Agenda

- This week’s expected outcomes
- This week’s topics
- This week’s homework
- Upcoming deadlines
- Questions and answers
Week 2 Outcomes

- Use DDL to create tables and indices in a DBMS.
- Use SQL to extract rows that match given criteria.
- Create server-based scripts to interactively query a data source and display the resulting rows in an HTML page.
Week 2 Topics

• Databases
  – Modeling
  – Normalization
  – Querying

• PHP and databases
  – Querying databases
  – Displaying results

• Logging
Database Review

• Database
  – Tables
    • Rows (records)
    • Columns (fields, attributes)
  – Primary key
    • Column(s) that uniquely identify a row
    • Usually a single column *surrogate key* (i.e. an “id” field) that has no real world significance
## Database Review

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>DEPT_CODE</th>
<th>PROF_OFFICE</th>
<th>PROF_EXTENSION</th>
<th>PROF_DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>ACCT</td>
<td>KLR 229D</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
<tr>
<td>114</td>
<td>ACCT</td>
<td>KLR 211</td>
<td>4436</td>
<td>Ph.D</td>
</tr>
<tr>
<td>301</td>
<td>ACCT</td>
<td>KLR 244</td>
<td>4683</td>
<td>Ph.D</td>
</tr>
<tr>
<td>435</td>
<td>ART</td>
<td>BBG 185</td>
<td>2278</td>
<td>Ph.D</td>
</tr>
<tr>
<td>387</td>
<td>BIOL</td>
<td>AAK 230</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
</tbody>
</table>
# Database Review

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>DEPT_CODE</th>
<th>PROF_OFFICE</th>
<th>PROF_EXTENSION</th>
<th>PROF_DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>ACCT</td>
<td>KLR 229D</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
<tr>
<td>114</td>
<td>ACCT</td>
<td>KLR 211</td>
<td>4436</td>
<td>Ph.D</td>
</tr>
<tr>
<td>301</td>
<td>ACCT</td>
<td>KLR 244</td>
<td>4683</td>
<td>Ph.D</td>
</tr>
<tr>
<td>435</td>
<td>ART</td>
<td>BBG 185</td>
<td>2278</td>
<td>Ph.D</td>
</tr>
<tr>
<td>387</td>
<td>BIOL</td>
<td>AAK 230</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
</tbody>
</table>

A row (record, entity)
# Database Review

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>DEPT_CODE</th>
<th>PROF_OFFICE</th>
<th>PROF_EXTENSION</th>
<th>PROF_DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>ACCT</td>
<td>KLR 229D</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
<tr>
<td>114</td>
<td>ACCT</td>
<td>KLR 211</td>
<td>4436</td>
<td>Ph.D</td>
</tr>
<tr>
<td>301</td>
<td>ACCT</td>
<td>KLR 244</td>
<td>4683</td>
<td>Ph.D</td>
</tr>
<tr>
<td>435</td>
<td>ART</td>
<td>BBG 185</td>
<td>2278</td>
<td>Ph.D</td>
</tr>
<tr>
<td>387</td>
<td>BIOL</td>
<td>AAK 230</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
</tbody>
</table>

A column (attribute, field)
# Database Review

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>DEPT_CODE</th>
<th>PROF_OFFICE</th>
<th>PROF_EXTENSION</th>
<th>PROF_DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>ACCT</td>
<td>KLR 229D</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
<tr>
<td>114</td>
<td>ACCT</td>
<td>KLR 211</td>
<td>4436</td>
<td>Ph.D</td>
</tr>
<tr>
<td>301</td>
<td>ACCT</td>
<td>KLR 244</td>
<td>4683</td>
<td>Ph.D</td>
</tr>
<tr>
<td>435</td>
<td>ART</td>
<td>BBG 185</td>
<td>2278</td>
<td>Ph.D</td>
</tr>
<tr>
<td>387</td>
<td>BIOL</td>
<td>AAK 230</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
</tbody>
</table>

Primary key (uniquely identifies a row)
Database Review

– Relationship
  • An association between entities (can be in the same or different tables)
  • Have cardinalities (1:M, 1:1, M:M)

– Foreign key
  • An attribute in one table that is a primary key in another table.
  • Connects the entities in a relationship
## Database Review

<table>
<thead>
<tr>
<th>CLASS_CODE</th>
<th>CRS_CODE</th>
<th>CLASS_SECTION</th>
<th>CLASS_TIME</th>
<th>CLASS_ROOM</th>
<th>EMP_NUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10012</td>
<td>ACCT-211</td>
<td>1</td>
<td>MWF 8:00-8:50 a.m.</td>
<td>KLR 225</td>
<td>105</td>
</tr>
<tr>
<td>10013</td>
<td>ACCT-211</td>
<td>2</td>
<td>MWF 9:00-9:50 a.m.</td>
<td>KLR 225</td>
<td>105</td>
</tr>
<tr>
<td>10014</td>
<td>ACCT-211</td>
<td>3</td>
<td>TTh 2:30-3:45 p.m.</td>
<td>KLR 225</td>
<td>342</td>
</tr>
<tr>
<td>10015</td>
<td>ACCT-212</td>
<td>1</td>
<td>MWF 10:00-10:50 a.m.</td>
<td>KLR 240</td>
<td>301</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMP_NUM</th>
<th>DEPT_CODE</th>
<th>PROF_OFFICE</th>
<th>PROF_EXTENSION</th>
<th>PROF_DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>ACCT</td>
<td>KLR 229D</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
<tr>
<td>114</td>
<td>ACCT</td>
<td>KLR 211</td>
<td>4436</td>
<td>Ph.D</td>
</tr>
<tr>
<td>301</td>
<td>ACCT</td>
<td>KLR 244</td>
<td>4683</td>
<td>Ph.D</td>
</tr>
<tr>
<td>435</td>
<td>ART</td>
<td>BBG 185</td>
<td>2278</td>
<td>Ph.D</td>
</tr>
<tr>
<td>387</td>
<td>BIOL</td>
<td>AAK 230</td>
<td>8665</td>
<td>Ph.D</td>
</tr>
</tbody>
</table>
### EMP_NUM in PROFESSOR is a foreign key in CLASS One professor can teach many classes.
Database Review

- ERD Modeling
  - Represents relationships graphically
  - Many notations (Crow’s foot below)
Database Review

- ERD Modeling
  - Represents relationships graphically
  - Many notations (Crow’s foot belongs)

Q: how do you represent many-to-many relationships?
Database Review

– Normalization: eliminate redundancy and dependency

• **First normal form**: tabular format, no repeating groups, primary key identified, non-key attributes are dependent on primary key.

• **Second normal form**: In 1NF and no partial dependencies (no dependencies on just part of the key).

• **Third normal form**: 2NF and no transitive dependencies (no nonkey attribute is dependent on another nonkey attribute),
Database Review

– Normalization: eliminate redundancy and dependency

  • **First normal form**: tabular format, no repeating groups, primary key identified, non-key attributes are dependent on primary key.

  • **Second normal form**: In 1NF and no partial dependencies (no dependencies on just part of the key).

  • **Third normal form**: 2NF and no transitive dependencies (no nonkey attribute is dependent on another nonkey attribute),
## Database Review

### MySQL vs SQLite

<table>
<thead>
<tr>
<th>MySQL</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs as a server process</td>
<td>Runs as an embedded library</td>
</tr>
<tr>
<td>Data kept in files on the server, many</td>
<td>Data kept in a single flat file per database</td>
</tr>
<tr>
<td>databases per file</td>
<td></td>
</tr>
<tr>
<td>Difficult to set up and manage</td>
<td>Simple to set up and manage</td>
</tr>
<tr>
<td>Has separate users, permissions</td>
<td>Has a single user with all permissions</td>
</tr>
<tr>
<td>Suitable for production environments</td>
<td>Suitable for development environments</td>
</tr>
<tr>
<td>Supports a robust superset of SQL syntax</td>
<td>Supports most, but not all SQL syntax</td>
</tr>
</tbody>
</table>
**Database Review**

- MySQL vs SQLite

<table>
<thead>
<tr>
<th>MySQL</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs as a server process</td>
<td>Runs as an embedded library</td>
</tr>
<tr>
<td>Data kept in files on the server, many databases per file</td>
<td>Data kept in a single flat file per database</td>
</tr>
<tr>
<td>Difficult to set up and manage</td>
<td>Simple to set up and manage</td>
</tr>
<tr>
<td>Has separate users, permissions</td>
<td>Has a single user with all permissions</td>
</tr>
<tr>
<td>Suitable for production environments</td>
<td>Suitable for development environments</td>
</tr>
<tr>
<td>Supports a robust superset of SQL syntax</td>
<td>Supports most, but not all SQL syntax</td>
</tr>
</tbody>
</table>

If we are careful to use standard SQL, almost everything we write in PHP will be portable between these two database management systems.
Database Review

- SQL – CREATE TABLE

```sql
CREATE TABLE IF NOT EXISTS CLASS (  
    CLASS_CODE INTEGER PRIMARY KEY NOT NULL,  
    CRS_CODE VARCHAR(25) NOT NULL,  
    CLASS_SECTION INTEGER NOT NULL,  
    CLASS_TIME VARCHAR(30) NOT NULL,  
    CLASS_ROOM VARCHAR(15) NOT NULL,  
    EMP_NUM INTEGER NOT NULL,  
    FOREIGN KEY(CRS_CODE) REFERENCES COURSE(CRS_CODE),  
    FOREIGN KEY(EMP_NUM) REFERENCES EMPLOYEE(EMP_NUM)  
);```
CREATE TABLE IF NOT EXISTS CLASS (  CLASS_CODE INTEGER PRIMARY KEY NOT NULL,  CRS_CODE VARCHAR(25) NOT NULL,  CLASS_SECTION INTEGER NOT NULL,  CLASS_TIME VARCHAR(30) NOT NULL,  CLASS_ROOM VARCHAR(15) NOT NULL,  EMP_NUM INTEGER NOT NULL,  FOREIGN KEY(CRS_CODE) REFERENCES COURSE(CRS_CODE),  FOREIGN KEY(EMP_NUM) REFERENCES EMPLOYEE(EMP_NUM) );

These tables need to be created before CLASS is.
Database Review

• SQL – CREATE INDEX

CREATE INDEX IF NOT EXISTS STU_NAME_IDX ON
STUDENT (STU_LNAME, STU_FNAME);

Why create indices?
Database Review

- SQL – INSERT

```
INSERT INTO STUDENT (STU_NUM, STU_LNAME, STU_FNAME, STU_INITIAL, STU_DOB, STU_HRS, STU_CLASS, STU_GPA, STU_TRANSFER, DEPT_CODE, STU_PHONE, EMP_NUM)
VALUES (300245, 'Peppard', 'Randy', 'K', '6/21/1975 0:00:00', 45, 'So', 2.61, 0, 'BIOL', 2134, 387);
```
Database Review

**SQL – UPDATE**

```
UPDATE STUDENT SET
  STU_INITIAL = 'L'
WHERE
  STU_NUM = 300245;
```

Affects one row. Why?

```
UPDATE STUDENT SET
  STU_CLASS = 'So'
WHERE
  STU_HRS > 32 AND
  STU_HRS <= 64;
```

Could affect multiple rows.
Database Review

• SQL – DELETE

```
DELETE
FROM
  STUDENT
WHERE
  STU_NUM = 300245;
```
Database Review

• SQL – Simple SELECT

```sql
SELECT *
FROM
  STUDENT
WHERE
  STU_GPA > 3.5
ORDER BY
  STU_LNAME, STU_FNAME ASC
```
Database Review

• SQL – Complex SELECT

```
SELECT COURSE.CRS_CODE || '.' || CLASS.CLASS_SECTION AS COURSE,
    COURSE.CRS_DESCRIPTION AS DESCRIPTION
FROM COURSE, STUDENT, CLASS, ENROLL
WHERE
    STUDENT.STU_LNAME='Robertson' AND
    STUDENT.STU_FNAME='Anne' AND
    STUDENT.STU_NUM = ENROLL.STU_NUM AND
    ENROLL.CLASS_CODE = CLASS.CLASS_CODE AND
    CLASS.CRS_CODE = COURSE.CRS_CODE;
```
Database Review

• SQL – Complex SELECT

```sql
SELECT COURSE.CRS_CODE || '.' || CLASS.CLASS_SECTION AS COURSE,
  COURSE.CRS_DESCRIPTION AS DESCRIPTION
FROM COURSE, STUDENT, CLASS, ENROLL
WHERE
  STUDENT.STU_LNAME = 'Robertson' AND
  STUDENT.STU_FNAME = 'Anne' AND
  STUDENT.STU_NUM = ENROLL.STU_NUM AND
  ENROLL.CLASS_CODE = CLASS.CLASS_CODE AND
  CLASS.CRS_CODE = COURSE.CRS_CODE;
```
Database Review

• SQL – Complex SELECT

```sql
SELECT COURSE.CRS_CODE || '.' || CLASS.CLASS_SECTION AS COURSE,
       COURSE.CRS_DESCRIPTION AS DESCRIPTION
FROM COURSE, STUDENT, CLASS, ENROLL
WHERE
    STUDENT.STU_LNAME='Robertson' AND
    STUDENT.STU_FNAME='Anne'
    AND STUDENT.STU_NUM = ENROLL.STU_NUM
    AND ENROLL.CLASS_CODE = CLASS.CLASS_CODE
    AND CLASS.CRS_CODE = COURSE.CRS_CODE;
```
Tools for working with SQLite

• SQLiteExpert Personal
  – http://www.sqliteexpert.com
Tools for working with SQLite

• Documentation:
  – [http://www.sqlite.org/docs.html](http://www.sqlite.org/docs.html)

• Naming convention
  – SQLite 3 (what we’re using): file extension is .db3

• Quick demonstration after lecture
Using PHP with SQL

• PDO
  – PHP Data Objects
  – Abstracts the database into an object

```php
try {
    $db = new PDO('sqlite:MicroUniversity.db3');
} catch (PDOException $e) {
    include 'error.inc';
    errorPage("Database error", $e);
    exit();
}
```
Using PHP with SQL

• PDO
  – Abstracts the database into an object

```php
try {
    $db = new PDO('sqlite:MicroUniversity.db3');
} catch (PDOException $e) {
    include 'error.inc';
    errorPage("Database error", $e);
    exit();
}
```

The “DSN” (data source name) designates which database to use. MySQL requires authentication.
Using PHP with SQL

• PDO
  – PHP Data Objects
  – Abstracts the database as an object

```php
try {
    $db = new PDO('sqlite:MicroUniversity.db3');
} catch (PDOException $e) {
    include 'error.inc';
    errorPage("Database error", $e);
    exit();
}
```

Exception handling. “Try” to connect to the database. If an error is thrown, then “catch” it and handle it properly.
Using PHP with SQL


```php
<?php
include_once 'ui.inc';
include_once 'util.inc';

function errorPage($title, $exception) {
    head($title);
    print "<h1>$title</h1>
    print "<p>Problem: {$exception->getMessage()}.</p>"
    foot();
}
?>
```

`try`
```php
try {
    $db = new PDO('sqlite:MicroUniversity.db3');
} catch (PDOException $e) {
    include 'error.inc';
    errorPage("Database error", $e);
    exit();
}
```

Every exception has a message that can be printed.
Using PHP with SQL

- Universal way to execute SQL in PHP
  - Prepare a statement
  - Bind needed parameters
  - Execute
  - Fetch and return rows (if querying)
Using PHP with SQL

• Universal way to execute SQL in PHP
  – Prepare a statement
  – Bind needed parameters
  – Execute
  – Fetch and return rows (if

Prepared statements with bound parameters avoid a category of security vulnerabilities known as “SQL Injection,” attacks
Using PHP with SQL

- Universal way to execute SQL in PHP

```php
function findStudentByName($lname, $fname) {
    global $db;
    $lname = "%{$lname}%;
    $fname = "%{$fname}%;
    $st = $db -> prepare('SELECT * FROM STUDENT WHERE ' .
                       'ORDER BY STU_LNAME, STU_FNAME');
    $st -> bindParam(1, $lname);
    $st -> bindParam(2, $fname);
    $st -> execute();
    return $st ->fetchAll(PDO::FETCH_ASSOC);
}
```
Using PHP with SQL

• Universal way to execute SQL in PHP

```php
function findStudentByName($lname, $fname) {
    global $db;
    $lname = "%{\$lname}\%";
    $fname = "%{\$fname}\%";
    $st = $db -> prepare('SELECT * FROM STUDENT WHERE ' .
                          'STU_LNAME LIKE ? AND STU_FNAME LIKE?' .
                          'ORDER BY STU_LNAME, STU_FNAME',
                        PDO::PARAM_STR, PDO::PARAM_STR);
    $st -> bindParam(1, $lname);
    $st -> bindParam(2, $fname);
    $st -> execute();
    return $st ->fetchAll(PDO::FETCH_ASSOC);
}
```

This returns the resulting rows as an array of associative arrays whose keys are the database field names.
Using PHP with SQL

- Universal way to execute SQL in PHP

```php
function findStudentByName($lname, $fname) {
    global $db;
    $lname = "%{$lname}%";
    $fname = "%{$fname}%";
    $stmt = $db->prepare('SELECT * FROM STUDENT WHERE ' .
                         'ORDER BY STU_LNAME, STU_FNAME');
    $stmt->bindParam(1, $lname);
    $stmt->bindParam(2, $fname);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

This returns the resulting rows as an array of associative arrays whose keys are the database field names.
Using PHP with SQL

• Using result rows

```php
$rows = findStudentByName($lname, $fname);
print "<table><tr><th>First</th><th>Last</th></tr><tr>\n); 
foreach ($rows as $row) {
    print "<tr><td>{$row['STU_FNAME']}</td>
          <td>{$row['STU_LNAME']}</td></tr><tr>\n";
}
print "</table>";
```
Using PHP with SQL

- Using result rows

```php
$rows = findStudentByName($lname, $fname);
print "<table><tr><th>First</th><th>Last</th></tr>
";
foreach ($rows as $row) {
    print "<tr><td>{$row['STU_FNAME']}</td>
    <td>{$row['STU_LNAME']}</td></tr>
";
}
print "</table>";
```

**foreach** walks through every row of the result set.

Access data within a row by using the column name as an index.
Using PHP with SQL

• Using result rows

```php
<?php $rows = findStudentByName($lname, $fname) ?>
<table>
  <tr>
    <th>First</th>
    <th>Last</th>
  </tr>
  <?php foreach ($rows as $row) : ?>
  <tr>
    <td><?php echo $row['STU_FNAME']; ?></td>
    <td><?php echo $row['STU_LNAME']; ?></td>
  </tr>
  <?php endforeach; ?>
</table>
```

A more tag-oriented approach to tables.
Using PHP with SQL

• Simple walkthrough

Find a student

Find a student

First name:

Last name:

Submit
Using PHP with SQL

• Simple walkthrough

Find a student

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annelise</td>
<td>Paulus</td>
</tr>
<tr>
<td>Anne</td>
<td>Robertson</td>
</tr>
<tr>
<td>Anne</td>
<td>Smithson</td>
</tr>
</tbody>
</table>
Using PHP with SQL

• Simple walkthrough
Using PHP with SQL

• Show me the code!
  – Available from
    http://cs.franklin.edu/~sharkesc/webd236/
Logging class Lib/Logger.inc

```php
class Logger {
    private static $instance;
    private $filename;
    private $level;

    const DEBUG = 0;
    const INFO = 1;
    const WARN = 2;
    const ERROR = 3;

    private function __construct($level=self::DEBUG, $filename='debug.log') {
        // log everything
        $this -> level = $level;
        $this -> filename = $filename;
    }
}
```
Logging

- Logging class Lib/Logger.inc

```php
public static function instance($level=self::DEBUG, $filename='debug.log') {
    if (!isset(self::$instance)) {
        self::$instance = new Logger($this->level, $this->filename);
    }
    return self::$instance;
}

public function debug($message) {
    return $this->log(self::DEBUG, $message);
}
```
Logging

- Logging class Lib/Logger.inc

```php
public function setLevel($level) {
    $this -> level = $level;
}

public function setFilename($filename) {
    $this -> filename = $filename;
}
```
Logging

- Logging class Lib/Logger.inc

```php
public function info($message) {
    return $this -> log(self::INFO, $message);
}

public function warn($message) {
    return $this -> log(self::WARN, $message);
}

public function error($message) {
    return $this -> log(self::ERROR, $message);
}
```
Logging

• Logging class Lib/Logger.inc

```php
private function log($level, $message) {
    if ($level >= $this -> level) {
        $names = array('DEBUG', 'INFO', 'WARN', 'ERROR');
        $timestamp = date("Y-m-d H:i:s", time());
        $fd = fopen($this -> filename, "a");
        fprintf($fd, "%s %s %s\n",
            $timestamp, $names[$level], $message);
        fclose($fd);
    }
}
```
Upcoming Deadlines

• Readings for next week
  – Chapters 5 and 6 in *PHP and MySQL*

• Assignments
  – Homework 1 due end of week 2
  – Homework 2 due end of week 3
  – Lab 1 due end of week 4

• Next week:
  – Avoiding ugly URLs, MVC pattern, and testing/debugging
General Q & A

• Questions?
• Comments?
• Concerns?