

# You Don't Need a DBA

or

What Every PHP Developer Should  
Know about Database Development

Maggie Nelson  
phplworks 2007



Database Developer



DBA

PHP Developer



System Administrator

=



**DB Developer:** “Help!  
My database script just  
did truncate on the  
users table!”

---

**DBA:** \*rolls eyes and  
recovers data\*

**PHP Developer:** “Help!  
My PHP script just did  
exec(‘rm -rf /’)!”

---

**SA:** \*rolls eyes and  
recovers data\*



# What do we ask DBAs for?

- “Can you add some indexes so my queries are faster?”
- “Why don’t you just install the database on a faster machine?”
- “Can’t you audit and fix my queries?  
Geesh!”



performance fast responsive

reliable **scalable** quick optimized





YOU are the application developer.

If your application uses a database,  
YOU are the one responsible for  
what it contains.





# Research - what will your application do?

- How fast do you need to return data?
- What will users expect?
- How often will the data change?
- Are you more like NYTimes.com or MySpace?



# The Rules

- Preserve referential integrity.
- Normalize.
- Code organization and standards.

Always treat your database code  
as any other code!



# The importance of referential integrity.

- Old school:

“We cancelled the order, but items in the order were never returned to the inventory!”

- Kids these days:

“I deleted my profile but people can still see me as a friend on their page!”

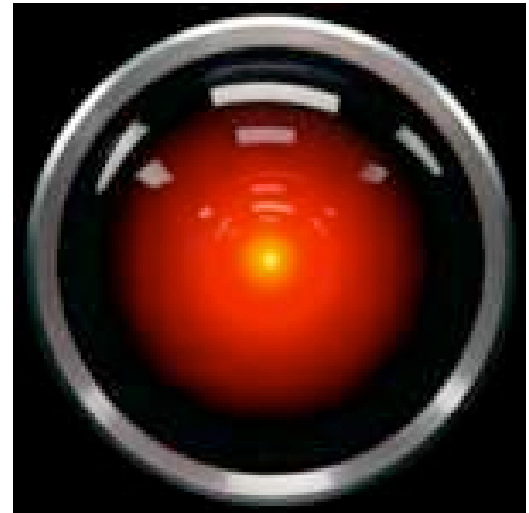


# Constraints

- “not null”, primary keys, foreign keys, unique keys, check constraints...

Let the database handle what happens when data is modified.

(Yes, it's smarter than you.)



Example: “not null” constraint

book
id <b>not null</b>
title
author_id
price

author
id
first_name
last_name

id integer not null,



## Example: primary key constraint

book
id not null <b>PK</b>
title
author_id
price

author
id
first_name
last_name

id integer not null  
constraint book\_pk primary key,



## Example: foreign key constraint

book
id not null PK
title
author_id <b>FK</b>
price

author
id
first_name
last_name

```
author_id not null  
constraint book_author_fk  
references author on delete cascade,
```



## Example: unique key constraint

book
id PK
title
author_id FK
price

} UK

author
id
first_name
last_name

```
constraint book_title_author_uk  
    unique (title, author_id),
```



## Example: check constraint

book	
id	PK
title	} UK
author_id	
price	CK

author	
id	
first_name	
last_name	

```
price number not null
constraint book_price_ck
check (price > 0),
```



## **DDL** (Data Definition Language)

- CREATE
- ALTER

## **DML** (Data Modification Language)

- INSERT
- UPDATE
- DELETE

## **SQL** (Structured Query Language)

- SELECT



# Stored Procedures

INSERT - UPDATE - DELETE

- Keep PHP easy and simple: put DML in stored procedures.



```
update book  
  set price = 24.99  
 where title = 'Cryptonomicon';
```



```
update book
  set price = :price
 where title = :title;
```



```
procedure update_book_price (  
    p_price book.price%type,  
    p_title book.title%type  
)  
is  
begin  
    update book  
        set price = p_price  
        where title = p_title;  
end;
```



```
procedure update_book_price (  
    p_book_price book.price%type,  
    p_book_title book.title%type  
)  
is  
    v_book_id number;  
begin  
    update book  
        set price = p_book_price  
        where title = p_book_title  
    returning id into v_book_id;  
  
    update_pending_orders (  
        p_book_id => v_book_id,  
        p_book_price => p_book_price);  
end;
```



Stored procedures are part of your application's code.

Make room for them in your lib.



“Why can’t you be more normal?”

- Aunt Mildred



# Don't be so repetitive repetitive

- Scared of joins?
- “But joining on so many tables is a performance hit!”
- Want easier queries?



# Database normalization is:



# Database normalization is:

minimization



# Database normalization is:

minimization  
of  
duplication



# Database normalization is:

minimization  
of  
duplication  
of  
information



# Database normalization is:

minimization  
of  
duplication  
of  
information

(pew!)

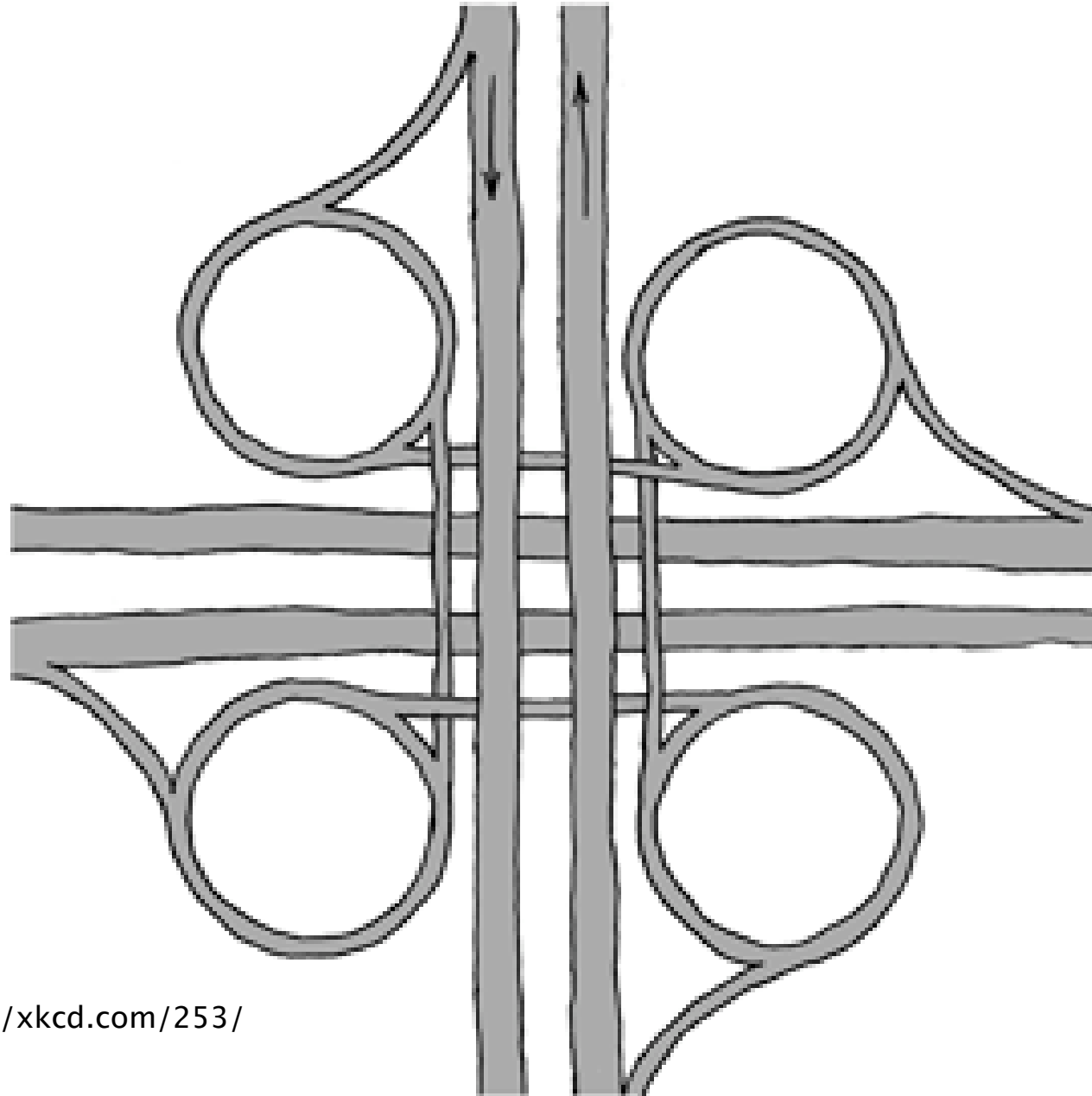


id	title	price	author
1	'Cryptonomicon'	24.99	'Neal Stephenson'
2	'Snow Crash'	5.99	'Neal Stephenson'
3	'The Name of the Rose'	12.99	'Umberto Eco'
4	'The Lion, The Witch and the Wardrobe'	5.99	'C. S. Lewis'

Table: book



# THE INESCAPABLE CLOVERLEAF:



<http://xkcd.com/253/>

Which is correct?

id	title	price	author
1	'Cryptonomicon'	24.99	'Neal Stephenson'
2	'Snow Crash'	5.99	'Neil Stephenson'
3	'The Name of the Rose'	12.99	'Umberto Eco'
4	'The Lion, The Witch and the Wardrobe'	5.99	'C. S. Lewis'

Table: book



How about authors without books?

id	title	price	author
1	'Cryptonomicon'	24.99	'Neal Stephenson'
2	'Snow Crash'	5.99	'Neal Stephenson'
3	'The Name of the Rose'	12.99	'Umberto Eco'
4	'The Lion, The Witch and the Wardrobe'	5.99	'C. S. Lewis'
5	???	???	'Maggie Nelson'

Table: book



Ordering authors by their last name?

id	title	price	author
1	'Cryptonomicon'	24.99	'Neal Stephenson'
2	'Snow Crash'	5.99	'Neal Stephenson'
3	'The Name of the Rose'	12.99	'Umberto Eco'
4	'The Lion, The Witch and the Wardrobe'	5.99	'C. S. Lewis'

Table: book





id	title	price	author_id
1	'Cryptonomicon'	24.99	1
2	'Snow Crash'	5.99	1
3	'The Name of the Rose'	12.99	2
4	'The Lion, The Witch and the Wardrobe'	5.99	3

Table: book



id	first_name	last_name
2	'Umberto'	'Eco'
3	'C. S.'	'Lewis'
4	'Maggie'	'Nelson'
1	'Neal'	'Stephenson'

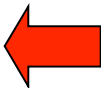


Table: author



It is **ALWAYS** possible to denormalize.

It is **SOMETIMES** possible to revert denormalization.



# Code organization and standards

- Have them.



## Example 1381. MySQL extension overview example

---

```
<?php
// Connecting, selecting database
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    or die('Could not connect: ' . mysql_error());
echo 'Connected successfully';
mysql_select_db('my_database') or die('Could not select database');

// Performing SQL query
$query = 'SELECT * FROM my_table';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());

// Printing results in HTML
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";

// Free resultset
mysql_free_result($result);

// Closing connection
mysql_close($link);
?>
```





# Organize your DB code



# Have a standard way to interact with the database

- Code to manage the db connection(s).
- Developers should only worry about writing good SQL and/or calling the right procedures.



# The PDO way

“The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP.”

<http://php.net/pdo>



```
<?php
```

```
$dbh = new PDO('oci:', 'scott', 'tiger');
```

```
$dbcode =
```

```
    "update book  
      set price = :price  
      where title = :title";
```

```
$stmt = $dbh->prepare($dbcode);
```

```
$stmt->bindParam(':price', $price);
```

```
$stmt->bindParam(':title', $title);
```

```
$stmt->execute();
```



```
?>
```

```
<?php
```

```
$dbh = new PDO('oci:', 'scott', 'tiger');
```

```
$dbcode =
```

```
    "begin
```

```
        update_book_price (
```

```
            p_price => :price,
```

```
            p_title => :title
```

```
        );
```

```
    end;";
```

```
$stmt = $dbh->prepare($dbcode);
```

```
$stmt->bindParam(':price', $price);
```

```
$stmt->bindParam(':title', $title);
```

```
$stmt->execute();
```



```
?>
```

# A quick word on caching



```
function getBooks($author_id)
{
    $dbh = new PDO('oci:', 'scott', 'tiger');
    $cache_key = 'getBooks' . $author_id;

    if (!$books = getFromCache($cache_key)) {

        $dbcode = "
            select title,
                   price
            from book
            where author_id = :author_id";

        $stmt = $dbh->prepare($dbcode);
        $stmt->bindParam(':author_id', $author_id);
        $stmt->execute();

        $books = $stmt->fetchAll();

        putInCache($cache_key, $books, 60*60);
    }

    return $books;
}
```



# Progress so far...

- Designed a nice database.
- Have an easy way to interact with it.
- Didn't need to bother DBAs.
  
- What's next?



# Optimization



“Premature optimization is the root of all evil.”

- D. Knuth



# Remember what your application is meant to accomplish

What do you want to gain? vs. What can you give up?



“There are no silver bullets - *none*. If there were they would be the default behavior and you would never hear about them.”

- Tom KYTE



Psst...

All this good design has already  
created a lot of potential for  
optimization...



We reused objects:

Primary keys, foreign keys, unique keys and other constraints can be used in place of separate indexing.



We conserved space:

Our data is now IDs, not large strings.



We improved lookup time:

Matching IDs, which are numeric, sequential and indexed is faster than matching strings.



# Optimization Tools

- Not DBAs...
- EXPLAIN PLAN
- Query trace
- <http://tahiti.oracle.com>
- Denormalization
- Common sense



Come see “How to Optimize a Database Query” tomorrow!



Questions?

