Adv Web Dev Arch
Lecture 7

CRUD and Database manipulation

Amir Amintabar, PhD

AJAX Calls

CRUD and Database Manipulation

for APIs development

SQL Database related knowledge needed for the development of API servers

Outline (Using Relational DB for Building an API server)

- 0- Review
- Review of SQL stametments
- 1- What is a Database?
- 2- Few definitions
- 3- Design a DB by example
- 4- Creating 1:M by example
- 5- Referential integrity **constraint**
- **6-** Learn SQL by Example!
  - Query practice examples using the online hosted NorthWind DB
  - https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
  - Where clauses, like operator
- 7- Sub-queries
- 8- Creating M:M by example

Review

Review

- setTimeout and order of execution (in what order numbers log?)
- Various web application architectures, …, their goal …
- Single or multiple threaded, asynchronous, none-blocking ..
- AJAX calls (Asynchronous JavaScript And XML),
- XMLHttpRequest readyState (0, 1, 2, 3, 4)
- GET, POST, the difference (data size, url, …)
- Modules in Nodejs,
- built in modules such as http, fs ..,
- installing new modules using NPM install,
- How to host a node js app,
- how to create DB user in a cPannel shared hosting accounts,
  - giving various privileges to DB user,
- restarting a node js app every time we make changes (you can install node-mon module on your local PC that automatically restarts the app every time you save changes).
- How node js connects to DB,
- how to install mySQL engine locally,
- how to use DB admin tools such as phpMyAdmin to create DB/ tables and where to run SQL statements, how to backup and restore.

```
console.log(1);
setTimeout(function () {
    console.log(2);
}, 0);
console.log(3);
```

status of the XMLHttpRequest :
0 (UNSENT): The XHR object has been created, but open() has not been called yet.
1 (OPENED): open() has been called.
2 (HEADERS_RECEIVED): send() has been called, and the headers of the response are available.
3 (LOADING): The response is being received. As data comes in, the responseText property is updated. **(3(LOADING) code be updated multiple times to this status if the server is sending a large size of response in multiple chunks)**
4 (DONE): The operation is complete, and either the request has been successfully completed (status code 2xx) or an error occurred.

What is a database?

What is a database?

- Q: So what do you think it is?
- A: A database is an organized collection of data, stored electronically in a server
- Typically a computer
- or computers distributed
- Or on RAID (redundant array of independent disks to improve robustness or speed etc)

Database example

- Lets see one in action (hosted northWind DB)
- https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- How many tables are there in that DB?
- Run a couple of queries such as
- SELECT * FROM [Customers]
- SELECT country FROM [Customers]
- SELECT distinct country FROM [Customers]
- SELECT * FROM Products, OrderDetails WHERE Products.ProductID=OrderDetails.ProductID AND Quantity>10
- SELECT country, count(CustomerID)
  FROM [Customers]
  group by country
- SELECT * FROM Customers WHERE Country='Mexico';

Database Management System (DBMS)

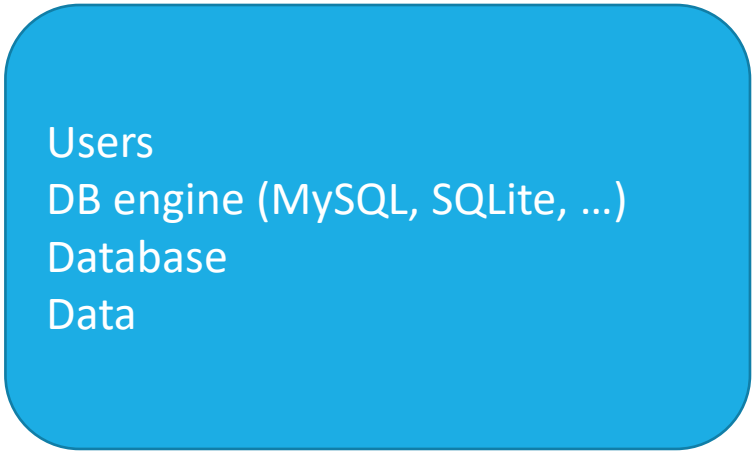
- What we just saw was a database, not Database Management System!

DBMS
DataBase Management System

2

What are DBMS?

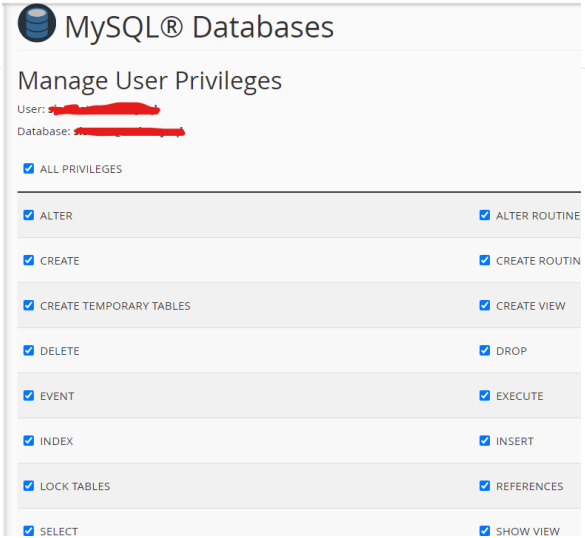
- The **DBMS** manages incoming, outgoing data, organizes it, and provides ways for the data to be modified or extracted by users or other programs



- **Massive**:
  - massive data in size! **Q**: Guess how many terra bytes of data we need to store entire students data?
- **Persistent**:
  - data in the database outlives the programs that execute on that data. Program will finish but the data remains on disk
- **Safe** (data has to stay in consistent state despite hardware failure, **malicious attacks**,
  - run critical applications such as telecommunications and banking systems, have to have guarantees that the data managed by the system
  - will stay in a consistent state, it won't be lost or overwritten when there are failures, and there can be hardware failures.
  - There can be **software failures**.
  - Even simple power outages. You don't want your bank balance to change because the power went out at your bank branch.
  - And of course there are the problem of malicious users that may try to corrupt data.

DBMSs are also

- **User management**
  - You can create DB users and give them privileges



- **Multi-user**
  - Concurrency control (e.g. at the same time one person can write)
- **Convenient**
  - High level query languages, declarative language (you will see why)
- **Efficient**
  - Performance, No duplication, fast access
- **Reliable**
  - 99.999 uptime! (even missing a single bank transaction could create an issue)

Relational DBMS could be categorized as

- **Relational** DBMs (RDBM) using SQL language
- **Non-relational** DBMs (also called NoSQL DBMS)

- For our APIs, we only focus on Relational DBMs, in this course

- RDBMS language: SQL
  - SQL: Structured Query Language
- Examples of RDBMS:
  - MySQL (focus on this course for our CRUD on APIs)
  - Sqlite (file base, light weigh and runs in browsers such as https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
  - Other examples: Oracle, PostgreSQL, SQL server,
-

Design a DB by example

How would you keep records of patients' visits to a clinic?

- Try to keep them in sticky notes? Notebooks or Spreadsheet ?
- For two patients for their clinic visits, create a spreadsheet to keep their records

Patient name	Age	Clinic Visit
Sarah Brown	97	01/01/2001
John Smith	56	08/08/2018

- Now, what if John visited the clinic on another date, 08/01/2019 ? Which one do you keep in the sheet. Or would you add the new one just next to previous one?

Patient name	Age	Clinic Visit
Sarah Brown	97	01/01/2001
John Smith	56	08/08/2018, 09/09/2019, …

Storing multiple values in same column is not a good practice!
Can you guess why?
Aside from reasons you listed, it also violates **1st Normal Form**

- What if there were no more room in that Column to add new visit dates?

Exercise 2 cont.

- … what if there were no more room in that Column to add new visit dates?
- What if we add multiple new columns to record additional visits

Patient name	Age	Clinic Visit 1	Clinic Visit 2	Clinic Visit 3
Sarah Brown	97	01/01/2001		
John Smith	56	08/08/2018	09/09/2019	02/02/2020

- Sarah is already 97, what if Sarah will never visit the clinic again?

Exercise 2 cont.

Patient name	Age	Clinic Visit 1	Clinic Visit 2
Sarah Brown	97	01/01/2001	
John Smith	56	08/08/2018	09/09/2019

Patient name	Age	Clinic Visit
Sarah Brown	97	01/01/2001
John Smith	56	08/08/2018
John Smith	56	09/09/2019

- Does this work ? Can it be implemented in an spreadsheet?
- Do you notice anything wrong with this approach?
  - Exactly! We have to repeat the name and age of patients everytime!
- Can you suggest a better format for our spreadsheet?

Patients visits record-keeping problem with better answer (RDB)

- Or we can simply move to **relational** databases
- Have multiple tables that in some way relate to one another

Unique to each patient

Patient

Patient_ID	Patient name	Age
1	Sarah Brown	97
2	John Smith	56

Visit

Patient_ID	Clinic Visit
1	01/01/2001
2	08/08/2018
2	09/09/2019

- **Patient table**: one row for each patient
- **Visit table**: one row for each visit. A patient might visit multiple times, thus multiple rows in the visit table would be related to that patient

- **Q:** We said tables in same relation DB are in some ways related. How these two are related?
- Because we have the column Pationt_ID in both tables, we can **relate** them based on that value. This is the **relational** part

Patients visits record-keeping Relational DB

- Now lets give each visit in the Visit table a unique number too

Patient

Patient_ID	Patient name	Age
1	Sarah Brown	97
2	John Smith	56

Unique to each visit

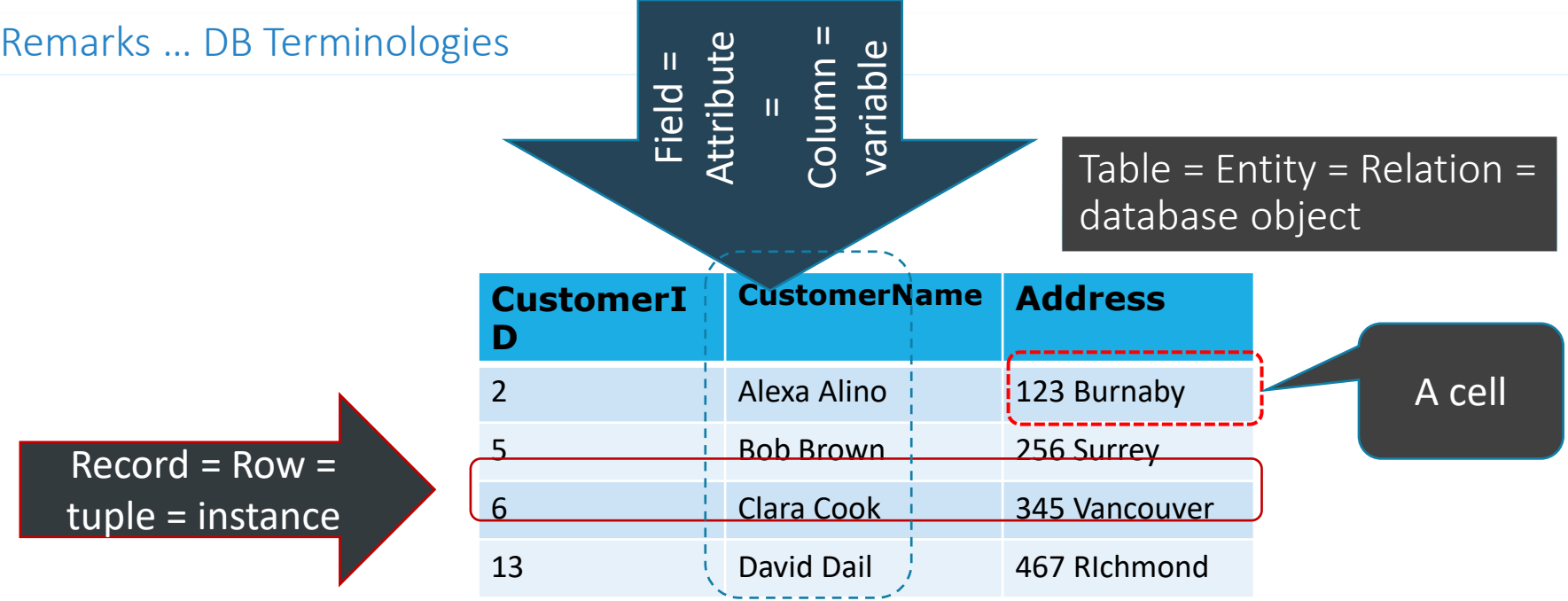
Visit

Visit_ID	Patient_ID	Clinic Visit
1	1	01/01/2001
2	2	08/08/2018
3	2	09/09/2019

- This setup allows me to answer questions like: Who visited the office yesterday
- I can simply match the visit record with the patient names with the help of Patient_ID which exists in both tables.

- Patient_ID in the Patient table is unique and is called **Primary key**
- **Q:** identify primary key in the Visit table?
- A: Visit_ID
- Patient_ID in the Visit table is called **Foreign key**. A primary key of another table presents in this table
- Q: Did you notice another issue with our database schema? Are you happy with the way we store the Age? Any potential problem you foresee?

Remarks … DB Terminologies



CustomerID	CustomerName	Address
2	Alexa Alino	123 Burnaby
5	Bob Brown	256 Surrey
6	Clara Cook	345 Vancouver
13	David Dail	467 RIchmond

- A **table** has rows and columns, where rows represents **records** and columns represent the **attributes**.
- Every **table** is broken up into smaller entities called **fields**. The fields in the Customers table consist of CustomerID, CustomerName and Address

Remarks …

- A **field** is a column in a table that is designed to maintain specific information about every record in the table.
- A **record**, also called a **row** or **tuple,** is each individual entry that exists in a table
- Data values are stored in a **Cell**
- "Table Name" is unique (in a database). You cannot have two tables both named "Patient"

One to Many relationship

- One patient can visit multiple times.
- i.e A single patient (single row) in the Patient table can relate to multiple rows of the Visit table
- That means there is a one to Many **(1:M)** Relationship from Patient to Visit table

Patient

Patient_ID	Patient name	Age
1	Sarah Brown	97
2	John Smith	56

Visit

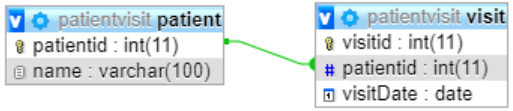
Visit_ID	Patient_ID	Clinic Visit
1	1	01/01/2001
2	2	08/08/2018
3	2	09/09/2019

- Similarly one to one relationship means a patient could visit the office "only" once!

Creating 1:M by example

Try this at phpMyAdmin

```
CREATE TABLE `patient` (
  `patientid` int(11) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
  PRIMARY KEY(patientid)
);

CREATE TABLE `visit` (
  `visitid` int(11) NOT NULL,
  `patientid` int(11) NOT NULL,
  `visitDate` date NOT NULL,
  PRIMARY KEY(visitid),
  CONSTRAINT fk_has_patient FOREIGN KEY(patientid)
          REFERENCES patient(patientid)
);
```




```
INSERT INTO patient VALUES
(1, 'Sara Brown'),
(2, 'John Smith'),
(3, 'Jack Ma');

INSERT INTO visit VALUES
(1, 1, '2002-01-01'),
(2, 2, '2018-08-08'),
(3, 2, '2019-09-09');
```

Showing rows 0 - 8 (9 total, Query took 0.0005 seconds.)

```
SELECT * FROM patient, visit
```

Show all | Number of rows: 25 Filter r

Options

patientid	name	visitid	patientid	visitDate
1	Sara Brown	1	1	2002-01-01
2	John Smith	1	1	2002-01-01
3	Jack Ma	1	1	2002-01-01
1	Sara Brown	2	2	2018-08-08
2	John Smith	2	2	2018-08-08
3	Jack Ma	2	2	2018-08-08
1	Sara Brown	3	2	2019-09-09
2	John Smith	3	2	2019-09-09
3	Jack Ma	3	2	2019-09-09

Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)

```
SELECT * FROM patient, visit WHERE patient.patientid = visit.patientid
```

Show all | Number of rows: 25 Filter rows: Sear

+ Options

patientid	name	visitid	patientid	visitDate
1	Sara Brown	1	1	2002-01-01
2	John Smith	2	2	2018-08-08
2	John Smith	3	2	2019-09-09

Q3: Assume we remove row 1 of Patient table; what would each query generate?

- Q1)What's the difference between
- a)
- `SELECT * FROM patient, visit`

- b)
- `SELECT * FROM patient, visit`
- `where patient.patient_ID = visit.patient_ID`

- b would generate =>
  patients and their visits
  Note:Jack Ma is not there, why?
  because
  `patient.patient_ID = visit. patient_ID`
  Does not meet for Jack ma
- Q2: Which one lists patients and each of their visits (if any)?

Patient

Patient_ID	Patient name
1	Sarah Brown
2	John Smith
3	Jack Ma

Visit

Visit_ID	Patient_ID	Clinic Visit
1	1	01/01/2001
2	2	08/08/2018
3	2	09/09/2019

Showing rows 0 - 8 (9 total, Query took 0.0005 seconds.)

```
SELECT * FROM patient, visit
```

Show all | Number of rows: 25 Filter r

Options

patientid	name	visitid	patientid	visitDate
1	Sara Brown	1	1	2002-01-01
2	John Smith	1	1	2002-01-01
3	Jack Ma	1	1	2002-01-01
1	Sara Brown	2	2	2018-08-08
2	John Smith	2	2	2018-08-08
3	Jack Ma	2	2	2018-08-08
1	Sara Brown	3	2	2019-09-09
2	John Smith	3	2	2019-09-09
3	Jack Ma	3	2	2019-09-09

Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)

```
SELECT * FROM patient, visit WHERE patient.patientid = visit.patientid
```

Show all | Number of rows: 25 Filter rows: Sear

+ Options

patientid	name	visitid	patientid	visitDate
1	Sara Brown	1	1	2002-01-01
2	John Smith	2	2	2018-08-08
2	John Smith	3	2	2019-09-09

Schema vs data

- Schema versus data
  - Schema: data model, data types of columns,

    Note: all data entered in rows of the same column must all have same data type
  - data: values stored in columns

```
CREATE TABLE `patient` (
  `patientid` int(11) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
  PRIMARY KEY(patientid)
);

...
```

```
INSERT INTO patient VALUES
(1, 'Sara Brown'),
(2, 'John Smith'),
(3, 'Jack Ma');
```

Popular Datatypes (MySQL)

VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
INT(*size*)	Integer number . The *size* parameter specifies the maximum display width (which is 255)
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65.
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'

- There are many more data types. Please refer to the source: https://www.w3schools.com/sql/sql_datatypes.asp

Referential integrity **constraint**

Referential integrity **constraint**

- Q: in our patient visit data model, which table has to be create before the other one?
- (Which table can exist by itself? Patient or visit?)
- A: patient table has to exist before the visit, since the primary key of the patient table is used in the visit table

Q: What if we delete the second row of patient table?

```
DELETE FROM patient
WHERE patient_ID=2;
```

A: the 2nd and 3rd rows of Visit table would not make sense!

Lets try it at phpMyAdmin!

Patient

Patient_ID	Patient name
1	Sarah Brown
2	John Smith
3	Jack Ma

Visit

Visit_ID	Patient_ID	Clinic Visit
1	1	01/01/2001
2	2	08/08/2018
3	2	09/09/2019

- **Referential integrity** means for every value of a foreign key there is a primary key with that value
- A primary key must exist before the foreign key can be defined
- Must create the patient table before the visit table
- E.g. For every value of patinentID in the visit table there is a value of patientID in the patient table

Learn SQL 6

by Examples!

SQL Statements(queries)

- open:https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- `SELECT * FROM Customers;`
- This statement selects all the records(all rows, all tuples) in the customers table
  SQL keywords(such as SELECT) are NOT case sensitive: selEct is the same as SELECT.
  ; is not needed by good practice
- ```
  SELECT * FROM Customers
  WHERE Country='Mexico';
  ```
- Example 3:
  ```
  SELECT * FROM Customers
  WHERE CustomerID=1;
  ```
- Example:
  SELECT CustomerName FROM Customers;
- Example:
  SELECT CustomerName FrOM Customers WHERE City="London"
- Example:
  ```
  SELECT Country FROM Customers;
  ```
- Example:
- `SELECT DISTINCT Country FROM Customers; ( keep unique rows)`
  ```

WHERE clause operators

BETWEEN
LIKE
IN

WHERE Clause: Operators in The WHERE Clause

- Q: what does this one do?
- `SELECT *`
- `FROM OrderDetails`
- `where Quantity>80`

- Q: list the customers who live in the city of Berlin in Germany

- ```
  SELECT * FROM Customers
  WHERE Country='Germany'
  AND City='Berlin';
  ```

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column
  ```

WHERE Clause: AND, OR and NOT Operators

- **Q:** What products were ordered for quantity of more than 80?

```
SELECT * FROM Products,OrderDetails
WHERE Products.ProductID = OrderDetails.ProductID
AND
OrderDetails.Quantity>80
```

Equal signs makes sure we talk about same product in both tables. (remember how PationtID relates two tables in week 1)

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- % - The percent sign represents **zero, one, or multiple characters**
- _ - The underscore represents a **single character**
- **Examples**:
- SELECT * FROM Customers WHERE CustomerName LIKE 'a%'
- Finds any CustomerName that start with "a"
- SELECT * FROM Customers WHERE CustomerName LIKE '%st%'
- Finds any values that has "st" in any position
- Q: selects all customers with a CustomerName ending with "a"

Note: You learned MS Access in your previous course. MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_). We do not cover MS Access in this course

SQL Wildcard Characters

- A wildcard character is used to substitute one or more characters in a string.
- Wildcard characters are used with the SQL LIKE operator.
- The LIKE operator itself is used in a WHERE clause to search for a specified pattern in a column.

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit

SQL IN (NOT IN) Operator

- allows you to specify multiple values in a WHERE clause.
- is a shorthand for multiple OR conditions.
- SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');

```
Q: what does this query return?
SELECT * FROM customers
where country
 in
   (select country from customers
    where country like '__')
```

SQL BETWEEN Operator and sorting alphabetically (ASCII code)

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

- Example:
- SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;
-
  Example
- SELECT * FROM customers
- where country BETWEEN 'UA' AND 'UZ'

'AAA'
'AB'
'B'
..
'ZA'
'Z...Z'
'a'
'b'

Note: if we sort the strings above alphabetically in ascending order , the uppercase ones
appear before the lower case ones.

The reason is their ASCII codes.
E.g. ASCII code of 'A' is 65
But ASCII code of 'a' is 97

ASCII stands for American Standard Code for Information Interchange.

Order by [column name]

"ORDER BY *column*" and "ORDER BY *column* ASC|DESC" Keywords

- Q: What does this query return?
- SELECT * FROM [Customers] ORDER BY City
-

- **Q:** What products were ordered for quantity of more than 50? (sort the result in descending order so that the highest ordered product appears at top of result)

- **A:** SELECT * FROM Products,OrderDetails where Products.ProductID = OrderDetails.ProductID AND OrderDetails.Quantity>50 ORDER BY OrderDetails.Quantity DESC

Arithmetic within SQL clauses

- We can use arithmetic within SQL clauses.

- Example:
- SELECT Quantity, Quantity*10 FROM OrderDetails;

- Example:
- SELECT *, Quantity*Price FROM Products, OrderDetails WHERE Products.ProductID = OrderDetails.ProductID;

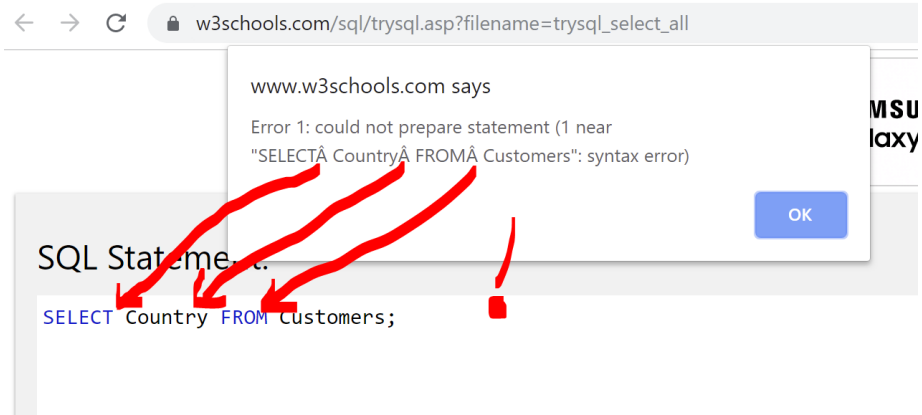
- Try this one too:
- SELECT *, Quantity*Price AS Total FROM Products, OrderDetails WHERE Products.ProductID = OrderDetails.ProductID;

- In AS Total, Total is an alias that can be used to name the new column our set result generate

Tips

Tip: beware of whitespaces

- Try this statement (copy it as is and paste it at https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in)
- and run it
- SELECT Country FROM Customers;
- What seems to be wrong here?



- The problem is that the above text contains invisible white spaces which are not valid blank spaces. You need to replace them with blank space
- You can use this link to remove white spaces: https://www.textfixer.com/tools/remove-white-spaces.php

Tip: confusing column names!

- Q1: Try to run this query. What is the problem? How can you fix it?

```
SELECT ProductID FROM Products,OrderDetails
WHERE Products.ProductID = OrderDetails.ProductID
AND
OrderDetails.Quantity>80
```

- Q2: Now remove the WHERE clause and again run the query. What do you get? Why? How can you fix it?

- Q3: what will this query return?
- SELECT * FROM Shippers, employees;

SQL functions

AVG, SUM, MIN, COUNT, …

SQL MIN(), MAX(), SUM, COUNT() Functions

- **Q**: How much is the most expensive product ?
  A: SELECT MAX(price) FROM products
- *SUM*(): returns the total sum of a numeric column
- **Q:** what is the total quantity of all products ordered combined?
- **A**: `SELECT SUM(Quantity) FROM OrderDetails;`
- Q: What is the total quantity of all goods shipped in all orders (use OrderDetails table)
- A: SELECT SUM(Quantity) FROM [OrderDetails]

- COUNT(), counts the number of rows in a table
- Try SELECT country FROM [Customers]
- Observe SELECT count(country) FROM [Customers]

- **Q:** How many different countries are there in the table Customers ?
- A: use count on SELECT distinct country FROM [Customers]
- SELECT count(distinct country) FROM [Customers]

Grouping on the result set …

GROUP BY Statement

- 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.

- Example:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

We count number of customerIDs that are in the group of same country

- Q: What does the above query return?
- Q: What if you remove " GROUP BY Country "?

Source: w3school.com

GROUP BY examples 1

- Q: Write a query to return number of orders by each city

- A: SELECT City,COUNT(orderID) FROM [Orders], customers where Orders.CustomerID=customers.CustomerID GROUP BY city

- Q: How can we verify the answer?

GROUP BY examples 2

- Q: Write a query statement to list the number of customers in each country, sorted high to low

- SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  ORDER BY COUNT(CustomerID) DESC;

- Q: Write the same query about for only the countries whose name starts with A:
- SELECT Country,COUNT(CustomerID)
- FROM Customers
- GROUP BY Country
- where country like "A%" Wait! "Where" does not work here!

GROUP BY examples 3

- Q: Write a query statement to count the cities of each country in the northWind DB at https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

```
SELECT Country, COUNT(City)
FROM Customers
GROUP BY Country
ORDER BY COUNT(City) DESC;
```

- Q: Can we make that query about a bit shorter and nicer?
- (yes we can use alias)

```
SELECT COUNT(City) as nofCities, Country
FROM Customers
GROUP BY Country
ORDER BY  nofCities  DESC;
```

The **Having** clause

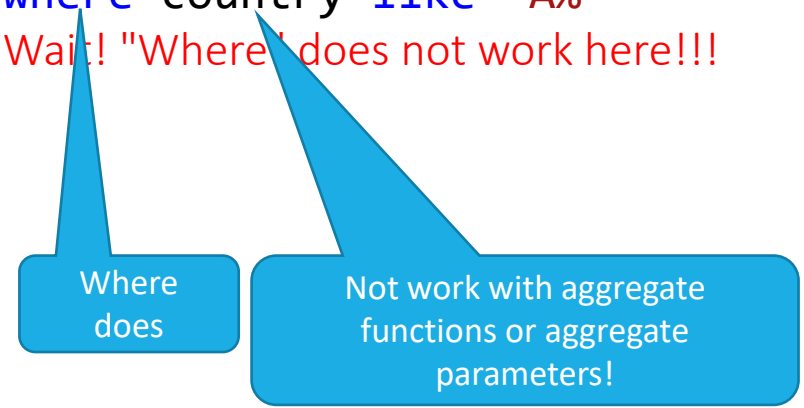
Made for use **with** aggerate functions and **after** "Group by"

WHERE does not work everyWHERE!

- Why this query does not run (throwes errors!)

```
SELECT  Country,COUNT(CustomerID)
FROM Customers
GROUP BY Country
where country like "A%"
```

- Wait! "Where" does not work here!!!



The SQL HAVING Clause

Select A1,A2,A3
From T1,T2, T3
Where condition
Group bY columns
Having conditions

Aggregation functions over values in multiple rows: min, max, sum, avg, count

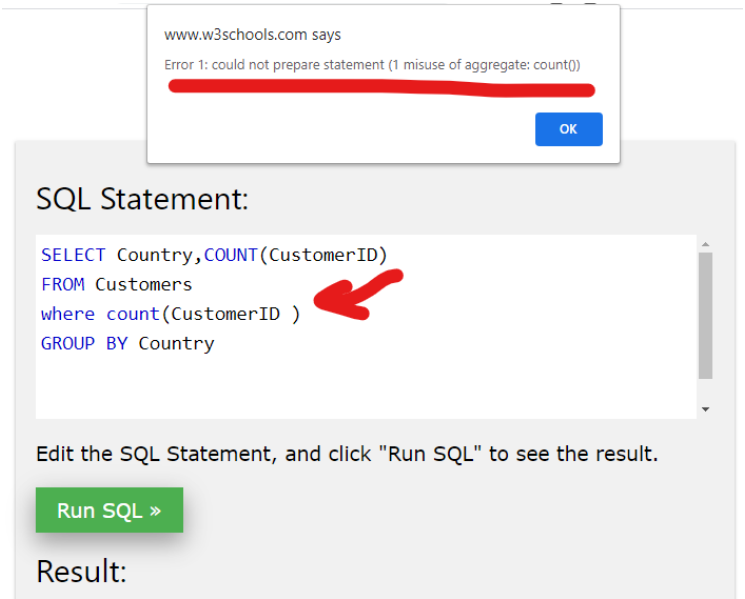
Note: The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one "or more columns".

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
So HAVING is like "WHERE" when we use GROUP BY(aggregated functions)!

The SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE clause could not be used with aggregate functions or after group by was performed.
- Example
- Write a query that lists the number of customers in each country.
- ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  ```

- Now Only include countries with more than 10 customers:
- (add this to last line)
- ```
  HAVING COUNT(CustomerID) > 10;
  ```



- SELECT Country,COUNT(CustomerID)
- FROM Customers
- where country like "A%"
- GROUP BY Country
- Its ok! "Where" is used before "Group by" and is not used on aggregate functions

"WHERE" does not work everyWHERE! (part 2)

```
SELECT Country,COUNT(CustomerID)
FROM Customers
GROUP BY Country
where country like "A%"
```

HAVING

Or place Where before Group by

```
SELECT Country,COUNT(CustomerID)
FROM Customers
where country like "A%"
GROUP BY Country
```

Q:List countries with more than 2 customers

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
GROUP BY Country
where c>2
```

HAVING

Note!

1) The "where" has to go before the "having" and the group by
2) "where" cannot be used on aggregate functions

"WHERE" does not work everyWHERE!

```
SELECT  Country, COUNT(CustomerID)
FROM Customers
GROUP BY Country
where CustomerName like "A%"
Not ok! Where cannot be used after we grouped rows
```

```
SELECT  Country, COUNT(CustomerID)
FROM Customers
where CustomerName like "A%"
GROUP BY Country
Is ok! Where is used before we group rows
```

"WHERE" does not work everyWHERE!

```
SELECT  Country, COUNT(CustomerID)
FROM Customers
where COUNT(CustomerID) >2
GROUP BY Country
```

1

Not ok! Where cannot be used on aggerate functions

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
where c>2
GROUP BY Country
```

2

Not ok! Where cannot be used on aggerate functions (place having c>2 after "group by "

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
having c>2
GROUP BY Country
```

3

Not ok! Having cannot be used before we have grouped the rows!

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
GROUP BY Country
having c>2
```

4

Its ok! Having is used after we have grouped the rows!

Summary of rules of using "having" and "where"

Note!

1) The "where" has to go before the "having" and the group by
2) "where" cannot be used on aggregate functions
3) "having" has to be used only after we used "group by" (having does not make sense if we have not grouped rows yet.

Sub-queries

Subquery

- Remember what we said about the **closure of SQL** where the result of each SQL query is again another table.

- That means we can run a query on table A and B which results in generating table T and run another query on Table T and C
- Example
- ```
  SELECT * FROM (SELECT * FROM Products)
  ```

- Or
-
  ```
  SELECT * FROM (SELECT * FROM Products where price > 10)
  ```
  ```

Subqueries, Example 1

- Q: Write a query to return products whose price is more than the average price of all products

- Step 1: get the average price
- ```
  Select avg(price) from products
  ```
- The result will be a number, 28.86, (which is actually table with only one cell)

- Step 2: Now we want to see which one of those products has price greater than 28.86:
- ```
  SELECT * FROM [Products] where price > 28.86
  ```

- Step 3: putting all together
- Replace 28.86 with the query from Step 1, but put it in brackets:

- ```
  SELECT * FROM [Products] where price >
  (Select avg(price) from products)
  ```


  ```

Subqueries Example 3:

- Q: List the most expensive product(s).
- A: We need the products' names and the prices. The table Products has both we need.
- Note: it is very important to know that we cannot assume there is only one product with the highest price. The products may be like this

Product name		Price
Milk	. . .	2
Jeans	. . .	10
Shirt	. . .	10

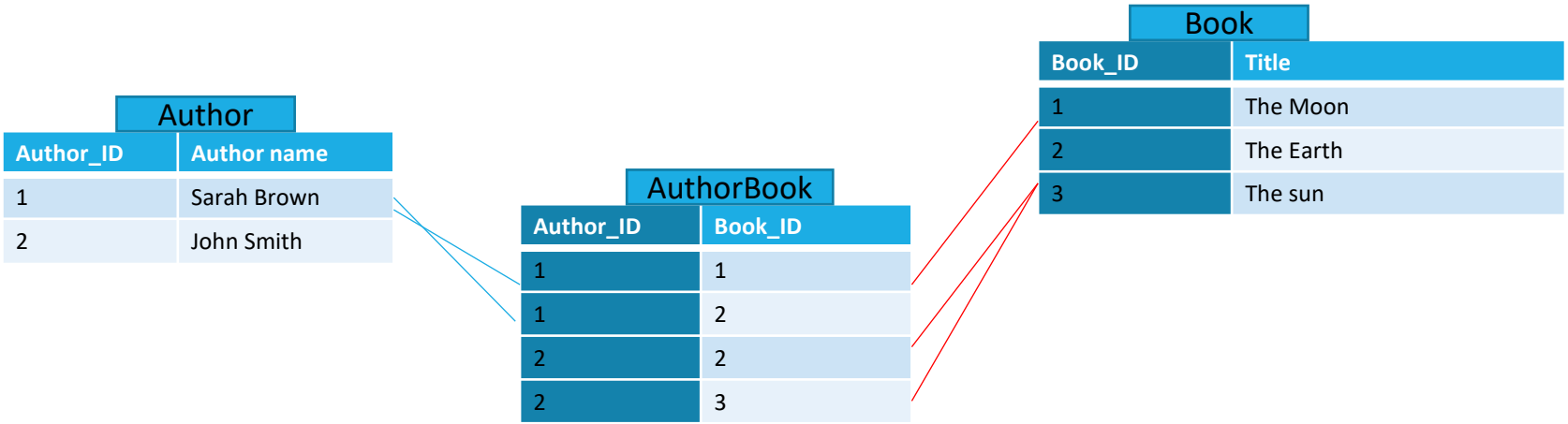
- Therefore the answer is
-
```
SELECT * FROM [Products]
where price = ( select max (price) from products)
```
-

Creating M:M

by example

Q: Design an RDBM to model the relationship between books and their authors:

- Sarah Brown authored "The Moon"
- Sarah Brown and John Smith co-authored "The Earth"
- John Smith authored "The Sun"



Author

Author_ID	Author name
1	Sarah Brown
2	John Smith

AuthorBook

Author_ID	Book_ID
1	1
1	2
2	2
2	3

Book

Book_ID	Title
1	The Moon
2	The Earth
3	The sun

- Q: What type of relationship between Autor and AuthorBook?
  Q: What type of relationship between Book and AuthorBook?
- A: 1:M and 1:M

M:M sample in SQL

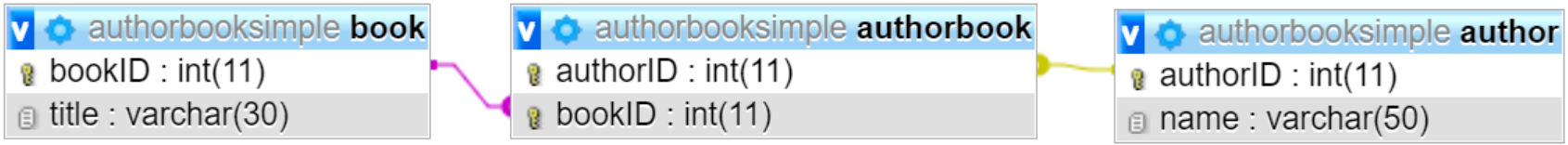
```
CREATE TABLE author (
 authorID INTEGER,
 name VARCHAR( 50),
 PRIMARY KEY(authorID)
);
CREATE TABLE book (
 bookID INTEGER,
 title VARCHAR( 30) NOT NULL,
 PRIMARY KEY(bookID)
);
CREATE TABLE authorBook (
 authorID INTEGER,
 bookID INTEGER,
 PRIMARY KEY( authorID, bookID),
 CONSTRAINT fk_has_author FOREIGN KEY( authorID) REFERENCES author( authorID),
 CONSTRAINT fk_has_book FOREIGN KEY( bookID) REFERENCES book( bookID)
 );
```

What kind of primary key is this?! $!!#%

Its **composite** primary key
Q: why there are two foreign keys here?

associative entity

- The table authorBook which links the other two tables is referred as ***associative entity*** or ***associative table***



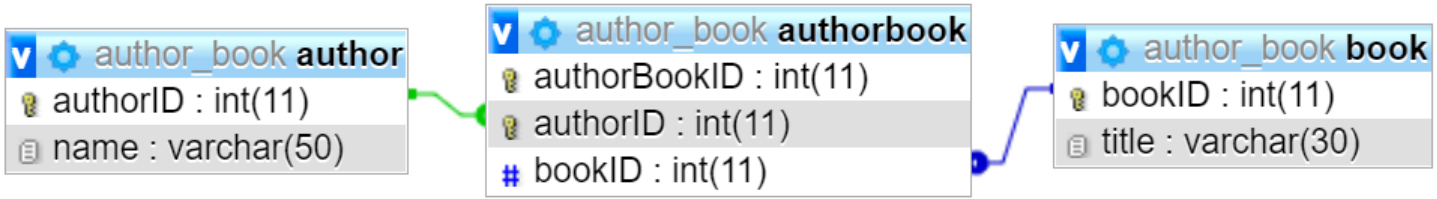
- Q: Now write SQL queries to insert these data into our database
- Sarah Brown authored "The Moon"
- Sarah Brown and John Smith co-authored "The Earth"
- John Smith authored "The Sun"

Same M:M relation, a bit different implementation of associative entity

```
CREATE TABLE author (
 authorID INTEGER,
 name VARCHAR( 50),
 PRIMARY KEY(authorID)
);
CREATE TABLE book (
 bookID INTEGER,
 title VARCHAR( 30) NOT NULL,
 PRIMARY KEY(bookID)
);
CREATE TABLE authorBook (
 authorBookID INTEGER,
 authorID INTEGER,
 bookID INTEGER,
 PRIMARY KEY( authorBookID, authorID),
 CONSTRAINT fk_has_author FOREIGN KEY( authorID) REFERENCES author( authorID),
 CONSTRAINT fk_has_book FOREIGN KEY( bookID) REFERENCES book( bookID)
 );
```

Do we need this?

- The table authorBook which links the other two tables is referred as *associative entity* or *associative table*



For more you can also check https://en.wikipedia.org/wiki/Associative_entity

AUTO INCREMENT Field

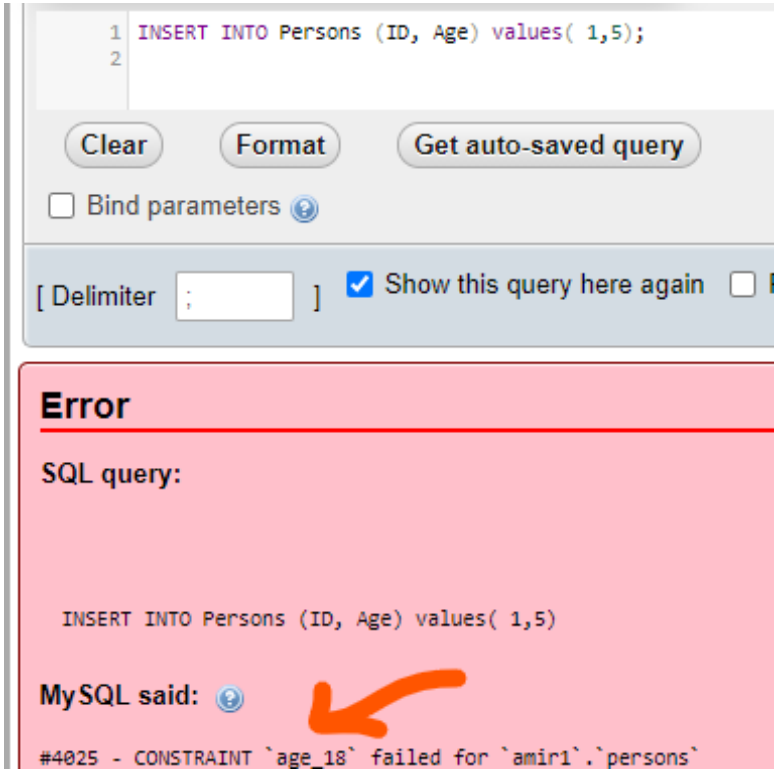
- allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

- MySQL
- ```
  CREATE TABLE Persons (
      Personid int AUTO_INCREMENT,
      LastName varchar(255) NOT NULL,
      PRIMARY KEY (Personid)
  );
  ```

- Q: Try it in your db using workbench or phpMyAdmin or in console
- Insert into Persons (LastName) values ("McDonnald")
- Note that we did not have to insert personID, why?
- Q: can we AUTO_INCREAMENT non integer fields?
  ```

CHECK - Ensures that all values in a column satisfies a specific condition

- SQL Server
- ```
  CREATE TABLE Persons (
      ID int ,
      LastName varchar(255) NOT NULL,
      Age int CHECK (Age>=18) );
  ```
- MySQL
- ```
  CREATE TABLE Persons (
      ID int ,
      LastName varchar(255) NOT NULL,
      Age int,
      CONSTRAINT age_18 CHECK (Age>=18) );
  ```

What's this?
Do we need it?

- **Q**:Try it in your DB with Workbench or phpMyAdmin
- **Q1**: try entering a person with age 5 and see what happens
- ```
  INSERT INTO Persons (ID, Age) values( 1,5);
  ```
- **Q2**: do you see any potential issue with the column Age (aside from the fact that it should be dateOfBirh, what else do you see?
- Tip: has something to do with null
- **Q3**: What if we wanted to also ensure the Age<150?
  ```

CHECK example error

- `INSERT INTO Persons (ID, Age) values( 1,5);`
- Will result in the error below:



Remarks

- SQL statements or clauses are NOT case sensitive
- Table names are NOT case sensitive
- SELECT City FROM Customers;
- SeleCT ciTy frOm cuStoMErs;
- **Keywords** e.g. "ORDER BY"
- **Operators** e.g. =, <>, OR
- **Clauses** e.g. WHERE
- **Statements** e.g. select * from …
- **Functions** like: min(), max()
- Operators are used in clauses
- **Keyword**: SQL reserved words
- **Identifiers**: Names we choose
- Constants: Fixed values
- **Clauses**: Portion of SQL statement