ‘Joe Blow’
```

Now they are equivalent

Aggregation

```
SELECT  avg(price)
FROM     Product
WHERE   maker="Toyota"
```

```
SELECT  count(*)
FROM     Product
WHERE   year > 1995
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(category)
FROM    Product
WHERE   year > 1995
```

same as Count(*)

We probably want:

```
SELECT  Count(DISTINCT category)
FROM    Product
WHERE   year > 1995
```

More Examples

Purchase(product, date, price, quantity)

```
SELECT  Sum(price * quantity)
FROM     Purchase
```

```
SELECT  Sum(price * quantity)
FROM     Purchase
WHERE   product = ‘bagel’
```

What do they mean ?

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM     Purchase
WHERE   product = ‘bagel’
```

$\Rightarrow$

50 (= 20+30)

Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.

```
SELECT      product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE       date > ‘10/1/2005’
GROUP BY    product
```

Let’s see what this means…

Grouping and Aggregation

1. Compute the FROM and WHERE clauses.
2. Group by the attributes in the GROUPBY
3. Compute the SELECT clause: grouped attributes and aggregates.

1&2. FROM-WHERE-GROUPBY

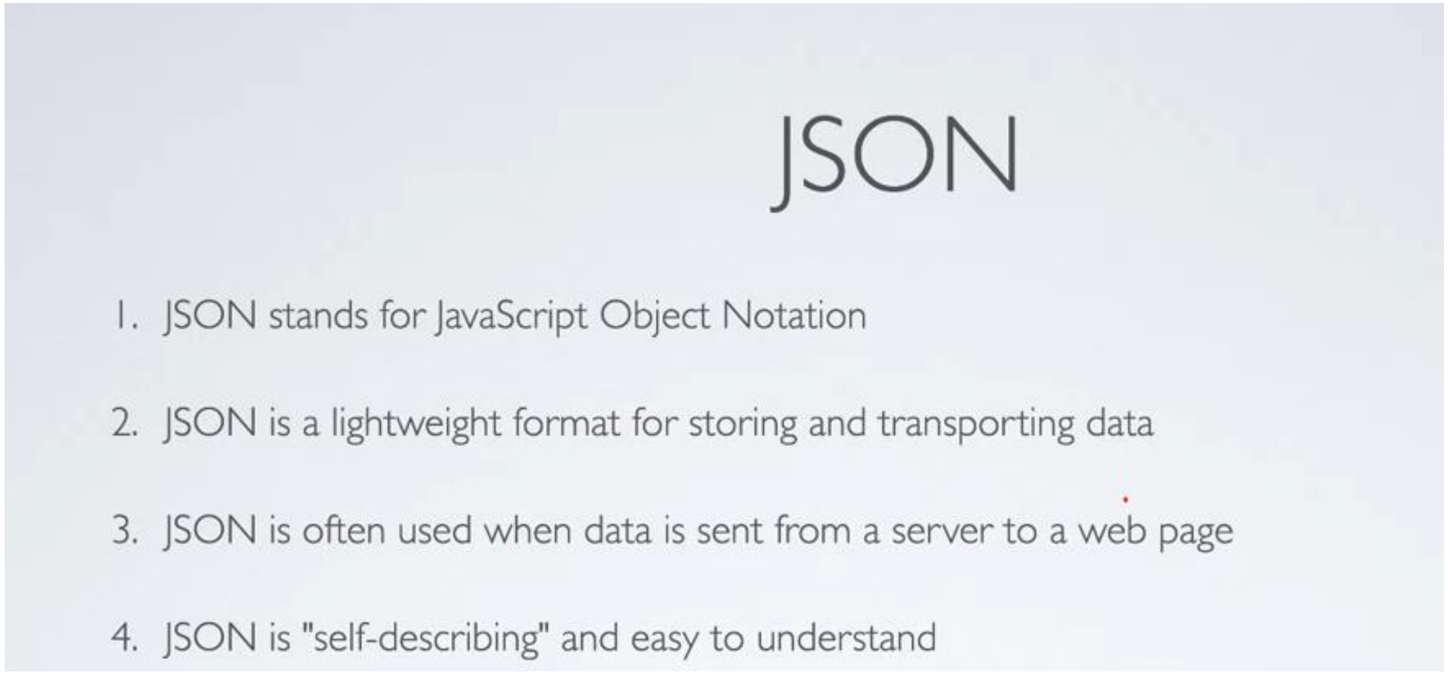
Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

3. SELECT

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

Product	TotalSales
Bagel	50
Banana	15

```
SELECT      product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE       date > '10/1/2005'
GROUP BY    product
```

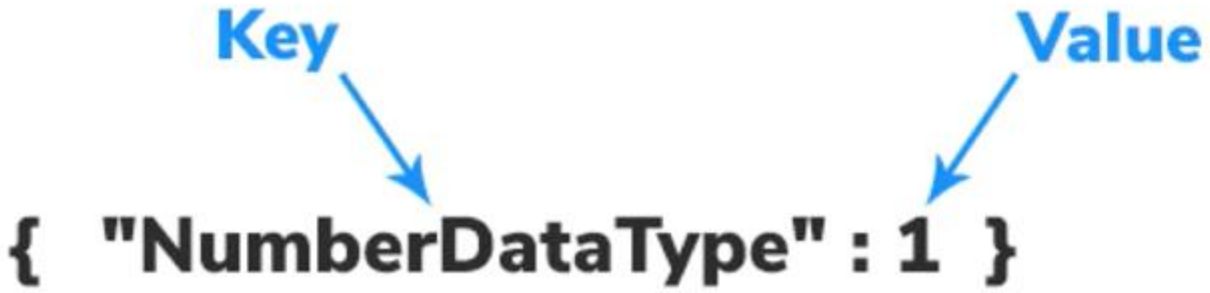


JSON

1. JSON stands for JavaScript Object Notation
2. JSON is a lightweight format for storing and transporting data
3. JSON is often used when data is sent from a server to a web page
4. JSON is "self-describing" and easy to understand

JSON FORMATTING

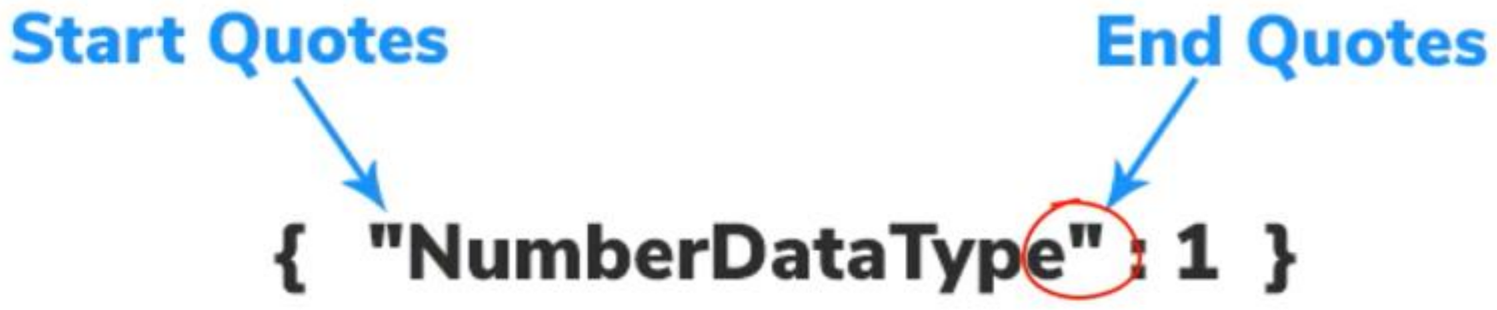
Rule # 1. Key/Identifier & Value



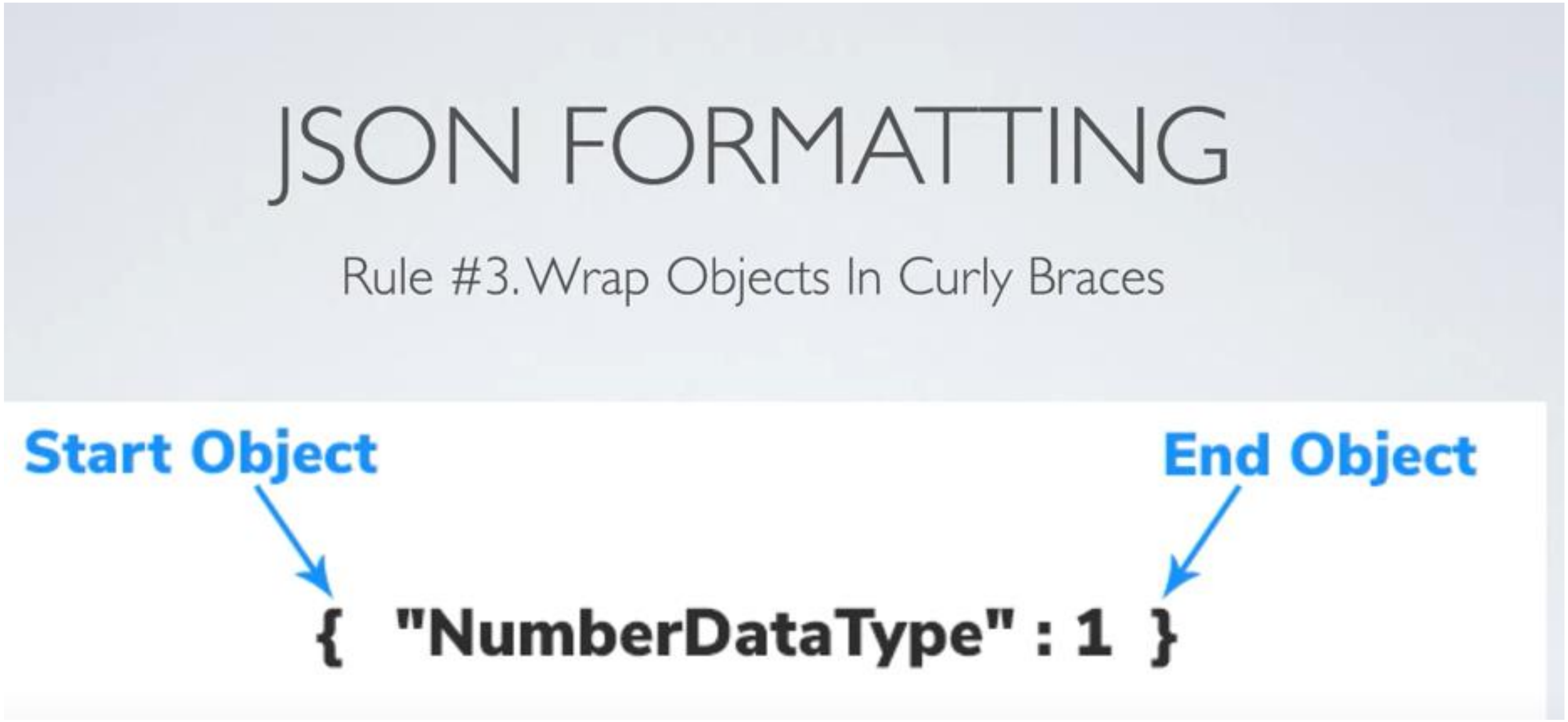
```
{  "NumberDataType" : 1  }
```

JSON FORMATTING

Rule #2. Key/Identifier Double Quoted



```
{  "NumberDataType" : 1  }
```



JSON FORMATTING

Rule #3. Wrap Objects In Curly Braces

Start Object
End Object

```
{  "NumberDataType" : 1  }
```