WEBD 236

Web Information Systems Programming

Week 9

FRANKLIN UNIVERSITY

Agenda

- This week's expected outcomes
- This week's topics
- This week's homework
- Upcoming deadlines
- Solution to Lab 2 – reprise from last week
- Questions and answers

FRANKLIN UNIVERSITY

Week 9 Outcomes

- Design databases from real-world problem statements
- Normalize databases
- Employ SQL to create database tables and indices
- Employ PHP scripting to create and load tables with initial data

Database Design

- Six steps to database design
  - Identify the data elements
  - Subdivide each element into its smallest useful components
  - Identify the tables and assign columns
  - Identify primary and foreign keys (relationships)
  - Normalize
  - Identify indices

Database Design

- "Things" to remember
  - Each table represents a group of "things"
  - Each row within the table is one "thing" in the group
  - Each column within a row represents an attribute of that one "thing"

Database Design

- Example:
  - Consulting company connects people with the right skills to clients who need those people to work on specific projects
  - Need to know which employees have certain skills (and at what level), and assign them to a project for certain dates.
  - A client can have more than one project, but each project has a specific contact person (sponsor) within the client's organization.

Database Design

- Identify the major entities

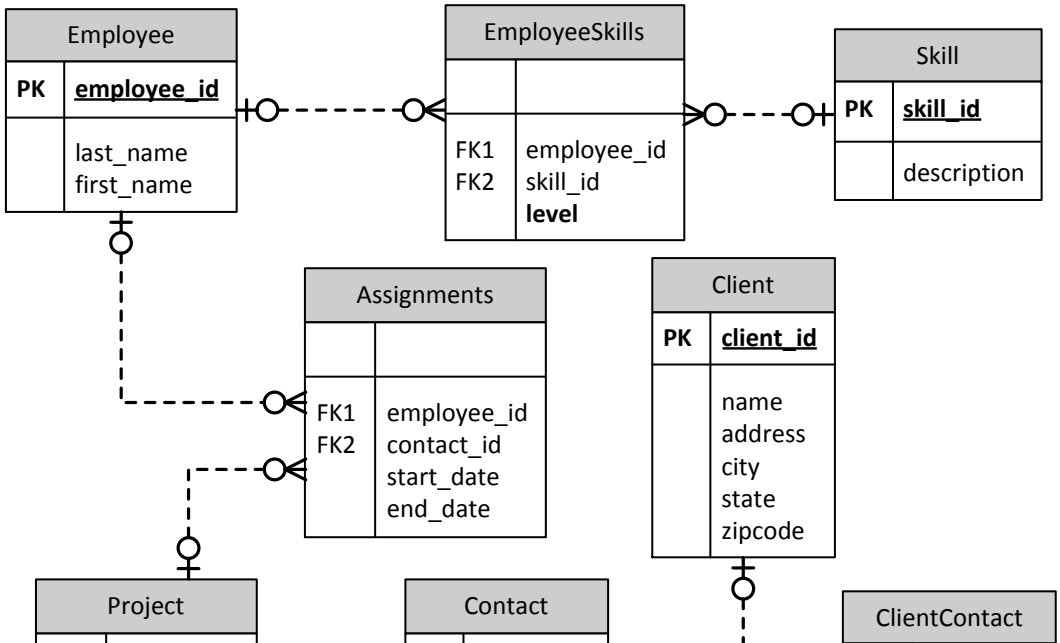
Database Design

- Identify the attributes of each entity

Database Design

- Draw the ERD

Database Design



Database Design

- Is the database normalized?
  - **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).

Database Design

- Is the database normalized?
  - **Third normal form**: 2NF and every non-key attribute must depend only on the primary key
  - **Boyce-Codd normal form:** (BCNF) a non-key attribute cannot depend on a non-key attribute (avoids transitive dependencies)
  - **Higher forms:** interesting, but not particularly useful for this course.

Database Design

- Is the database normalized?
  - If the DB is not in at least 3rd normal form, what good reason do you have for doing so?

Database Design

- What should we create indices for?
  - Primary keys
  - Foreign keys
  - Use frequently in search or join (see above)

Database Design

- Tools for modeling/designing databases
  - DB vendors usually provide excellent tools
    - MySQL: MySQL Workbench
    - Oracle: SQL Developer Data Modeler
    - Microsoft: Visio

You have access to this for free through MSDNAA: http://msdn07.e-academy.com/FU_DC

SQL for Creating Databases

- CREATE DATABASE – Needed in a multi-database server (MySQL), but not for SQLite.

```
CREATE DATABASE IF NOT EXISTS Contracting;
```

- USE – again, for servers w/ many DBs

```
USE Contracting;
```

- DROP DATABASE – irreversable

```
DROP DATABASE Contracting;
```

SQL for Creating Databases

- CREATE TABLE for Client

Client	
PK	**client_id**
	name address city state zipcode

SQL for Creating Databases

- CREATE TABLE for Client

```
CREATE TABLE IF NOT EXISTS Client {
  client_id INTEGER      PRIMARY KEY NOT NULL, /* auto-inc */
  name      VARCHAR(100) NOT NULL,
  address   VARCHAR(60)  NOT NULL,
  city      VARCHAR(30)  NOT NULL,
  state     CHAR(2)      NOT NULL,
  zipcode   VARCHAR(10)  NOT NULL
}
```

Other common, types:
DECIMAL, TEXT, CLOB, BLOB,
DATE, TIME, DATETIME

SQL for Creating Databases

- CREATE TABLE for EmployeeSkills

EmployeeSkills	
FK1 FK2	employee_id skill_id **level**

SQL for Creating Databases

- CREATE TABLE for EmployeeSkills

EmployeeSkills	
FK1 FK2	employee_id skill_id **level**

```
CREATE TABLE IF NOT EXISTS EmployeeSkills {
  employee_id INTEGER NOT NULL,
  skill_id    INTEGER NOT NULL,
  level       INTEGER NOT NULL,
  FOREIGN KEY(employee_id) REFERENCES Employee(employee_id),
  FOREIGN KEY(skill_id)    REFERENCES Skill(skill_id)
}
```

Remember, Employee and Skill need to exist before this command is executed.

SQL for Creating Databases

- DROP TABLE

```
DROP TABLE IF EXISTS Client;
```

- CREATE INDEX

```
CREATE UNIQUE INDEX email ON Contact(email);
```

- DROP INDEX

```
DROP INDEX email ON Contact;
```


SQL for Creating Databases

- Privileges
  - Individual users in the DB server can have rights to perform certain actions:
    - Globally
    - On certain databases
    - On certain tables within databases
    - On certain columns within tables
  - CREATE, ALTER, DROP, INDEX, SELECT, INSERT, UPDATE, DELETE

SQL for Creating Databases

- Privileges
  - Reality: at best two users at the DB level
    - One that can do everything
    - One that is read-only (for a data warehouse)
  - In a web application, privileges are usually enforced at the *application* level in *code*.
    - Approaches:
      - RBAC: role based access control
      - ACL: access control lists
      - DAC: discretionary access control

More on this in Ch. 21

Loading Initial Data

- Why load initial data
  - Need the DB to be in an initial state for the application to work
  - Need something to test interactively

```
Joe,O'Conner,joeo@example.com
John,Roberts,johnr@example.com
Bill,Kilbourne,billk@example.com
Ruth,Richardson,ruthr@example.com
```

`data.txt` file containing contact information to load

Loading Initial Data

```
class Db {
    protected static function getDb() {
        try {
            $db = new PDO('sqlite:contacts.db3');
        } catch (PDOException $e) {
            die("Could not open database. " .
                $e -> getMessage());
        }
        return $db;
    }

    // continued...
```


Loading Initial Data

```
class Db {
    public static function resetTable($class) {
        $reflector = new ReflectionClass($class);

        $db = self::getDb();
        $db -> beginTransaction();

        $db -> exec('DROP TABLE IF EXISTS '.
            $reflector -> getConstant('TABLE_NAME'));
        $db -> exec($reflector ->
            getConstant('CREATE_TABLE_SQL'));
        $db -> commit();
    }

    // continued...
```

Loading Initial Data

```
class Db {
    public static function resetTable($class) {
        $reflector = new ReflectionClass($class);

        $db = self::getDb();
        $db -> beginTransaction();

        $db -> exec('DROP TABLE
            $reflector -> getCon
        $db -> exec($reflector -
            getConstant('CREATE_
        $db -> commit();
    }

    // continued...
```

ReflectionClass lets you find out information about a class at runtime (i.e. a list of methods, properties, constants, etc.)

Loading Initial Data

```
class Db {
    public static function loadInitial($fileName, $class) {

        $fp = @fopen($fileName, "r");
        if ($fp) {
            while (($line = fgets($fp)) !== false) {
                $args = explode(",", $line);
                $reflector = new ReflectionClass($class);
                $object = $reflector -> newInstanceArgs($args);
                $object -> insert();
            }
            fclose($fp);
        }
    }

    // continued...
```

Loading Initial Data

```
class Db {
    public static function loadInitial($f

        $fp = @fopen($fileName, "r");
        if ($fp) {
            while (($line = fgets($fp)) !== false) {
                $args = explode(",", $line);
                $reflector = new Refl tionClass($class);
                $object = $reflector -> n nstanceArgs($args);
                $object -> insert();
            }
            fclose($fp);
        }
    }

    // continued...
```

fopen opens a file (duh).

fgets reads a string from the file up to a newline or eof.

Loading Initial Data

```
class Model extends Db {
    protected $id;

    public function __construct($id = null) {
        if ($id) {
            $this -> setId($id);
        }
    }
    public function getId() {
        return $this -> id;
    }
    public function setId($id) {
        $this -> id = $id;
        return $this;
    }
}
```

Loading Initial Data

```
class Contact extends Model {
    protected $first_name;
    protected $last_name;
    protected $email;

    const TABLE_NAME = 'Contacts';

    const CREATE_TABLE_SQL = '
      CREATE TABLE Contacts (
        contact_id INTEGER PRIMARY KEY,
        first_name VARCHAR(20) NOT NULL,
        last_name  VARCHAR(30) NOT NULL,
        email VARCHAR(50) NOT NULL
      );';

 // continued...
```


Loading Initial Data

```
class Contact extends Model {

   const INSERT_SQL = '
      INSERT INTO Contacts
        (first_name, last_name, email)
      VALUES
        (:first_name, :last_name, :email)';

    public function __construct($first, $last, $email) {
        $this -> first_name = $first;
        $this -> last_name = $last;
        $this -> email = $email;
    }

 // continued...
```

Loading Initial Data

```
class Contact extends Model {

    public function insert() {
        $db = self::getDb();
        $st = $db -> prepare(self::INSERT_SQL);
        $st -> bindParam(":first_name", $this -> first_name);
        $st -> bindParam(":last_name", $this -> last_name);
        $st -> bindParam(":email", $this -> email);
        $st -> execute();
        $this -> id = $db -> lastInsertId();
    }
}

// and then somewhere...
Db::resetTable("Contact");
Db::loadInitial("data.txt", "Contact");
```


Solution to Lab 2

General Q & A

- Questions?
- Comments?
- Concerns?