

# Web Design and Programming

Week 9

20 June 2024

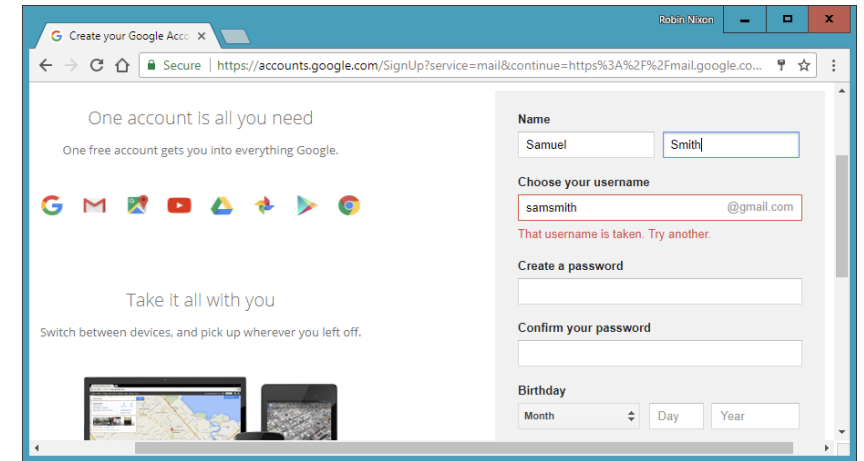
Instructor: Dr. Peeraya Sripian

# Course schedule

Week	Date	Topic
1	4/18	Intro to WWW, Intro to HTML
2	4/25	CSS Fundamental
	5/2	Holiday (GW)
3	5/9	CSS and Bootstrap
4	5/16	Work on midterm project
5	COIL 22 MAY 18:00-19:30 (counted as 1 class, replacing 23 May)	
6	5/30	Midterm project presentation week
7	6/6	PHP fundamentals + Installation XAMPP
8	6/13	PHP fundamentals 2 + Intro of Final project
9	6/20	mySQL fundamentals
10	6/27	Assessing MySQL using PHP, MVC pattern
11	7/4	Cookies, sessions, and authentication + <b>Proposal of final project</b>
12	7/11	Javascript and PHP validation
13	7/18	Final project development
14	7/25	<b>Final project presentation</b>

# Bringing it all together

- PHP, MySQL, JavaScript (sometimes aided by jQuery or etc.), CSS, and HTML all work together to produce “dynamic web content”
  - PHP handles all main work on web server
  - MySQL manages all the data
  - CSS + JavaScript looks after web page presentation
  - JavaScript → Talk to PHP code whenever it need to update something



Asynchronous communication in Gmail

# The form Element

- The **form** element is a container for all the content in the form and its input controls.
- The **action** and **method** attributes are necessary for interacting with the processing program.

```
<form action="URL" method="POST or GET">  
  <!-- Form content and inputs here -->  
</form>
```

# The `$_POST` Array

- A browser sends user input through either `GET` or `POST` request
- The `POST` request is usually preferred
- The web server bundles up all user input and puts in array `$_POST`
- To access this array, use the element key (ex: "isbn")
  - `$_POST['isbn']` or `$_POST["isbn"]`
- All values in `$_POST` will be retrieved at the beginning of the program and ignored after that

# Today's topic

- MySQL basics
  - Summary of Database Terms
  - Accessing MySQL via the CMD / Terminal
  - MySQL Functions
  - Accessing MySQL via phpMyAdmin
- Break for 10 minutes
- Mastering MySQL
  - Database Design
  - Normalization
  - Relationships
  - Transactions
- Assignment 6

# MySQL

- MySQL is a very popular, open source database for web servers.
- Officially pronounced “my Ess Que Ell” (not my sequel).
- Handles very large databases; very fast performance.
- Why are we using MySQL?
  - Free (much cheaper than Oracle!)
  - Each student can install MySQL locally.
  - Easy to use Shell for creating tables, querying tables, etc.
  - Easy to use with Java JDBC

# MySQL Basics

- Database – A structured collection of records or data stored in computer and organized so it can be quickly searched
- SQL – Structured Query Language
  - Loosely based on English
  - Used in other databases such as Oracle and Microsoft SQL server
- MySQL Database contains one or more tables
  - Each table contains *records* or *rows*
  - Within the rows, there are *columns* or *fields* that contain the data



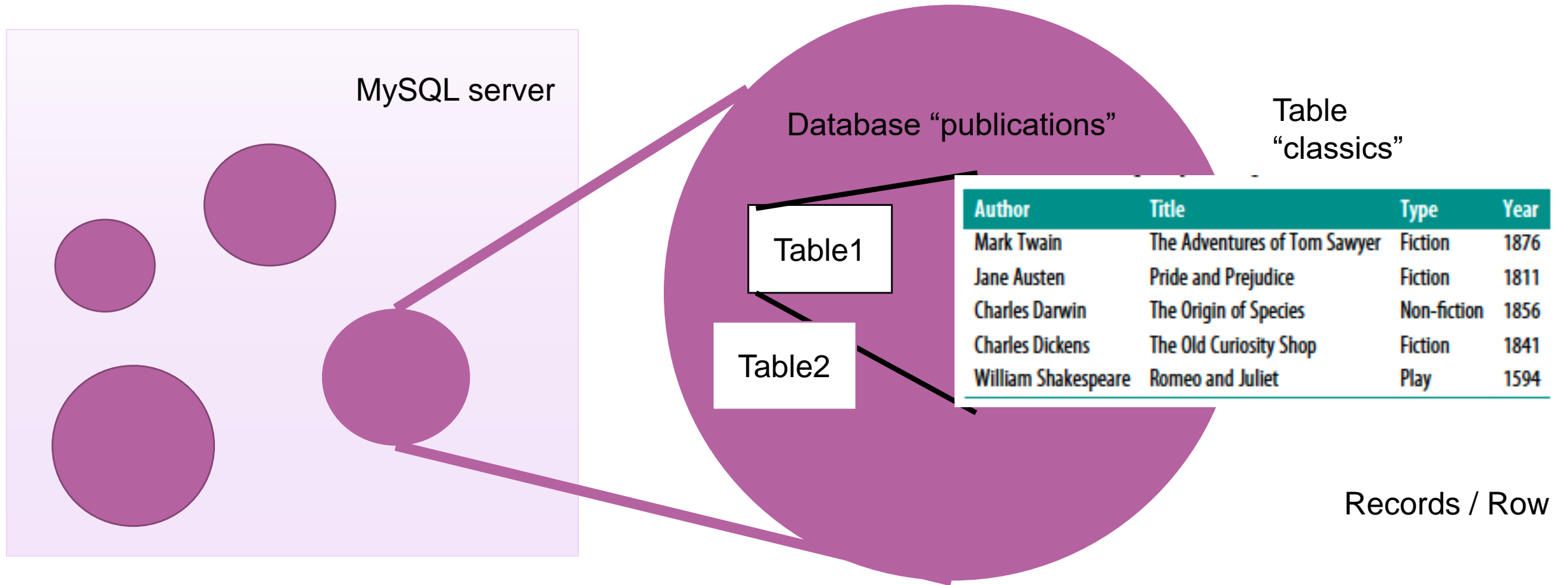
# Summary of Database Terms

- Database
  - The overall container for a collection of MySQL data
- Table
  - A subcontainer within a database that stores the actual data
- Row
  - A single record within a table, which may contain several fields
- Column
  - The name of a field within a row

Example of a simple database

Author	Title	Type	Year
Mark Twain	The Adventures of Tom Sawyer	Fiction	1876
Jane Austen	Pride and Prejudice	Fiction	1811
Charles Darwin	The Origin of Species	Non-fiction	1856
Charles Dickens	The Old Curiosity Shop	Fiction	1841
William Shakespeare	Romeo and Juliet	Play	1594

# Structure



# Relational database

## One-to-One

Table 9-8a (Customers)

CustNo	Name
1	Emma Brown
2	Darren Ryder
3	Earl B. Thurston
4	David Miller

Table 9-8b (Addresses)

Address	Zip
1565 Rainbow Road	90014
4758 Emily Drive	23219
862 Gregory Lane	40601
3647 Cedar Lane	02154

## Many-to-Many

Columns from  
Table 9-8b  
(Customers)

Zip	Cust.
90014	1
23219	2
(etc...)	
40601	3
02154	4

Intermediary  
Table 9-12  
(Customer/ISBN)

CustNo	ISBN
1	0596101015
2	0596101015
2	0596527403
3	0596005436
4	0596006815

Columns from  
Table 9-4  
(Titles)

ISBN	Title
0596101015	PHP Cookbook
(etc...)	
0596527403	Dynamic HTML
0596005436	PHP and MySQL
0596006815	Programming PHP

## One-to-Many

Table 9-8a (Customers)

CustNo	Name
1	Emma Brown
2	Darren Ryder
(etc...)	
3	Earl B. Thurston
4	David Miller

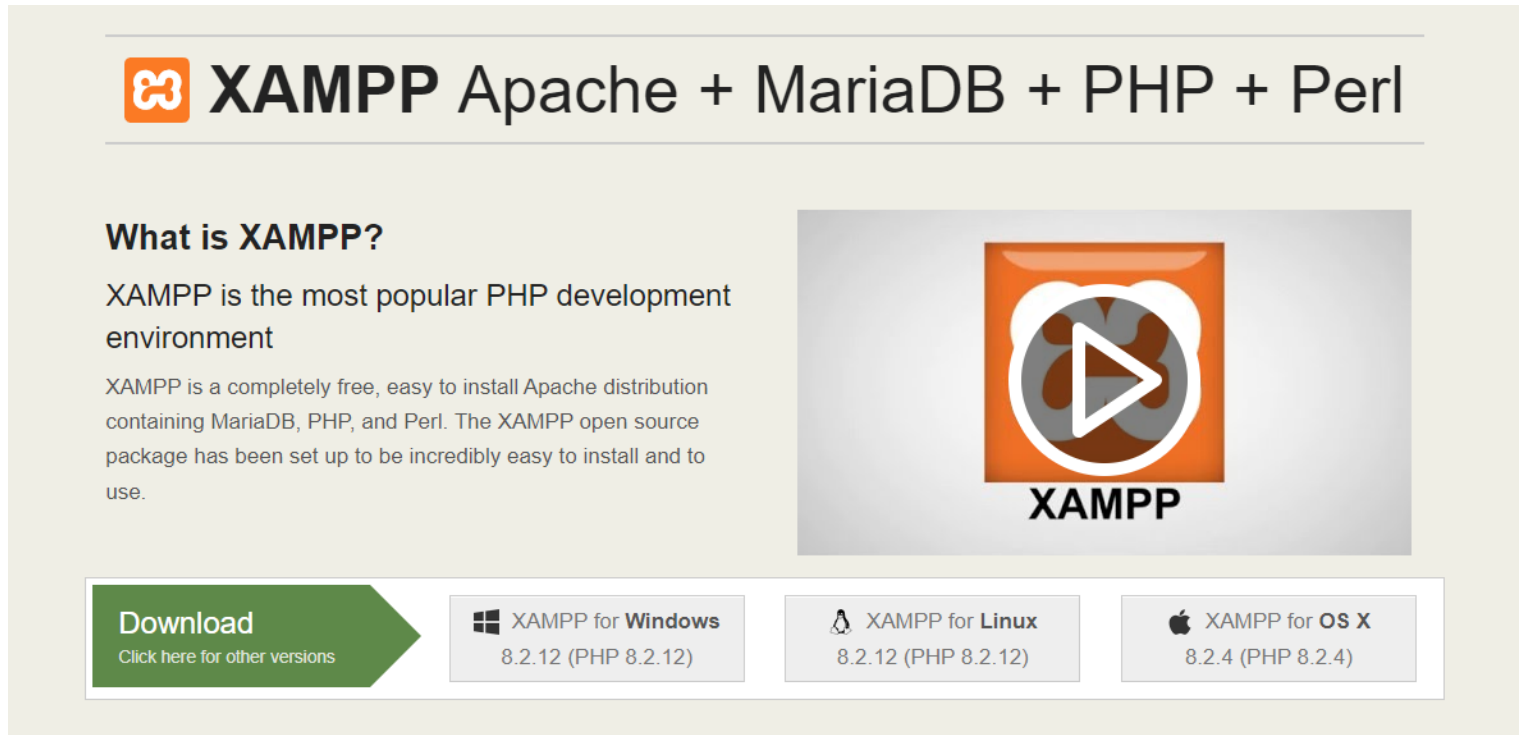
Table 9-7. (Purchases)

CustNo	ISBN	Date
1	0596101015	Mar 03 2009
2	0596527403	Dec 19 2008
2	0596101015	Dec 19 2008
3	0596005436	Jun 22 2009
4	0596006815	Jan 16 2009

# For creating your own server (localhost)

- Let's install XAMPP

<https://www.apachefriends.org/index.html>



The screenshot shows the XAMPP website. At the top, there is a logo and the text "XAMPP Apache + MariaDB + PHP + Perl". Below this, there is a section titled "What is XAMPP?" which states that XAMPP is the most popular PHP development environment and is a completely free, easy-to-install Apache distribution containing MariaDB, PHP, and Perl. To the right of this text is a large XAMPP logo. At the bottom, there is a "Download" button with a link to other versions, and three buttons for downloading XAMPP for Windows, Linux, and OS X, each with the version number and PHP version.


**XAMPP** Apache + MariaDB + PHP + Perl


**What is XAMPP?**


XAMPP is the most popular PHP development environment

XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.

**Download**  
Click here for other versions

 **XAMPP for Windows**  
8.2.12 (PHP 8.2.12)

 **XAMPP for Linux**  
8.2.12 (PHP 8.2.12)

 **XAMPP for OS X**  
8.2.4 (PHP 8.2.4)



# Accessing mysql via command line

You need to change the status of MySQL database to “Running” in XAMPP first

- For MAC
  - open Xampp application manager
  - Welcome tab – choose Open Application Folder
  - Right click on folder “**bin**” – select new terminal at folder
  - Type the following command

```
./mysql --user=root --password=
```

- For Window
  - Open Xampp application manager
  - Explorer -> navigate to `mysql\bin` folder
  - Type cmd in the address bar, it will open the location in the command prompt
  - Type the following command

```
C:\xampp\mysql\bin
```

```
mysql.exe -u root --password
```

- When prompt “Enter password”, press enter (no password is set)

Note: The user name and password is set to “root” and “” for the first installation of XAMPP  
Once you change your user name and the password, you need to use them in login

# Class server

(need to connect to university network)

- Let's access Mysql command line in the server

1. Type `SSH username@172.21.82.208` enter

2. Type your `password` enter

3. Type `msql` enter

4. Type `mysql -u username -p` enter

5. Then type your `password` enter

# Accessing MySQL

- MySQL provides an interactive shell for creating tables, inserting data, etc.
  - MySQL via the command line
  - MySQL via the PHP code

# MySQL client via the command line

- MySQL client – a program that interfaces with their server
- `mysql` – a text based client that is most commonly used
  - Usually comes with your installation of MySQL
  - Should be in `/bin/` or `/usr/bin` directory
- You need a username and password to connect to MySQL, even with the `mysql` client



# Basic Queries

- Once logged in, you can try some simple queries.
- For example:

```
gift@giftserver:~$ mysql
bash-4.4# mysql -u gift -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3128
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

- Note that most MySQL commands end with a semicolon (;)
- MySQL returns the total number of rows found, and the total time to execute the query.

## Basic Queries

- Keywords may be entered in any lettercase.
- The following queries are **equivalent**: ->

```
MariaDB [(none)]> SELECT VERSION(), CURRENT_DATE;  
+-----+-----+  
| VERSION() | CURRENT_DATE |  
+-----+-----+  
| 10.4.28-MariaDB | 2023-06-06 |  
+-----+-----+  
1 row in set (0.000 sec)
```

```
MariaDB [(none)]> select version(), current_date;  
+-----+-----+  
| version() | current_date |  
+-----+-----+  
| 10.4.28-MariaDB | 2023-06-06 |  
+-----+-----+  
1 row in set (0.000 sec)
```

```
MariaDB [(none)]> SeLeCt vErSiOn(), current_DATE;  
+-----+-----+  
| vErSiOn() | current_DATE |  
+-----+-----+  
| 10.4.28-MariaDB | 2023-06-06 |  
+-----+-----+  
1 row in set (0.000 sec)
```

# Canceling a Command

- If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing `\c`

```
MariaDB [(none)]> SELECT  
                    -> USER()  
                    -> \c  
MariaDB [(none)]>
```

# When you get stuck

`mysql>`

- If you see this, then you can enter a command, end it with ; and press enter.

- If you see a prompt like this:

`'>`

- Waiting for single quote for closing the string

`">`

- Waiting for double quote for closing the string

`->`

- Terminate the query or exit from the query using \c



# Using a Database

- To get started on your own database, first check which databases currently exist.
- Use the `SHOW` statement to find out which databases currently exist on the server:

```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
2 rows in set (0.01 sec)
```



# Let's create a table

```
CREATE TABLE test.books(book_id INT, title TEXT, status INT);
```

- Create a table in test database, the table is named “books”
- Also define 3 columns for this table:
  - book\_id – integer types (INT)
  - title – text (anything you can type at the keyboard)
  - status - INT



# Let's insert some values

```
INSERT INTO books VALUES ( 100 , 'Heart of Darkness' , 0 );  
INSERT INTO books VALUES ( 101 , 'The Catcher of the Rye' , 1 );  
INSERT INTO books VALUES ( 102 , 'My Antonia' , 0 );
```

- Add data to the books table



# Display the data you just inserted

```
SELECT * FROM books;
```

- Select everything (\*) from table books

```
SELECT * FROM books WHERE status = 1;
```

- Select only rows in which **status** equals 1

```
[MariaDB [test]> select * from books;
+-----+-----+-----+
| book_id | title                | status |
+-----+-----+-----+
|      100 | Heart of Darkness    |      0 |
|      101 | The Catcher of the Rye |      1 |
|      102 | My Antonia           |      0 |
+-----+-----+-----+
3 rows in set (0.005 sec)
```

```
[MariaDB [test]> select * from books where status = 1;
+-----+-----+-----+
| book_id | title                | status |
+-----+-----+-----+
|      101 | The Catcher of the Rye |      1 |
+-----+-----+-----+
1 row in set (0.002 sec)
```





# Update the data

```
UPDATE books SET status = 0 WHERE book_id = 101;  
SELECT * FROM books WHERE status = 0;
```

- Update table books by setting the value of status to 0 for all rows where book\_id = 101
- Show all data from table books where status is 0

```
UPDATE books SET title = 'The Catcher in the Rye' , status = 1  
WHERE book_id = 101 ;
```

- Update table books by setting the value of title to "The Catcher in the Rye" and status to 1 for all rows where book\_id=101



# Join tables

```
CREATE TABLE status_names ( status_id INT , status_name CHAR ( 8 ));  
INSERT INTO status_names VALUES ( 0 , 'Inactive' ), ( 1 , 'Active' );
```

- Create another table to store the definition of each status value

```
SELECT book_id, title, status_name  
FROM books JOIN status_names  
WHERE status = status_id;
```

- SQL statements can be broken in multiple lines.  
Nothing is processed until a ; is typed

```
[MariaDB [test]> insert into status_names values(0,'Inactive'),(1,'Active');  
Query OK, 2 rows affected (0.005 sec)  
Records: 2 Duplicates: 0 Warnings: 0
```

```
[MariaDB [test]> select * from status_names;
```

status_id	status_name
0	Inactive
1	Active

2 rows in set (0.000 sec)

```
[MariaDB [test]> select book_id, title, status_name  
-> from books join status_names  
-> where status=status_id;
```

book_id	title	status_name
100	Heart of Darkness	Inactive
101	The Catcher of the Rye	Active
102	My Antonia	Inactive

3 rows in set (0.005 sec)

# Common MySQL commands

Command	Action
ALTER	Alter a database or table
BACKUP	Back up a table
\c	Cancel input
CREATE	Create a database
DELETE	Delete a row from a table
DESCRIBE	Describe a table's columns
DROP	Delete a database or table
EXIT (Ctrl-C)	Exit
GRANT	Change user privileges
HELP (\h, \?)	Display help
INSERT	Insert data
LOCK	Lock table(s)
QUIT (\q)	Same as EXIT
RENAME	Rename a table
SHOW	List details about an object
SOURCE	Execute a file
STATUS (\s)	Display the current status
TRUNCATE	Empty a table
UNLOCK	Unlock table(s)
UPDATE	Update an existing record
USE	Use a database

# Data Types

# CHAR / VARCHAR

- Accept Text string and impose a limit on the size of the field. CHAR has a specified size while VARCHAR does not pad the space (size can be vary). VARCHAR has some overhead --> Slightly slower
- CHAR is a little more effiecient if the size are similar in all records

Data type	Bytes used	Examples
CHAR( <i>n</i> )	Exactly <i>n</i> ( $\leq 255$ )	CHAR(5) "Hello" uses 5 bytes CHAR(57) "Goodbye" uses 57 bytes
VARCHAR( <i>n</i> )	Up to <i>n</i> ( $\leq 65535$ )	VARCHAR(7) "Hello" uses 5 bytes VARCHAR(100) "Goodbye" uses 7 bytes

# BINARY / TEXT

- BINARY data type can be used to store strings of bytes that is not character set (for example: GIF image)

Data type	Bytes used	Examples
<code>BINARY(<i>n</i>)</code>	Exactly <i>n</i> ( $\leq 255$ )	As <code>CHAR</code> but contains binary data
<code>VARBINARY(<i>n</i>)</code>	Up to <i>n</i> ( $\leq 65535$ )	As <code>VARCHAR</code> but contains binary data

- TEXT data type – same as VARCHAR, no default value and indexes only first *n* characters of a TEXT column

Data type	Bytes used	Attributes
<code>TINYTEXT(<i>n</i>)</code>	Up to <i>n</i> ( $\leq 255$ )	Treated as a string with a character set
<code>TEXT(<i>n</i>)</code>	Up to <i>n</i> ( $\leq 65535$ )	Treated as a string with a character set
<code>MEDIUMTEXT(<i>n</i>)</code>	Up to <i>n</i> ( $\leq 1.67\text{e}+7$ )	Treated as a string with a character set
<code>LONGTEXT(<i>n</i>)</code>	Up to <i>n</i> ( $\leq 4.29\text{e}+9$ )	Treated as a string with a character set

# BLOB

- BLOB – Binar Large Object
- Cannot have default value (unlike BINARY)

Data type	Bytes used	Attributes
TINYBLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 255$ )	Treated as binary data — no character set
BLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 65535$ )	Treated as binary data — no character set
MEDIUMBLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 1.67\text{e}+7$ )	Treated as binary data — no character set
LONGBLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 4.29\text{e}+9$ )	Treated as binary data — no character set

# Numeric

- MySQL supports various numeric data types
- The most memory that a numeric field can use up is **8 bytes**
- It is better to choose the smallest data that fit the largest value you can expect for that column - to keep your database small and quick

Data type	Bytes used	Minimum value		Maximum value	
		Signed	Unsigned	Signed	Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8.38e+6	0	8.38e+6	1.67e+7
INT / INTEGER	4	-2.15e+9	0	2.15e+9	4.29e+9
BIGINT	8	-9.22e+18	0	9.22e+18	1.84e+19
FLOAT	4	-3.40e+38	<i>n/a</i>	3.4e+38	<i>n/a</i>
DOUBLE / REAL	8	-1.80e+308	<i>n/a</i>	1.80e+308	<i>n/a</i>



# Numeric

To specify whether a data type is unsigned

- `CREATE TABLE tablename (fieldname INT UNSIGNED);`
- `CREATE TABLE tablename (fieldname INT(4));`

Any numbers with a width of **<4** be padded with zeros (only if the field is not specified with a width or greater number)

- `CREATE TABLE tablename (fieldname INT(4) ZEROFILL);`

# DATE and TIME

- **TIMESTAMP** has a very narrow range (from years 1970 through 2037)
- **DATETIME** will hold any date you specify

Data type	Time/date format
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (Only years 0000 and 1901–2155)

# AUTO\_INCREMENT attribute

- When we need to ensure that every row is guaranteed to be unique
- At least one value must differ in any two rows.
- This data type can be used as `id`

```
ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;
```

- `ALTER` is similar to `CREATE`, it operates on existing table and can add, change or delete columns
- Now `id` is set as `key` (index), will have unique number every time

```
CREATE TABLE classics (author VARCHAR(128), title VARCHAR(128),  
type VARCHAR(16), year CHAR(4),  
id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE InnoDB;
```

# Indexes

# Creating an Index

- To achieve fast searches, you can do by adding index
- Index can be created when you created a table, or afterward
- Types of indexes
  - Regular INDEX
  - A PRIMARY KEY
  - FULLTEXT index

```
ALTER TABLE classics ADD INDEX(author(20));
```

- Author column is indexed, but for the first 20 characters only. To optimize database access speed.

# Creating the table with indexes

- Adding index to a large table can take a very long time
- It is better to create with the table

```
CREATE TABLE classics (  
  author VARCHAR(128),  
  title VARCHAR(128),  
  category VARCHAR(16),  
  year SMALLINT,  
  INDEX(author(20)),  
  INDEX(title(20)),  
  INDEX(category(4)),  
  INDEX(year)) ENGINE InnoDB;
```

# Primary Keys

- AUTO\_INCREMENT can be used to create primary key
- However, for publications tables, maybe ISBN number is better

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

Be careful that this command will not work if there is data in the table (error: Duplicate entry for key1)

Therefore, for a table with existing data, better to add a column (without index) and populate the column with data and use a primary key



# Table classics

```
CREATE TABLE classics ( author VARCHAR( 128), title VARCHAR( 128), type  
VARCHAR( 16), year CHAR( 4), id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY)  
ENGINE InnoDB;
```

- This SQL command creates a table named "classics" with columns for author, title, type, year, and id. The id column is a unique identifier that automatically increments for each new row. The table uses the InnoDB storage engine.





# Adding data to table classics

```
INSERT INTO classics( author, title, type, year) VALUES('Mark Twain', ' The  
Adventures of Tom Sawyer', 'Fiction', '1876');
```

```
INSERT INTO classics( author, title, type, year) VALUES('Jane Austen', '  
Pride and Prejudice', ' Fiction', '1811');
```

```
INSERT INTO classics( author, title, type, year) VALUES('Charles Darwin', '  
The Origin of Species', ' Non-Fiction', ' 1856');
```

```
INSERT INTO classics( author, title, type, year) VALUES('Charles Dickens', '  
The Old Curiosity Shop', ' Fiction', ' 1841');
```

```
INSERT INTO classics( author, title, type, year) VALUES('William  
Shakespeare', ' Romeo and Juliet', ' Play', ' 1594');
```



Let's do it together

# Populating the new column with data and using a primary key

```
ALTER TABLE classics ADD isbn CHAR(13);  
UPDATE classics SET isbn='9781598184891' WHERE year='1876';  
UPDATE classics SET isbn='9780582506206' WHERE year='1811';  
UPDATE classics SET isbn='9780517123201' WHERE year='1856';  
UPDATE classics SET isbn='9780099533474' WHERE year='1841';  
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
```



```
ALTER TABLE classics ADD PRIMARY KEY(isbn);
```

If you already have a column that is set as primary key (for example: `id`), it will give you error "Multiple primary key defined" → drop that column first

```
ALTER TABLE classics DROP id;
```

```
DESCRIBE classics;
```



# Adding the duplicating data

```
INSERT INTO classics( author, title, type, year, isbn)
VALUES('Charles Dickens', 'Little Dorrit', 'Fiction', '1857',
'9780141439969');
```

# Querying a MySQL Database

# SELECT

`SELECT something FROM tablename;`

- Something can be `*` = every column
- Or select only certain columns

```
SELECT author,title FROM classics;  
SELECT title,isbn FROM classics;
```

- Or `COUNT`

```
SELECT COUNT(*) FROM classics;
```

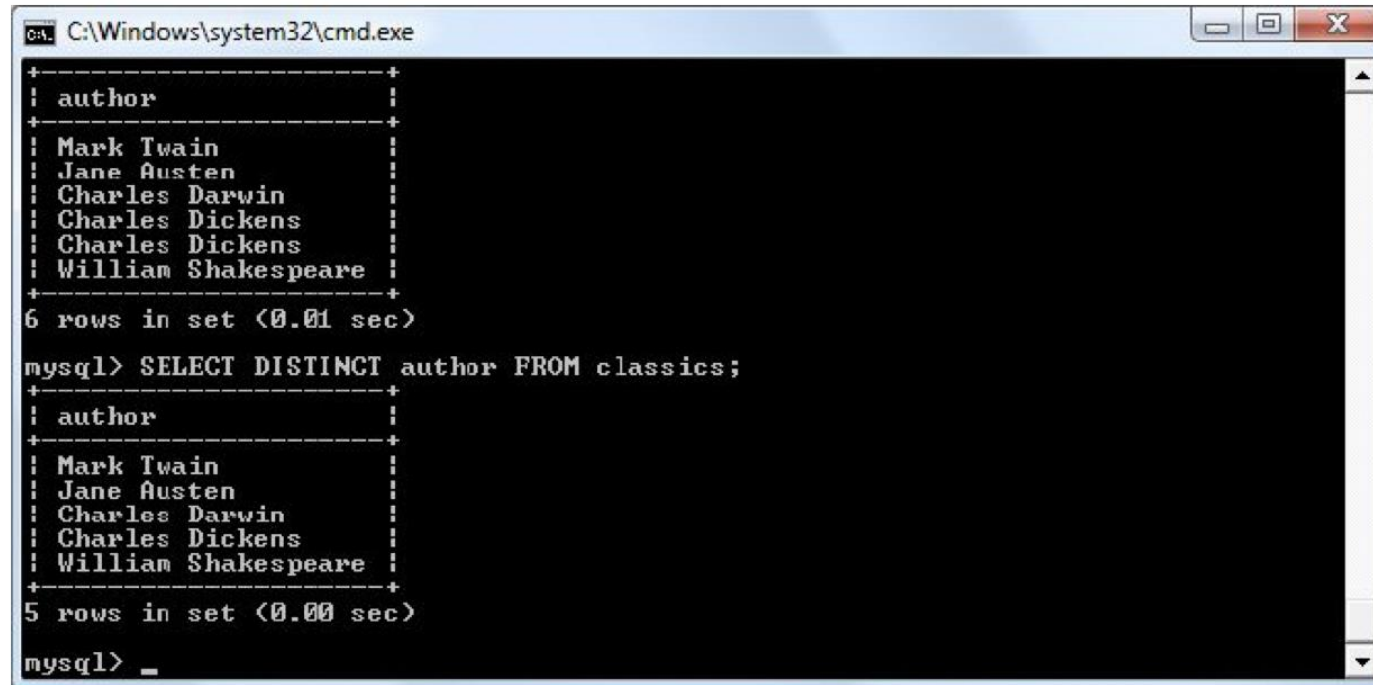
- Display number of rows in the table



# SELECT DISTINCT

- Weed out multiple entries when they contain the same data

```
SELECT author FROM classics;  
SELECT DISTINCT author FROM classics;
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the output of two MySQL queries. The first query is "SELECT author FROM classics;" which returns 6 rows: Mark Twain, Jane Austen, Charles Darwin, Charles Dickens, Charles Dickens, and William Shakespeare. The second query is "SELECT DISTINCT author FROM classics;" which returns 5 rows: Mark Twain, Jane Austen, Charles Darwin, Charles Dickens, and William Shakespeare. The prompt "mysql>" is visible at the bottom of the window.

```
C:\Windows\system32\cmd.exe  
+-----+  
| author |  
+-----+  
| Mark Twain |  
| Jane Austen |  
| Charles Darwin |  
| Charles Dickens |  
| Charles Dickens |  
| William Shakespeare |  
+-----+  
6 rows in set (0.01 sec)  
  
mysql> SELECT DISTINCT author FROM classics;  
+-----+  
| author |  
+-----+  
| Mark Twain |  
| Jane Austen |  
| Charles Darwin |  
| Charles Dickens |  
| William Shakespeare |  
+-----+  
5 rows in set (0.00 sec)  
  
mysql> _
```



# DELETE

- Remove a row from a table

```
DELETE FROM classics WHERE title='Little Dorrit';
```



# WHERE

- Narrow down queries by returning only those where a certain expression is true

```
SELECT author,title FROM classics WHERE author="Mark Twain";
SELECT author,title FROM classics WHERE isbn="9781598184891";
```

author	title
Mark Twain	The Adventures of Tom Sawyer

- Pattern matching with **LIKE** qualifier (% can match empty string too)

```
SELECT author,title FROM classics WHERE author LIKE "Charles%";
```

author	title
Charles Darwin	The Origin of Species
Charles Dickens	The Old Curiosity Shop

```
SELECT author,title FROM classics WHERE title LIKE "%Species";
SELECT author,title FROM classics WHERE title LIKE "%and%";
```



# LIMIT

Row index in sql start at 0

- Choost how many rows to return in a query, and where in the table to start returning them

<code>SELECT author,title FROM classics LIMIT 3;</code>	return the first 3 rows from the table
<code>SELECT author,title FROM classics LIMIT 1,2;</code>	returns 2 rows starting at 1
<code>SELECT author,title FROM classics LIMIT 3,1;</code>	returns a single row starting at 3

```
MariaDB [publications]> SELECT author,title FROM classics LIMIT 3;
```

author	title
Charles Dickens	The Old Curiosity Shop
Charles Darwin	The Origin of Species
Jane Austen	Pride and Prejudice



# UPDATE .. SET

- Update the contents of a field

```
MariaDB [publications]> UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)'  
-> WHERE author='Mark Twain';  
Query OK, 1 row affected (0.003 sec)  
Rows matched: 1 Changed: 1 Warnings: 0
```

```
MariaDB [publications]> select * from classics;
```

author	title	type	year	isbn
Charles Dickens	The Old Curiosity Shop	Fiction	1841	9780099533474
Charles Darwin	The Origin of Species	Non-Fiction	1856	9780517123201
Jane Austen	Pride and Prejudice	Fiction	1811	9780582506206
Mark Twain (Samuel Langhorne Clemens)	The Adventures of Tom Sawyer	Fiction	1876	9781598184891

4 rows in set (0.001 sec)



# ORDER BY

- Sorts returned results by one or more columns in ascending or descending order

```
MariaDB [publications]> SELECT author, title FROM classics ORDER BY author;
```

author	title
Charles Darwin	The Origin of Species
Charles Dickens	The Old Curiosity Shop
Jane Austen	Pride and Prejudice
Mark Twain (Samuel Langhorne Clemens)	The Adventures of Tom Sawyer

4 rows in set (0.002 sec)

```
MariaDB [publications]> SELECT author, title FROM classics ORDER by title DESC;
```

author	title
Charles Darwin	The Origin of Species
Charles Dickens	The Old Curiosity Shop
Mark Twain (Samuel Langhorne Clemens)	The Adventures of Tom Sawyer
Jane Austen	Pride and Prejudice

4 rows in set (0.001 sec)



# GROUP BY

- Group results returned from queries
- Good for retrieving information about a group of data

```
MariaDB [publications]> SELECT type, COUNT(author) FROM classics GROUP BY type;
```

```
+-----+-----+
| type      | COUNT(author) |
+-----+-----+
| Fiction    | 3             |
| Non-Fiction | 1             |
+-----+-----+
2 rows in set (0.007 sec)
```

# Join tables together

- Natural Join
  - Automatically joins columns that have the same name
- Join on
  - Specify the column on which to join two tables
- Using AS
  - Follow a table name with AS and the alias to use
  - Can also be used to rename a column (whether or not joining tables)
  - Useful when you have a long queries that reference the same table names many times



# Example of join

- Supposed we have two tables, classics and customers

```
MariaDB [publications]> SELECT * FROM customers;
```

name	isbn
Joe Bloggs	9780099533474
Jack Wilson	9780517123201
Mary Smith	9780582506206

```
MariaDB [publications]> SELECT * FROM classics;
```

author	title	type	year	isbn
Charles Dickens	The Old Curiosity Shop	Fiction	1841	9780099533474
Charles Darwin	The Origin of Species	Non-Fiction	1856	9780517123201
Jane Austen	Pride and Prejudice	Fiction	1811	9780582506206
Mark Twain (Samuel Langhorne Clemens)	The Adventures of Tom Sawyer	Fiction	1876	9781598184891

4 rows in set (0.001 sec)



# Example of join (cont.)

- Joining two tables using SELECT, WHERE

```
MariaDB [publications]> SELECT name, author, title FROM customers, classics WHERE customers.isbn=classics.isbn;
```

+-----+-----+-----+		
name	author	title
+-----+-----+-----+		
Joe Bloggs	Charles Dickens	The Old Curiosity Shop
Jack Wilson	Charles Darwin	The Origin of Species
Mary Smith	Jane Austen	Pride and Prejudice
+-----+-----+-----+		

```
3 rows in set (0.002 sec)
```



# Example of join (cont.)

- Joining two tables using NATURAL JOIN

```
MariaDB [publications]> SELECT name, author, title FROM customers NATURAL JOIN classics;
```

name	author	title
Joe Bloggs	Charles Dickens	The Old Curiosity Shop
Jack Wilson	Charles Darwin	The Origin of Species
Mary Smith	Jane Austen	Pride and Prejudice

```
3 rows in set (0.001 sec)
```

- Same result can be achieved using JOIN..ON or AS

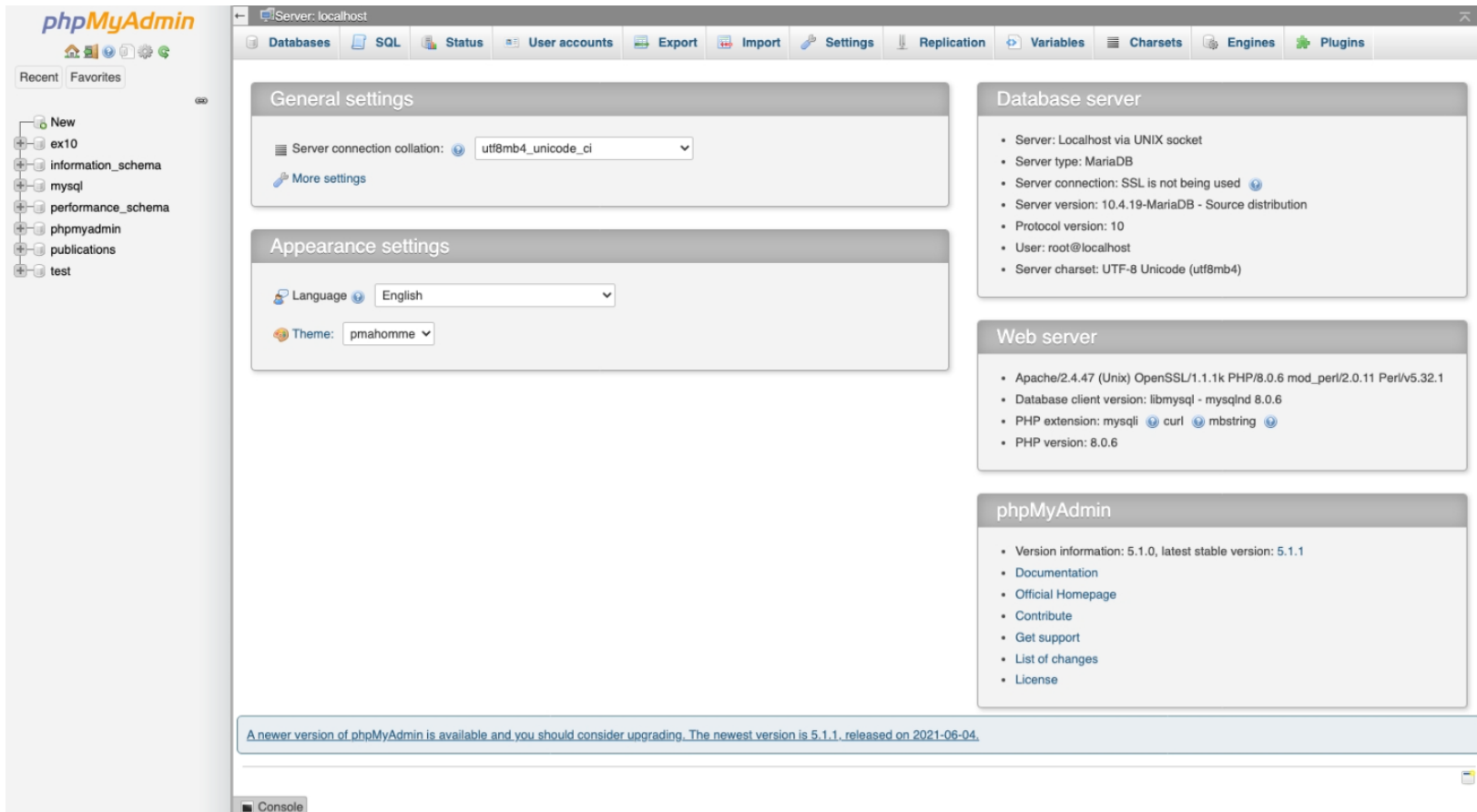
```
SELECT name, author, title FROM customers JOIN classics ON customers.isbn=classics.isbn;  
SELECT name, author, title from customers AS cust, classics AS class WHERE cust.isbn=class.isbn;
```



# Accessing MySQL via phpMyAdmin

- Simply access by <http://localhost/phpmyadmin/>

(Don't forget to run your server)



Break for 10 mins

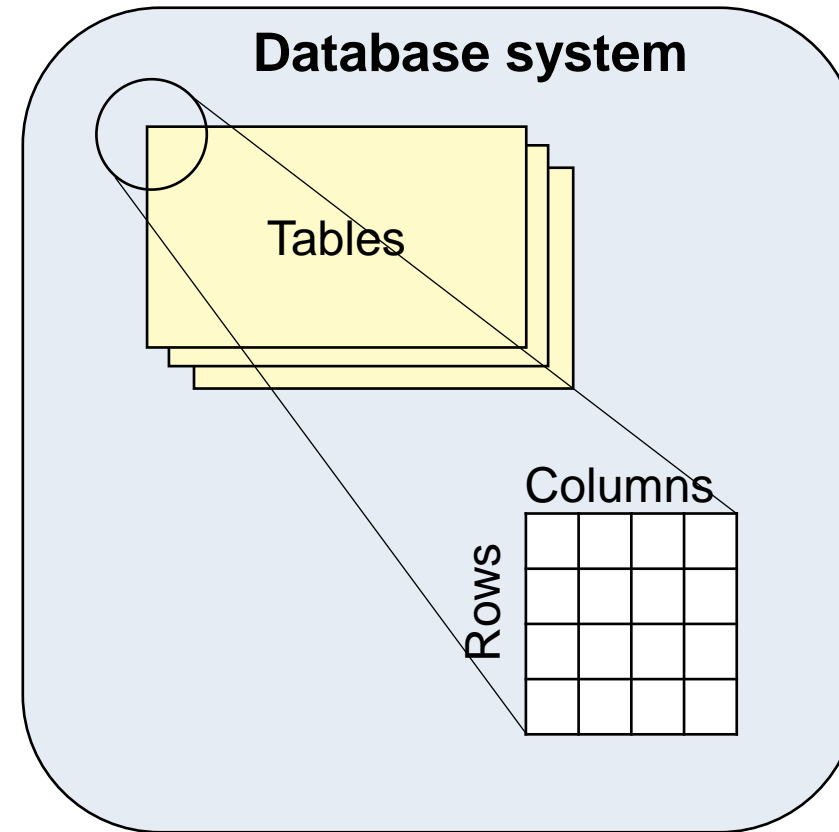
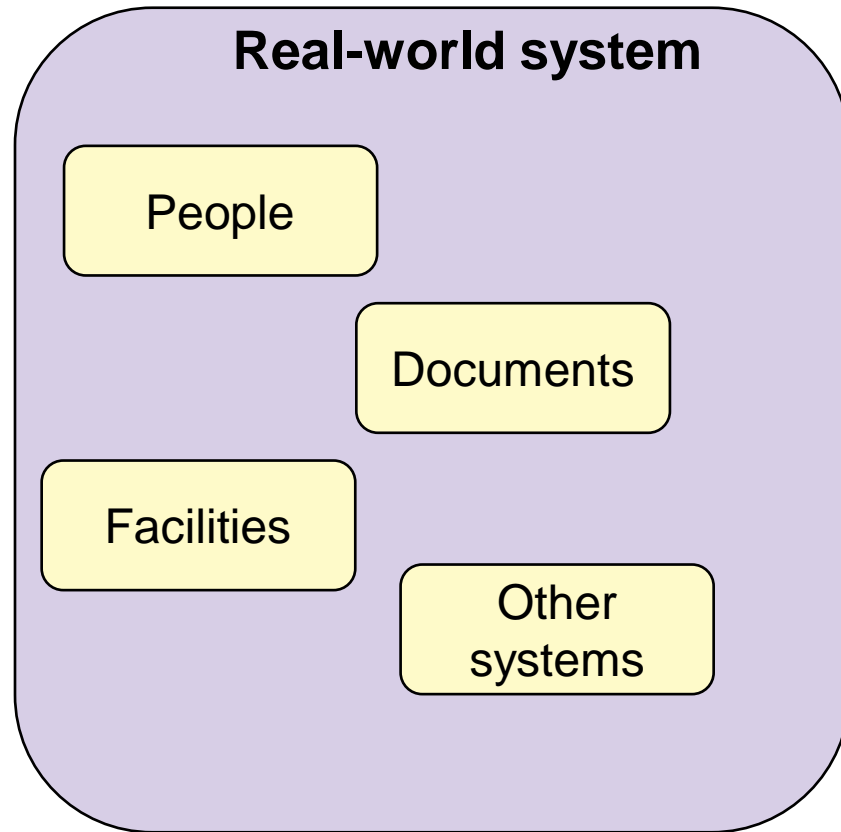
Until 11:00

try accessing your  
phpMyAdmin in localhost

**localhost/phpmyadmin**

# Database design

# Database system vs. real world-system



# Designing a data structure: steps

1. Identify the data elements
2. Subdivide each element into its smallest useful components
3. Identify the tables and assign columns
4. Identify the primary and foreign keys
5. Review whether the data structure is normalized
6. Identify the indexes

# How to identify the data elements

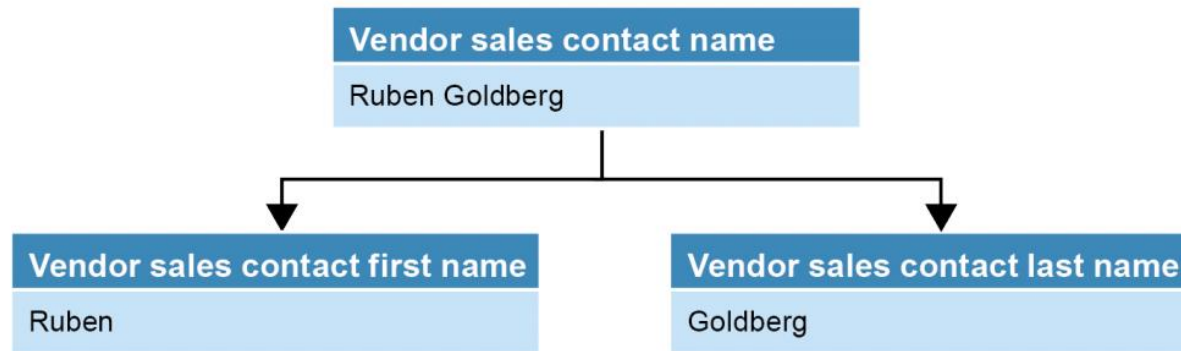
The data elements identified on the invoice document

- Vendor name
- Vendor address
- Vendor phone number
- Vendor fax number
- Vendor web address
- Invoice date
- Invoice terms
- Invoice number
- Item part number
- Item quantity
- Item description
- Item unit price
- Item extension
- Vendor sales contact name
- Vendor sales contact extension
- Vendor AR contact name
- Vendor AR contact extension
- Invoice total

Acme Fabrication, Inc.				
Custom Contraptions, Contrivances and Confabulations			Invoice Number:	101-1088
1234 West Industrial Way East Los Angeles California 90022			Invoice Date:	10/05/17
800.555.1212 fax 562.555.1213 www.acmefabrication.com			Terms:	Net 30
Part No.	Qty.	Description	Unit Price	Extension
CUST345	12	Design service, hr	100.00	1200.00
457332	7	Baling wire, 25x3ft roll	79.90	559.30
50173	4375	Duct tape, black, yd	1.09	4768.75
328771	2	Rubber tubing, 100ft roll	4.79	9.58
CUST281	7	Assembly, hr	75.00	525.00
CUST917	2	Testing, hr	125.00	250.00
		Sales Tax		245.20
Your salesperson: Ruben Goldberg, ext 4512			\$7,557.83	
Accounts receivable: Inigo Jones, ext 4901			PLEASE PAY THIS AMOUNT	
			Thanks for your business!	

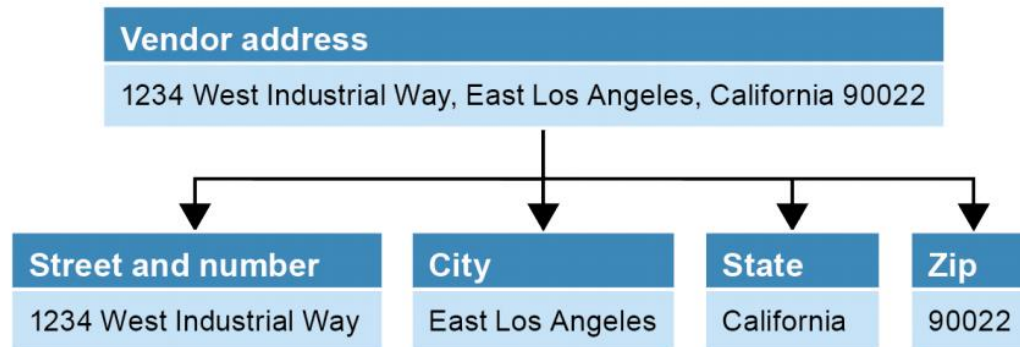
# How to subdivide the data elements

## A name that's divided into first and last names



- Easily perform operations like sorting by last name and using the first name in a salutation, such as “Dear Ruben.”
- If the full name is stored in a single column, string function will be necessary to do this --> inefficient code

## An address that's divided into street address, city, state, and zip code



- Note that street number and street name is stored together – typically used together
- Data elements should be divided into smallest *useful values*

# How to identify the tables and assign columns

## Possible tables and columns for an accounts payable system

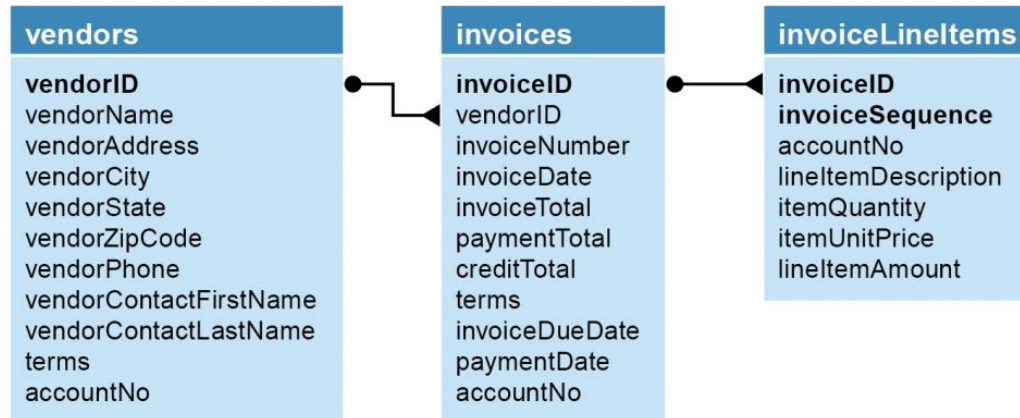
Vendors	Invoices	Invoice line items
Vendor name	Invoice number*	Invoice number*
Vendor address	Invoice date	<del>Item part number</del>
Vendor city	Terms*	Item quantity
Vendor state	Invoice total	Item description
Vendor zip code	<i>Payment date</i>	Item unit price
Vendor phone number	<i>Payment total</i>	Item extension
<del>Vendor fax number</del>	<i>Invoice due date</i>	<i>Account number*</i>
<del>Vendor web address</del>	<i>Credit total</i>	<i>Sequence number</i>
Vendor contact first name	<i>Account number*</i>	
Vendor contact last name		
<del>Vendor contact phone</del>		
<del>Vendor AR first name</del>		
<del>Vendor AR last name</del>		
<del>Vendor AR phone</del>		
<i>Terms*</i>		
<i>Account number*</i>		

- From the invoice, we can identify 3 entities
  - Vendors
  - Invoices
  - Invoice line items
- These entities → Tables
- Then determine which data element are associated with each entity
- Some element could be listed under 2 or more entities (add \*)
- These elements → Columns of the tables



# How to identify the primary and foreign keys

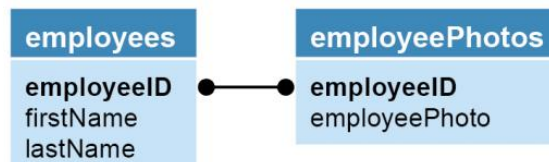
## The relationships between the tables in the accounts payable system



## Two tables with a many-to-many relationship



## Two tables with a one-to-one relationship



- **primary key** – uniquely identify each row in a table
  - **vendorName** could be primary key for **vendors** Table
  - Use ID is better, like **vendorID**
- **Composite key** – use two columns to identify each row
  - **invoiceID, invoiceSequence**
- **Foreign key** – the relationship between the tables
- For many to many relationship, linking table can be defined
- For two tables with 1-1 relationship: usually use when table contain column with large amount of data (such as photo files, etc)

# How to enforce the relationships between tables

## Operations that can violate referential integrity

This operation...	Violates referential integrity if...
Delete a row from the primary key table	The foreign key table contains one or more rows related to the deleted row
Insert a row in the foreign key table	The foreign key value doesn't have a matching primary key value in the related table
Update the value of a foreign key	The new foreign key value doesn't have a matching primary key value in the related table
Update the value of a primary key	The foreign key table contains one or more rows related to the row that's changed

# How normalization works

- Review whether the data structure is *normalized*
- How the data is separated into related tables
- The steps until now will ensure that your database is normalized (partially), however, it can be further normalized

# Normalization

## A table that contains repeating columns

vendorName	invoiceNumber	itemDescription_1	itemDescription_2	itemDescription_3
Cahners Publishing	112897	VB ad	SQL ad	Library directory
Zylka design	97/552	Catalogs	SQL flyer	NULL
Zylka design	97/553B	Card revision	NULL	NULL

Some problem caused by unnormalized data structure

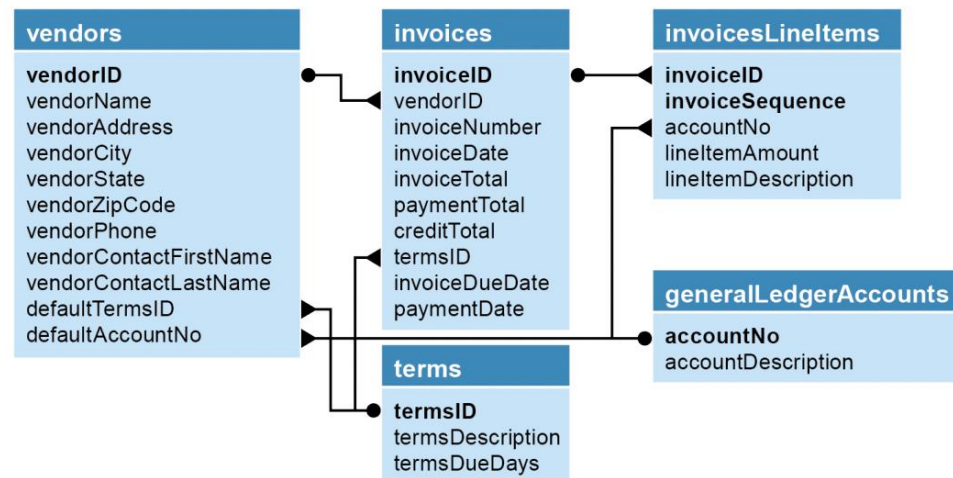
- Repeating itemDescription(1-3)
- Because an invoice can have many items, this itemDescription\_x will be repeated for the maximum number of line items (*data redundancy*)
- Waste storage space

## A table that contains redundant data

vendorName	invoiceNumber	itemDescription
Cahners Publishing	112897	VB ad
Cahners Publishing	112897	SQL ad
Cahners Publishing	112897	Library directory
Zylka design	97/522	Catalogs
Zylka design	97/522	SQL flyer
Zylka design	97/533B	Card revisions

- Repeated invoice number (*data redundancy*)
- Storage problem especially if the repeated column is large
- Cause maintenance problem if the column contain value that can change (change invoice Number, all rows need updated)

## The accounts payable system in third normal form



# How to identify the columns to be indexed

- When to create an index
  - When the column is a **foreign** key
  - When the column is used frequently in **search** conditions or **joins**
  - When the column contains a **large** number of **distinct** values
  - When the column is **updated infrequently**
- MySQL automatically create an index for a *primary* key
- Index – enable quick information location, no need to search through all the rows
- *Composite indexes* – 2+ columns (use when the columns are not updated frequently)
- Indexes should not be created more quantities than you need

# Normalization

# How to normalize a data structure

- The benefits of normalization
  - Normalized database has more tables, each table has index on its primary key, the database has more indexes → More efficient data retrieval
  - Each table contains information about a single entity, each index has fewer columns (usually one) and fewer rows. → More efficient data retrieval and insert, update, and delete.
  - Minimized data redundancy, simplifies maintenance and reduces storage.

# Normalization levels

- First Normal Form (1NF)
  - Each column is unique
- Second Normal Form (2NF)
  - Every non-key column must depend on the entire primary key
- Third Normal Form (3NF)
  - Every non-key column must depend only on the primary key
  - 3NF is achieved → the database is normalized
- Boyce-Codd Normal Form (BCNF)
  - A non-key column cannot depend on another non-key column (prevent transitive dependencies)
- Fourth Normal Form (4NF)
  - A table must not have more than one multivalued dependency
- Fifth Normal Form (5NF)
  - Further splitting the data into smaller and smaller table until all redundancy is eliminated



# How to apply the first normal form

## The invoice data with a column that contains repeating values

vendorName	invoiceNumber	itemDescription
Cahners Publishing	112897	VB ad, SQL ad, Library directory
Zylka design	97/522	Catalogs, SQL Flyer
Zylka design	97/533B	Card revision

## The invoice data with repeating columns

vendorName	invoiceNumber	itemDescription_1	itemDescription_2	itemDescription_3
Cahners Publishing	112897	VB ad	SQL ad	Library directory
Zylka design	97/552	Catalogs	SQL flyer	NULL
Zylka design	97/553B	Card revision	NULL	NULL

## The invoice data in first normal form

vendorName	invoiceNumber	itemDescription
Cahners Publishing	112897	VB ad
Cahners Publishing	112897	SQL ad
Cahners Publishing	112897	Library directory
Zylka design	97/522	Catalogs
Zylka design	97/522	SQL flyer
Zylka design	97/533B	Card revisions

- Eliminated the repeating values and columns.
- One row for each line item.
- However, data redundancy.
- Can be solve by applying second normal form

# How to apply the second normal form

The invoice data in first normal form with keys added

invoiceID	vendorName	invoiceNumber	invoiceSequence	itemDescription
1	Cahners Publishing	112897	1	VB ad
1	Cahners Publishing	112897	2	SQL ad
1	Cahners Publishing	112897	3	Library directory
2	Zylka design	97/522	1	Catalogs
2	Zylka design	97/522	2	SQL flyer
3	Zylka design	97/533B	1	Card revision

Primary keys = invoiceID & invoiceSequence

Only ItemDescription depends on **Primary keys**  
*vendorName and invoiceNumber depends on **invoiceID***

Non-key columns

The invoice data in second normal form

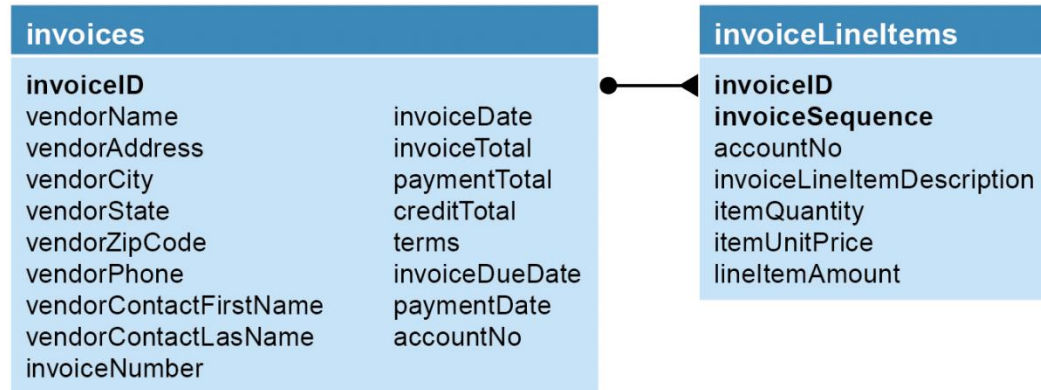
invoiceNumber	vendorName	invoiceID
11287	Cahners Publishing	1
97/522	Zylka design	2
97/533B	Zylka design	3

invoiceID	invoiceSequence	itemDescription
1	1	VB ad
1	2	SQL ad
1	3	Library directory
2	1	Catalogs
2	2	SQL flyer
3	1	Card revision

- Move the columns that does not depend on Primary keys to another table
- 2 Tables
  1. Info related to invoice
  2. Info related to individual line items
- The relationship between tables is based on InvoiceID
  - InvoiceID is primary key of Table#1
  - InvoiceID is foreign key of Table #2

# How to apply the third normal form

## The accounts payable system in second normal form

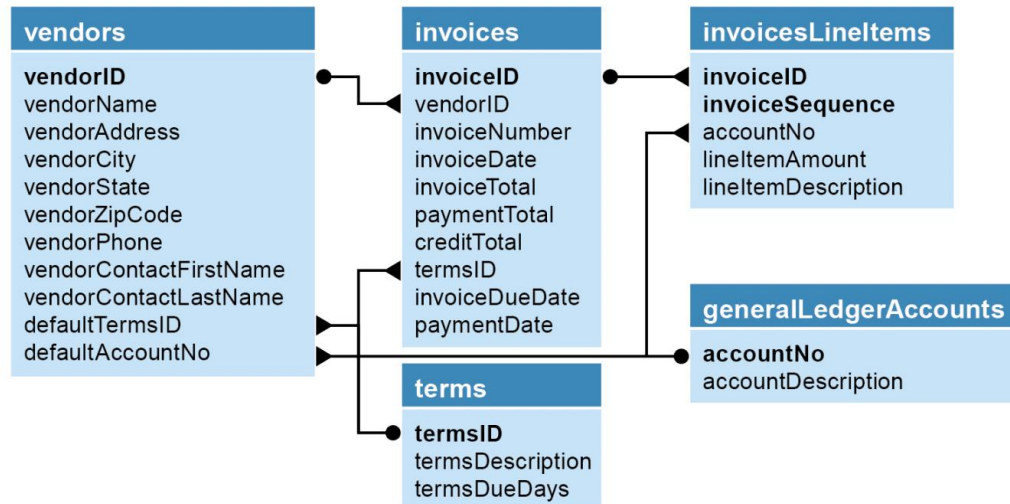


Every non-key column depends ONLY on the primary key

Questions about the structure

1. Does the vendor information (vendorName, vendorAddress, etc.) depend only on the invoiceID column? (if no, then vendor information should be in a separate table)
2. Does the terms column depend only on the invoiceID column? (No → new table)
3. Does the accountNo column depend only on the invoiceID column? (it depends on invoiceID and invoiceSequence → go to invoicesLineItems table)
4. Can the invoiceDueDate and lineItemAmount columns be derived from other data? (lineItemAmount can be calculated from other data → omit the column, InvoiceDueDate can also be calculated from invoiceDate and termDueDays, which could be overridden → should not omit)

## The accounts payable system in third normal form



The solution should depend on how the data will be used

Another example of  
normalization

# Inefficient design of a database table

Author 1	Author 2	Title	ISBN	Price	Customer name	Customer address	Purchase date
David Sklar	Adam Trachtenberg	PHP Cookbook	0596101015	44.99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Danny Goodman		Dynamic HTML	0596527403	59.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Hugh E. Williams	David Lane	PHP and MySQL	0596005436	44.95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
David Sklar	Adam Trachtenberg	PHP Cookbook	0596101015	44.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Rasmus Lerdorf	Kevin Tatroe & Peter MacIntyre	Programming PHP	0596006815	39.99	David Miller	3647 Cedar Lane, Waltham, MA 02154	Jan 16 2009

- A single table containing all of the author names, book titles, and (fictional) customer details.
- Obviously this is an inefficient design, because data is duplicated all over the place (duplications are highlighted), but it represents a starting point.

# First normal form

- For a database to satisfy the First Normal Form , it must fulfill three requirements:
  1. There should be no repeating columns containing the same kind of data.
  2. All columns should contain a single value.
  3. There should be a primary key to uniquely identify each row.

Title	ISBN	Price	Customer name	Customer address	Purchase date
PHP Cookbook	0596101015	44.99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Dynamic HTML	0596527403	59.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
PHP and MySQL	0596005436	44.95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
PHP Cookbook	0596101015	44.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Programming PHP	0596006815	39.99	David Miller	3647 Cedar Lane, Waltham, MA 02154	Jan 16 2009

Removed author columns and put in new table

Table 9-3. The new Authors table	
ISBN	Author
0596101015	David Sklar
0596101015	Adam Trachtenberg
0596527403	Danny Goodman
0596005436	Hugh E. Williams
0596005436	David Lane
0596006815	Rasmus Lerdorf
0596006815	Kevin Tatroe
0596006815	Peter MacIntyre



# Second normal form

- The Second Normal Form is all about redundancy across multiple rows.
- Identifying columns whose data repeats in different places and then removing them to their own tables.

*Table 9-4. The new Titles table*

ISBN	Title	Price
0596101015	PHP Cookbook	44.99
0596527403	Dynamic HTML	59.99
0596005436	PHP and MySQL	44.95
0596006815	Programming PHP	39.99

*Table 9-5. The customer details from Table 9-2*

ISBN	Customer name	Customer address	Purchase date
0596101015	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
0596527403	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
0596005436	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
0596101015	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
0596006815	David Miller	3647 Cedar Lane, Waltham, MA 02154	Jan 16 2009

*Table 9-6. The new Customers table*

CustNo	Name	Address	City	State	Zip
1	Emma Brown	1565 Rainbow Road	Los Angeles	CA	90014
2	Darren Ryder	4758 Emily Drive	Richmond	VA	23219
3	Earl B. Thurston	862 Gregory Lane	Frankfort	KY	40601
4	David Miller	3647 Cedar Lane	Waltham	MA	02154

*Table 9-7. The new Purchases table*

CustNo	ISBN	Date
1	0596101015	Mar 03 2009
2	0596527403	Dec 19 2008
2	0596101015	Dec 19 2008
3	0596005436	Jun 22 2009
4	0596006815	Jan 16 2009

# Third normal form

- Data that **is not** directly dependent on the primary key but **is dependent** on another value in the table should be moved into separated tables

Table 9-8. Third Normal Form Customers table

CustNo	Name	Address	Zip
1	Emma Brown	1565 Rainbow Road	90014
2	Darren Ryder	4758 Emily Drive	23219
3	Earl B. Thurston	862 Gregory Lane	40601
4	David Miller	3647 Cedar Lane	02154

Table 9-10. Third Normal Form Cities table

CityID	Name	StateID
1234	Los Angeles	5
5678	Richmond	46
4321	Frankfort	17
8765	Waltham	21

Table 9-9. Third Normal Form Zip codes table

Zip	CityID
90014	1234
23219	5678

Table 9-11. Third Normal Form States table

StateID	Name	Abbreviation
5	California	CA
46	Virginia	VA
17	Kentucky	KY
21	Massachusetts	MA



# When not to use normalization

- You should never fully normalize your tables on sites that will cause MySQL to trash
- Multiple tables = multiple calls to MySQL for each query

# Relationships

# Relationships

- MySQL is a relational database management system – tables store not only data but the relationships among the data
  1. One-to-one
  2. One-to-many
  3. Many-to-many

# One-to-one

- only one customer lives at each address, and each address can have only one customer.

*Table 9-8. Third Normal Form Customers table*

CustNo	Name	Address	Zip
1	Emma Brown	1565 Rainbow Road	90014
2	Darren Ryder	4758 Emily Drive	23219
3	Earl B. Thurston	862 Gregory Lane	40601
4	David Miller	3647 Cedar Lane	02154

*Table 9-8a (Customers)*

CustNo	Name
1	Emma Brown .....
2	Darren Ryder .....
3	Earl B. Thurston .....
4	David Miller.....

*Table 9-8b (Addresses)*

Address	Zip
1565 Rainbow Road	90014
4758 Emily Drive	23219
862 Gregory Lane	40601
3647 Cedar Lane	02154

*Figure 9-1. The Customers table, Table 9-8, split into two tables*

# One-to-Many

- When one row in one table is linked to many rows in another table
- Ex: there is only one of each customer in Customers table but one customer can have more than one purchase

*Table 9-8a (Customers)*

CustNo	Name
1	Emma Brown .....
2	Darren Ryder .....
	(etc....) .....
3	Earl B. Thurston .....
4	David Miller .....

*Table 9-7. (Purchases)*

CustNo	ISBN	Date
1	0596101015	Mar 03 2009
2	0596527403	Dec 19 2008
2	0596101015	Dec 19 2008
3	0596005436	Jun 22 2009
4	0596006815	Jan 16 2009

To represent a one-to-many relationship in a relational database, create a table for the “many” and a table for the “one.” The table for the “many” must contain a column that lists the primary key from the “one” table. Thus, the Purchases table will contain a column that lists the primary key of the customer.

# Many-to-Many

- Many rows in one table links to many rows in another table
- Need a third table containing the same key column from both

Columns from  
Table 9-8  
(Customers)

Intermediary  
Table 9-12  
(Customer/ISBN)

Columns from  
Table 9-4  
(Titles)

Zip	CustNo	CustNo	ISBN	ISBN	Title
90014	1 .....	1	0596101015	0596101015	PHP Cookbook
23219	2 .....	2	0596101015	(etc...)	
(etc...)	..... 2	2	0596527403		
40601	3 .....	3	0596005436	0596005436	Dynamic HTML
02154	4 .....	4	0596006815	0596006815	PHP and MySQL
					Programming PHP

With this intermediary table in place, it is possible to traverse all the info in the DB through a series of relations.

For example – find out purchases in 23219 zip code, look at Table 9-8 Customers, then see customers 2, then look up customer name, or look at Table 9-12 to see what is purchased

Figure 9-3. Creating a many-to-many relationship via a third table

# Transaction

# Transactions

- In some applications, it is important that a sequence of queries runs in the correct order and **every single query successfully** completes
- For example, transfer money from one bank account to another
  - The update (subtract fund from first account) fails, and now both accounts have the funds.
  - You subtract the funds from the first bank account, but the update request to add them to the second account fails, and the funds have disappeared into thin air.
- How to keep track of all these?
- MySQL comes with powerful transaction-handling features



# Transaction Storage Engines

- MySQL transaction facility can be used by InnoDB Storage engine (usually default)
- Or force it to use when create a table like this
- **Let's do it together**
- I created a database called ex10 here

```
MariaDB [(none)]> use ex09;  
Database changed  
MariaDB [ex09]> create table accounts(number INT, balance  
FLOAT, PRIMARY KEY(number)) ENGINE innodb;  
Query OK, 0 rows affected (0.011 sec)
```

```
MariaDB [ex09]> describe accounts;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| number | int(11) | NO | PRI | NULL | |  
| balance | float | YES | | NULL | |  
+-----+-----+-----+-----+-----+-----+  
2 rows in set (0.018 sec)
```

```
MariaDB [ex09]>
```

# Let's practice

- Now let's create two rows to practice using transaction

```
MariaDB [ex09]> INSERT INTO accounts(number, balance) VALUES(12345,1025.50);  
Query OK, 1 row affected (0.001 sec)
```

```
MariaDB [ex09]> INSERT INTO accounts(number, balance) VALUES(67890,140.00);  
Query OK, 1 row affected (0.001 sec)
```

```
MariaDB [ex09]> select * from accounts;
```

```
+-----+-----+  
| number | balance |  
+-----+-----+  
| 12345  | 1025.5  |  
| 67890  | 140     |  
+-----+-----+
```

```
2 rows in set (0.000 sec)
```

```
MariaDB [ex09]>
```

# BEGIN and COMMIT

```
MariaDB [ex09]> BEGIN;
Query OK, 0 rows affected (0.000 sec)

MariaDB [ex09]> UPDATE accounts SET
balance=balance+25.11 WHERE number=12345;
Query OK, 1 row affected (0.000 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [ex09]> COMMIT;
Query OK, 0 rows affected (0.001 sec)

MariaDB [ex09]> SELECT * FROM accounts;
+-----+-----+
| number | balance |
+-----+-----+
| 12345  | 1050.61 |
| 67890  | 140     |
+-----+-----+
2 rows in set (0.000 sec)

MariaDB [ex09]>
```

- Transactions in MySQL start with BEGIN
- COMMIT – until MySQL receive COMMIT, the change you make is temporary
- Possible to cancel a transaction by issuing ROLLBACK command instead

# COMMIT and ROLLBACK

```
MariaDB [ex09]> BEGIN; UPDATE accounts SET balance=balance-250 WHERE number=12345;  
Query OK, 0 rows affected (0.000 sec)
```

```
Query OK, 1 row affected (0.001 sec)  
Rows matched: 1 Changed: 1 Warnings: 0
```

```
MariaDB [ex09]> UPDATE accounts SET balance=balance+250 WHERE number=67890; SELECT  
* FROM accounts;
```

```
Query OK, 1 row affected (0.000 sec)  
Rows matched: 1 Changed: 1 Warnings: 0
```

number	balance
12345	800.61
67890	390

→ 250 less than before  
→ Increased by 250

2 rows in set (0.000 sec)

Assume something went wrong and  
you wish to undo this transaction

```
MariaDB [ex09]> rollback;  
Query OK, 0 rows affected (0.007 sec)
```

```
MariaDB [ex09]> select * from accounts;
```

number	balance
12345	1050.61
67890	140

2 rows in set (0.000 sec)

# Transaction Storage Engines (cont.)

- EXPLAIN can be used to investigate how the queries you issued to it are interpreted

```
MariaDB [ex09]> EXPLAIN SELECT * FROM accounts WHERE number=12345;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	accounts	const	PRIMARY	PRIMARY	4	const	1	

```
1 row in set (0.005 sec)
```

# Backing up and restoring

- Beneficial when you need to migrate your database to a new server
- Need to test backups from time to time to ensure they are valid and still working
- Can be done using `mysqldump` command (outside of Mysql command line)
- Must make sure no one writes to a table while backing up (shut down MySQL server before running `mysqldump`, or lock the tables while running `mysqldump`)
- Can be saved into .csv using >

```
mysqldump -u root -p test > mydatabase.sql
```

For XAMPP, your username = root, password = (nothing)

# Restoring from a backup file

- Passing the file to restore from using `<` symbol.
- To restore a single database, use the `-D` option

```
mysql -u root -p < mydatabase.sql
```

# Example of using `mysqldump`

```
MariaDB [ex09]> explain select * from accouns where nu  
mber = 12345;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	accouns	const	PRIMARY	PRIMARY	4	const	1	

```
1 row in set (0.000 sec)
```

```
MariaDB [ex09]> exit  
Bye
```

```
C:\xampp\mysql\bin>mysqldump -u root -p test > mydatabase.sql  
Enter password:
```



# End of the topic

---

MySQL