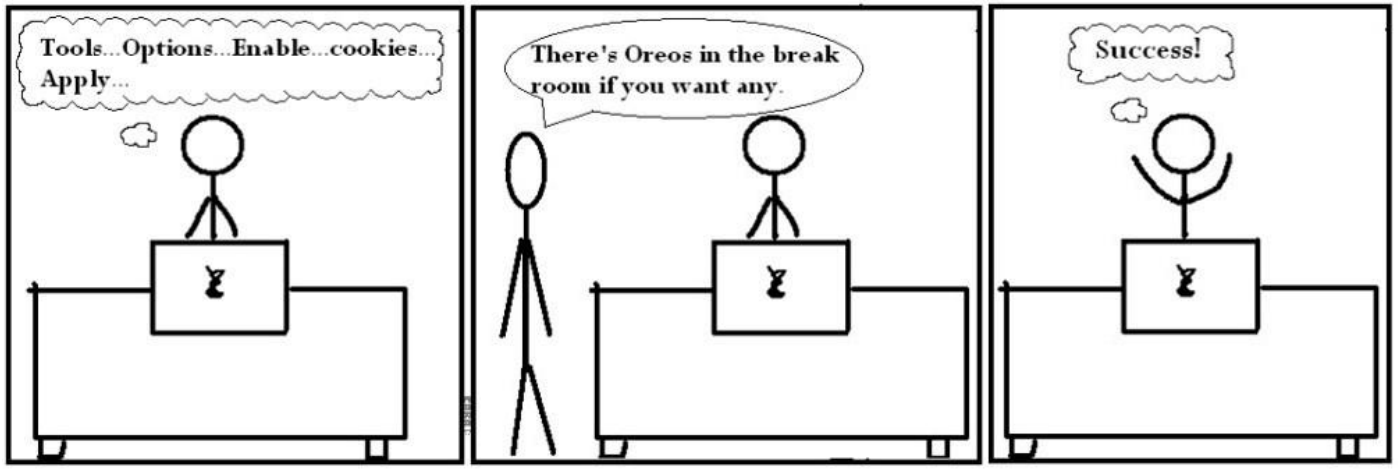
CSE 154



LECTURE 22:RELATIONAL DATABASES AND SQL

Relational databases

- **relational database**: A method of structuring data as tables associated to each other by shared attributes.
- a table row corresponds to a unit of data called a **record**; a column corresponds to an attribute of that record
- relational databases typically use **Structured Query Language** (SQL) to define, manage, and search data

Why use a database?

- **powerful**: can search it, filter data, combine data from multiple sources
- **fast**: can search/filter a database very quickly compared to a file
- **big**: scale well up to very large data sizes
- **safe**: built-in mechanisms for failure recovery (e.g. **transactions**)
- **multi-user**: concurrency features let many users view/edit data at same time
- **abstract**: provides layer of abstraction between stored data and app(s)
  - many database programs understand the same SQL commands

Database software

- Oracle
- Microsoft SQL Server (powerful) and Microsoft Access (simple)
- PostgreSQL (powerful/complex free open-source database system)
- SQLite (transportable, lightweight free open-source database system)
- MySQL (simple free open-source database system)
  - many servers run "LAMP" (Linux, Apache, MySQL, and PHP)
  - Wikipedia is run on PHP and MySQL
  - we will use MySQL in this course

Example simpsons database

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

id	name
1234	Krabappel
5678	Hoover
9012	Obourn

teachers

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 154	9012
10004	Informatics 100	1234

courses

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

- to test queries on this database, use username `homer`, password `d0ughnut`

Example world database

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...	...	...	...	...	...	...	...

countries (Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, ca pital, code2)

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...	...	...	...	...

cities

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...	...	...	...

languages

- to test queries on this database, use username `traveler`, password `packmybags`

Example imdb database

id	first_name	last_name	gender
433259	William	Shatner	M
797926	Britney	Spears	F
831289	Sigourney	Weaver	F
...			

actors

id	name	year	rank
112290	Fight Club	1999	8.5
209658	Meet the Parents	2000	7
210511	Memento	2000	8.7
...			

movies

actor_id	movie_id	role
433259	313398	Capt. James T. Kirk
433259	407323	Sgt. T.J. Hooker
797926	342189	Herself
...		

roles

movie_id	genre
209658	Comedy
313398	Action
313398	Sci-Fi
...	

movies_genres

id	first_name	last_name
24758	David	Fincher
66965	Jay	Roach
72723	William	Shatner
...		

directors

director_id	movie_id
24758	112290
66965	209658
72723	313398
...	

movies_directors

- also available, `imdb_small` with fewer records (for testing queries)
- to test queries on this database, use the username/password that we will email to you soon

SQL basics

```SQL
SELECT name FROM cities WHERE id = 17;
```

```SQL
INSERT INTO countries VALUES ('SLD', 'ENG', 'T', 100.0);
```

- **Structured Query Language (SQL)**: a language for searching and updating a database
- a standard syntax that is used by all database software (with minor incompatibilities)
  - generally case-insensitive
- a **declarative** language: describes what data you are seeking, not exactly how to find it

The SQL SELECT statement

```SQL
SELECT column(s) FROM table;
```

```SQL
SELECT name, code FROM countries;
```

name	code
China	CHN
United States	IND
Indonesia	USA
Brazil	BRA
Pakistan	PAK
…	…

- the SELECT statement searches a database and returns a set of results
- the column name(s) written after SELECT filter which parts of the rows are returned
- table and column names are case-sensitive

The DISTINCT modifier

```PHP
SELECT DISTINCT column(s) FROM table;
```

- eliminates duplicates from the result set

```SQL
SELECT language
FROM languages;
```

language
Dutch
English
English
Papiamento
Spanish
Spanish
Spanish
...

```SQL
SELECT DISTINCT language
FROM languages;
```

language
Dutch
English
Papiamento
Spanish
...

The WHERE clause

```
SELECT column(s) FROM table WHERE condition(s);                SQL
```

```
SELECT name, population FROM cities WHERE country_code = "FSM";
```

name	population
Weno	22000
Palikir	8600

- `WHERE` clause filters out rows based on their columns' data values
- in large databases, it's critical to use a `WHERE` clause to reduce the result set size
- suggestion: when trying to write a query, think of the `FROM` part first, then the `WHERE` part, and lastly the `SELECT` part

More about the WHERE clause

```
WHERE column operator value(s)                                    SQL
```

```
SELECT name, gnp FROM countries WHERE gnp > 2000000;              SQL
```

- the `WHERE` portion of a SELECT statement can use the following operators:
  - `=`, `>`, `>=`, `<`, `<=`
  - `<>` : not equal
  - `BETWEEN` *min* `AND` *max*
  - `LIKE` *pattern*
  - `IN` (*value*, *value*, ..., *value*)

code	name	gnp
JPN	Japan	3787042.00
DEU	Germany	2133367.00
USA	United States	8510700.00
...	...	...

Multiple WHERE clauses: AND, OR

```
SELECT * FROM cities WHERE code = 'USA' AND population >= 2000000;
```

id	name	country_code	district	population
3793	New York	USA	New York	8008278
3794	Los Angeles	USA	California	3694820
3795	Chicago	USA	Illinois	2896016
…	…	…	…	…

- multiple `WHERE` conditions can be combined using `AND` and `OR`

Approximate matches: LIKE

```
WHERE column LIKE pattern
```
SQL

```
SELECT code, name, population FROM countries WHERE name
LIKE 'United%';
```
SQL

code	name	population
ARE	United Arab Emirates	2441000
GBR	United Kingdom	59623400
USA	United States	278357000
UMI	United States Minor Outlying Islands	0

- `LIKE 'text%'` searches for text that starts with a given prefix
- `LIKE '%text'` searches for text that ends with a given suffix
- `LIKE '%text%'` searches for text that contains a given substring

Sorting by a column: ORDER BY

```
ORDER BY column(s)                                    SQL
```

```
SELECT code, name, population FROM countries
WHERE name LIKE 'United%' ORDER BY population;        SQL
```

code	name	population
UMI	United States Minor Outlying Islands	0
ARE	United Arab Emirates	2441000
GBR	United Kingdom	59623400
USA	United States	278357000

- can write ASC or DESC to sort in ascending (default) or descending order:

```
SELECT * FROM countries
ORDER BY population
DESC;                                                 SQL
```

- can specify multiple orderings in decreasing order of significance:

```
SELECT * FROM countries ORDER BY population DESC, gnp;  SQL
```

Limiting rows: LIMIT

```
LIMIT number                                                    SQL
```

```
SELECT name FROM cities WHERE name LIKE 'K%' LIMIT 5;   SQL
```

name
Kabul
Khulna
Kingston upon Hull
Koudougou
Kafr al-Dawwar

- can be used to get the top-N of a given category (`ORDER BY` and `LIMIT`)
- also useful as a sanity check to make sure your query doesn't return 10^7 rows

Querying a Database in PHP with PDO

```
$name = new PDO("dbprogram:dbname=database;host=server",
username, password);
$name->query("SQL query");
```
PHP

```
# connect to world database on local server
$db = new PDO("mysql:dbname=world;host=localhost",
"traveler", "packmybags");
$db->query("SELECT * FROM countries WHERE population >
100000000;");
```

- PDO database library allows you to connect to many different database programs
  - replaces older, less versatile functions like `mysql_connect`
- PDO object's `query` function returns rows that match a query

Result rows: query

```php
$db = new PDO("dbprogram:dbname=database;host=server",
username, password);
$rows = $db->query("SQL query");
foreach ($rows as $row) {
  do something with $row;
}
```

- `query` returns all result rows
  - each row is an associative array of [column name -> value]
  - example: `$row["population"]` gives the value of the `population` column

A complete example

```php
$db = new PDO("mysql:dbname=imdb_small", "jessica",
"guinness");
$rows = $db->query("SELECT * FROM actors WHERE last_name
LIKE 'Del%'");
foreach ($rows as $row) {
  ?>
  <li> First name: <?= $row["first_name"] ?>,
       Last name:  <?= $row["last_name"]  ?> </li>
  <?php
}
```

PHP

- First name: Benicio, Last name: Del Toro
- First name: Michael, Last name: Delano
- ...

output

Including variables in a query

```php
# get query parameter for name of movie
$title = $_GET["movietitle"];
$rows = $db->query("SELECT year FROM movies
WHERE name = '$title'");
```

- you should not directly include variables or query parameters in a query
- they might contain illegal characters or SQL syntax to mess up the query

Quoting variables

```php
# get query parameter for name of movie
$title = $_GET["movietitle"];
$title = $db->quote($title);
$rows = $db->query("SELECT year FROM movies WHERE name =
$title");
```

- call PDO's `quote` method on any variable to be inserted
- `quote` escapes any illegal chars and surrounds the value with `'` quotes
- prevents bugs and security problems in queries containing user input

Database/query errors

```php
$db = new PDO("mysql:dbname=imdb_small", "jessica", "guinness");
$rows = $db->query("SEEELECT * FROM movies WHERE year = 2000");
                    # FALSE
```

- database commands can often fail (invalid query; server not responding; etc.)
- normally, PDO commands fail silently by returning `FALSE` or `NULL`
- but this makes it hard to notice and handle problems

Exceptions for errors

```php
$db = new PDO("mysql:dbname=imdb_small", "jessica", "guinness");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$rows = $db->query("SEEELECT * FROM movies WHERE year = 2000");
                    # kaboom!
```

- using `setAttribute`, you can tell PDO to throw (generate) a `PDOException` when an error occurs
- the exceptions will appear as error messages on the page output
- you can **catch** the exception to gracefully handle the error

Catching an exception

```php
try {
  statement(s);
} catch (ExceptionType $name) {
  code to handle the error;
}
```

- a `try/catch` statement attempts to run some code, but if it throws a given kind of exception, the program jumps to the `catch` block and runs that code to handle the error

Example with error checking

```php
try {
  $db = new PDO("mysql:dbname=imdb_small", "jessica",
                "guinness");
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $rows = $db->query("SEEELECT * FROM movies WHERE year = 2000");
  foreach ($rows as row) { ... }
} catch (PDOException $ex) {
  ?>
  <p>Sorry, a database error occurred. Please try again later.
  </p>
  <p>(Error details: <?= $ex->getMessage() ?>)</p>
  <?php
}
```

PHP

PDOStatement methods

The `$rows` variable returned by PDO's `query` method is technically not an array but an object of type `PDOStatement`. It can be `foreach`-ed over like an array, but it also has the following methods:

columnCount()	number of columns in the results
fetch()	return the next row from the results
fetchColumn(*number*)	return the next column from the results
rowCount()	number of rows returned by the query

```php
if ($db->rowCount() > 0) {
  $first_row = $db->fetch();
  ...
}
```