

Leveraging the Power of Oracle with PHP

Taking Advantage of the Database
PHP Quebec 2007

Roberto Mansfield
University of Pennsylvania
School of Arts & Sciences

Disclaimer

- I really like MySQL.
- I'm not bashing MySQL.
- Some of my best projects use MySQL.

Why use Oracle?

- Because its there (already using it for other projects).
- Because the customer requires it.
- Because we need to link into an existing Oracle data store.

Seven Tips for Oracle newbies (coming from MySQL)

- Primary keys (where's AUTO_INCREMENT!?)
- Date Handling
- Concatenation
- Left JOIN made easy
- Escaping values (bind variables)
- Nested queries, Inline views
- COMMIT!

1. Primary keys

- Mysql makes primary keys easy with the `auto_increment` column attribute.

```
CREATE TABLE my_table (  
    my_table_id    INT    not null primary key auto_increment,  
    etc...
```

- In Oracle, you define a sequence which guarantees a unique value:

```
CREATE SEQUENCE my_sequence STARTING WITH 1 INCREMENT BY 1;
```

1. Primary keys

- Then you select from the sequence to get your unique value:

```
SELECT my_sequence.nextval FROM dual;
```

- Or, use it directly in an INSERT statement:

```
INSERT INTO my_table (my_table_id, ... )  
VALUES (my_sequence.nextval, ... );
```

- You can use one sequence for all primary keys if you don't care about sequential values for all id fields.
- You can grant SELECT permission on a sequence just like on a table.

2. Date handling

- Use “sysdate” to get the current system time.

```
SELECT SYSDATE FROM DUAL;
```

- Default format for date output is:
day-Month-year (e.g., 08-AUG-07)

2. Date handling

- Use `TO_STRING()` to format dates for output from your query.

```
SELECT TO_STRING(date_field, 'mm/dd/yyyy hh24:mi:ss'), ... ;
```

- See Oracle docs online for all date format elements.
- Don't parse dates in PHP – make the database do the work!

2. Date handling

- Use TO_DATE() to format dates for input:

```
INSERT INTO table (date_field)
VALUES ( TO_DATE('01/01/2007', 'mm/dd/yyyy') );
```

- Or:

```
UPDATE table SET field = value
WHERE date_field = TO_DATE(...)
```

2. Date handling

- Internally, DATES are a floating point value
 - Integer portion is the current day
 - Fractional portion is the current time

- Examples:

- Today's date at the 'zero' hour:

```
SELECT TRUNC(SYSDATE) FROM DUAL;
```

- Twenty-four hours from now:

```
SELECT (SYSDATE + 1) FROM DUAL;
```

- Tomorrow at 9am:

```
SELECT TRUNC(SYSDATE) + 1 + (9/24) FROM DUAL;
```

3. Concatenation

- Oracle has the CONCAT function to join two strings together.
- Unlike MySQL, CONCAT takes only two parameters, so nesting is required to join additional fields:

```
SELECT CONCAT(CONCAT(first_name, ' '), last_name) AS name  
FROM person;
```

- Good thing we don't have a middle_name field.

3. Concatenation

- Of course, no one uses the CONCAT() function in Oracle.
- Instead we use the concat operator:

```
SELECT first_name || ' ' || last_name AS name  
FROM person;
```

4. Left JOIN

- Here's an example of SQL statement with a left join:

```
SELECT customer.name,  
       sales.order_number  
FROM customer  
LEFT JOIN sales ON  
       customer.customer_id = sales.customer_id;
```

- This retrieves all customer names and any matching sales order numbers.

4. Left JOIN

- Oracle has an easy short cut for specifying left joins using the (+) modifier.
- Write your SQL as usual, but use (+) to indicate restrictions which can be treated as left joins:

```
SELECT customer.name,  
       sales.order_number  
FROM customer,  
     sales  
WHERE customer.customer_id = sales.customer_id (+);
```

- This syntax simplifies writing joins especially when multiple left joins are needed.

5. Escaping values and variable binding

- Oracle uses a repeated single quote rather than a backslash for escaping the quote character:

```
INSERT INTO table (field) VALUES ('Isn''t this a nice day?');
```

- Make sure PHP's magic quotes are off – or you'll need to write code to remove the unnecessary backslashes.

5. Escaping values and variable binding

- The safest approach is to use variable binding:

```
$id = 1;
```

```
$field = $_REQUEST['bad_user_input'];
```

```
$sql = "INSERT INTO test (id, field) VALUES (:id, :field)";
```

```
$cursor = oci_parse($db, $sql);
```

```
oci_bind_by_name($cursor, "id", $id);
```

```
oci_bind_by_name($cursor, "field", $field);
```

```
oci_execute($cursor);
```

6. Nested queries (and inline views)

- Nested queries (supported in MySQL 4.1 and later) make many tasks easier:

```
SELECT c.customer_id,  
       c.customer_name,  
       ( SELECT count(*)  
         FROM orders o  
         WHERE o.customer_id = c.customer_id )  
FROM customer c;
```

6. Nested queries (and inline views)

- A nested query can also appear in the FROM clause or WHERE clause:

```
SELECT c.customer_id,  
       c.customer_name  
FROM customer c  
WHERE ( SELECT count(*)  
        FROM orders o  
        WHERE o.customer_id = c.customer_id ) > 100;
```

6. Nested queries (and inline views)

- A nested query in the FROM clause is usually called an inline view.

```
SELECT cust.customer_id,  
       cust.customer_name,  
       order.order_id,  
       order.order_date,  
       order.description  
FROM   customer c,  
       order,  
       ( SELECT customer_id,  
           MAX(order_date)  
         FROM orders  
         GROUP BY customer_id ) view  
WHERE  cust.customer_id = order.customer_id  
       AND cust.customer_id = view.customer_id  
       AND order.order_date = view.order_date;
```

7. COMMIT

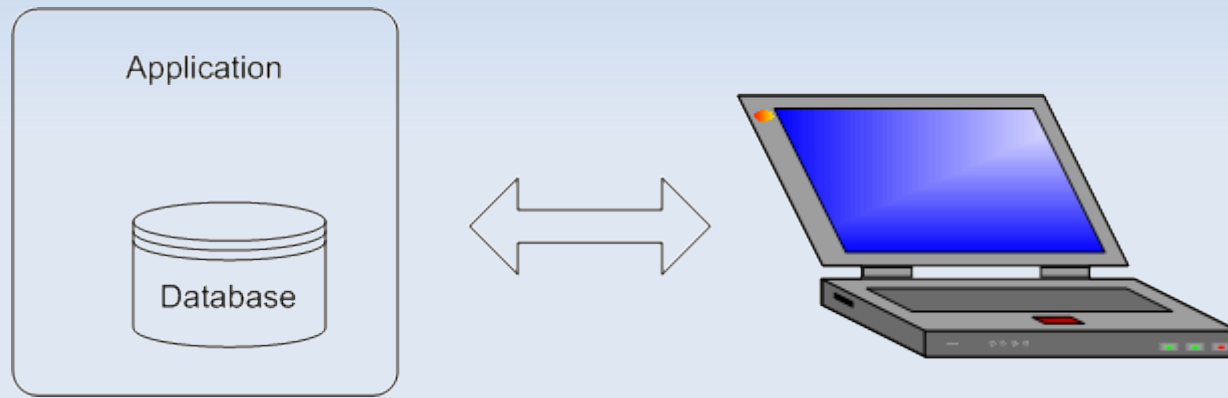
- Don't forget to COMMIT after a database update.
- This goes for your code AND your database tool.
- Be aware of how your database commands (either oci or an abstraction layer) handles commits for you.

```
oci_execute($cursor);           // defaults to autocommit
oci_execute($cursor, OCI_DEFAULT); // the OCI_DEFAULT is no autocommit
```

Gross Generalizations about LAMP Environment

- When developers first start coding:
 - We think about the application first and the database second.
 - The application and database are tightly coupled.
- And since developers are lazy:
 - We aren't encouraged to develop better database skills.
 - We don't bother to learn how we can benefit from the evolving features in MySQL.

Application-centric Design



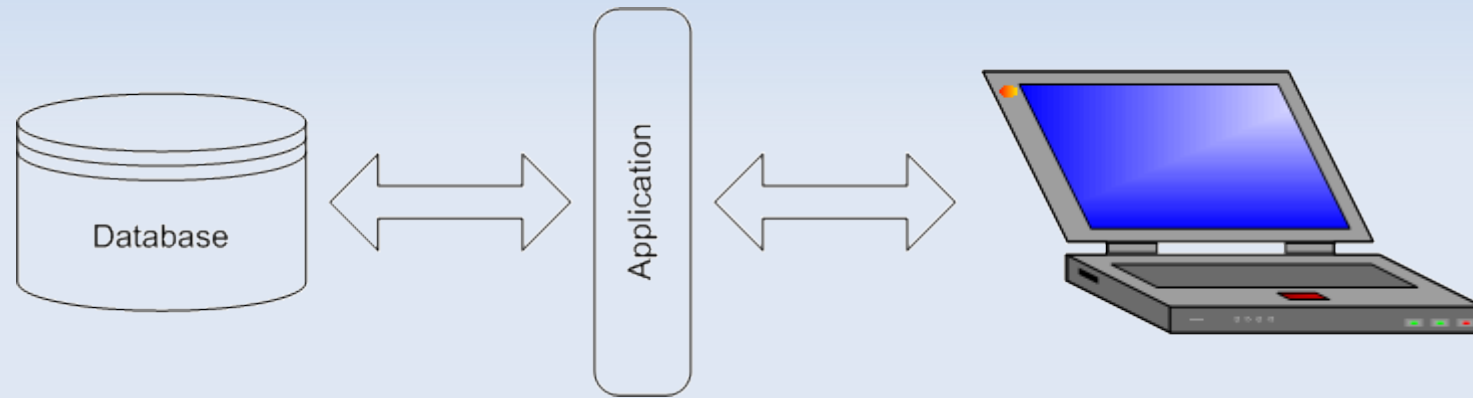
What's wrong with this approach?

- Nothing.
 - Data integrity is enforced by the PHP app.
 - Data model documented in PHP code.
 - Use well known PHP to parse/manipulate data.
- Everything.
 - Data integrity can be enforced by database layer.
 - Data model can be documented in database schema.
 - Take advantage of Oracle functions to manipulate data.

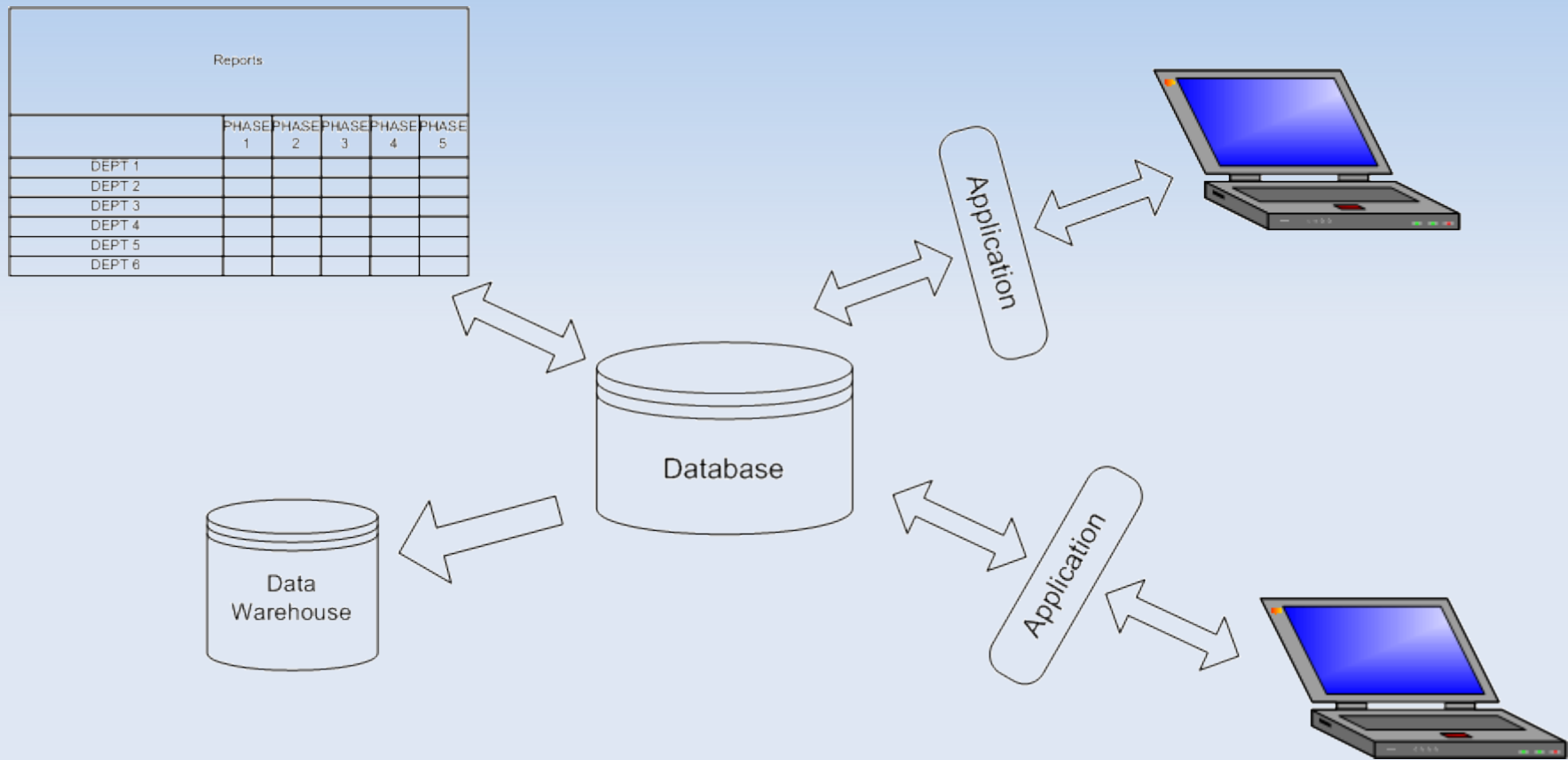
What are we missing?

- Data needs change, so we need a robust data model that is flexible.
- Data will always find new uses, so we need a database that can stand apart from the application.
- Data needs to be consistent, so we need the database to police itself.
- Data needs to be secure, so we want to provide only the minimum access required to the application.

Data-centric Design



Data-centric Design



The Typical 1st Database App

- Create some tables
- `mysql/oci_connect()`
- `mysql/oci_query()`
- `mysql/oci_fetch()`
- Generate a page
- Easy as pie!

Designing your database

- Building a good data model is time well spent.
- Spec columns to reasonable types and sizes (not all TEXT). Consider future reporting needs.
- Determine the relationships between tables
- Keep an up to date data model for documentation

Build your database

- Maintain an up-to-date SQL script for table creation – great for documentation.
- Create primary keys and indexes
 - Index columns which drive queries
 - Index foreign keys

Build your database

- **Create constraints:**

- Foreign keys (with or without cascading deletes)
- Acceptable column values
- MySQL supports constraints using the InnoDB engine

```
CREATE TABLE orders (  
  order_id      INT      NOT NULL PRIMARY KEY,  
  customer_id   INT      NOT NULL,  
  ...  
  CONSTRAINT cust_id_fk  
    FOREIGN KEY (customer_id)  
    REFERENCES customer (customer_id)  
    ON DELETE CASCADE);
```

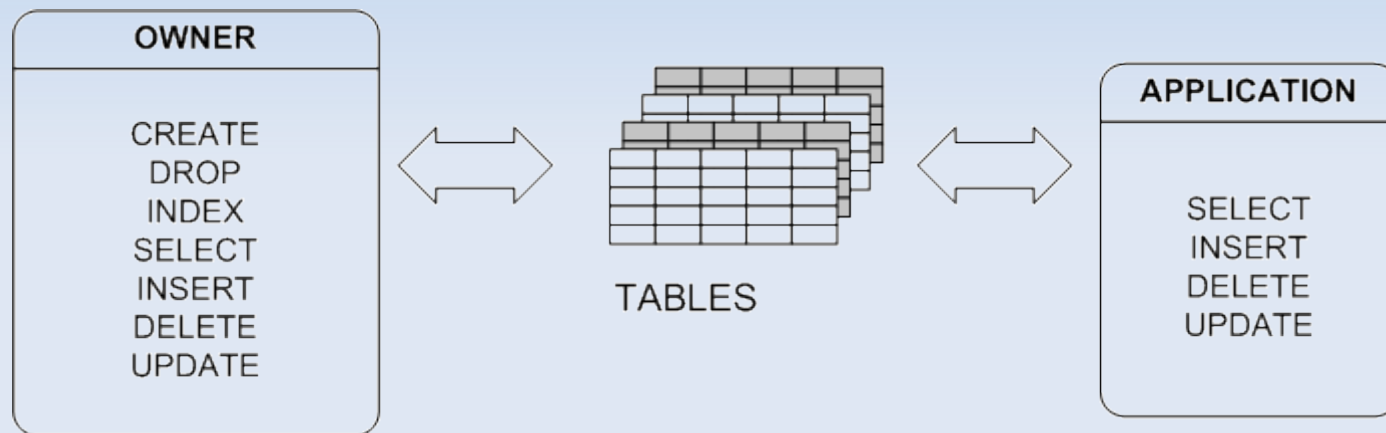
Functions and Stored Procedures

- Coding benefits:
 - Compiled, run fast, run on the database server.
 - Include complex logic directly in your query.
 - Keep your business logic with the database.
 - Avoid repeating code across different platforms.
- Security benefits:
 - Better privilege separation since access is granted to functions/procedures and NOT directly to tables.
 - Reduce SQL injection risk.

Privilege separation

- Stage 1 of privilege separation:
 - Create one account for administering the database
 - Create **another** account for accessing the database from the web application
 - Grant minimal privileges to the application account.
- The application usually needs INSERT, DELETE and UPDATE unless its read-only view of the data.

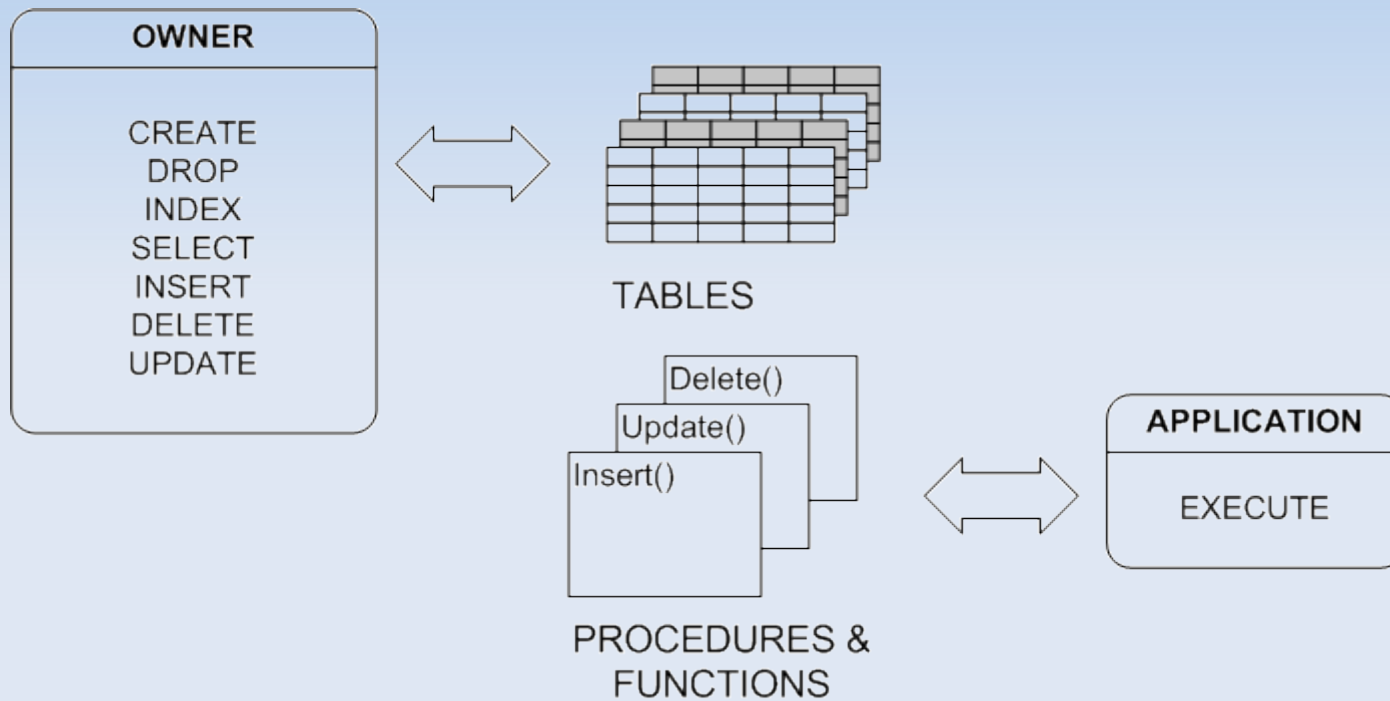
Privilege separation



Privilege separation

- Stage 2 of privilege separation:
 - Create functions and procedures for data access and data updates
 - Grant the application account access to the functions and procedures
 - The application account no longer needs direct access to the tables!

Privilege separation



Functions

- A simple PL/SQL function example:

```
CREATE or REPLACE FUNCTION order_count (p_customer_id IN NUMBER)
RETURN NUMBER
AS
    v_count NUMBER;
BEGIN

    SELECT count(*) INTO v_count
    FROM orders
    WHERE customer_id = p_customer_id;

    RETURN v_count;

END;
```

Functions

- We can now call the `order_count()` function directly, or use it within other queries:

```
$sql = "SELECT first_name,  
            last_name,  
            address,  
            phone,  
            email  
        FROM customer  
        WHERE order_count(customer_id) < 50";
```

Procedures

- A PL/SQL procedure example:

```
CREATE or REPLACE PROCEDURE delete_customer (p_customer_id IN number,
                                             p_result      OUT varchar2)
AS
BEGIN
    p_result := null;
    -- need to delete customer data and all order data in
    -- customer, orders, and order_detail tables
    DELETE FROM order_detail
        WHERE order_id IN ( SELECT order_id
                           FROM orders
                           WHERE customer_id = p_customer_id );

    DELETE FROM orders   WHERE customer_id = p_customer_id;
    DELETE FROM customer WHERE customer_id = p_customer_id;

    COMMIT;

EXCEPTION
    -- catch all exception handler
    WHEN OTHERS THEN
        ROLLBACK;
        p_result := SQLERRM;
        RETURN;

END;
```

Procedures

- On the PHP side, we can write a function/method to interface with the PL/SQL procedure:

```
function delete_customer($customer_id) {  
  
    global $db;  
  
    $sql = "CALL schema_owner.delete_customer(:customer_id, :sql_error)";  
  
    $cursor = oci_parse($db, $sql);  
  
    oci_bind_by_name($cursor, "customer_id", $customer_id, -1);  
    oci_bind_by_name($cursor, "sql_error", $sql_error, 200);  
  
    oci_execute($cursor, OCI_DEFAULT);  
  
    oci_free_cursor($cursor);  
  
    if ( $sql_error ) {  
        do_some_error_handling($sql_error);  
    }  
  
}
```

Procedure with Dynamic SQL

- Let's say we want to create a generic PL/SQL procedure to update an arbitrary field in an arbitrary table.

Procedure with Dynamic SQL

```
CREATE or REPLACE PROCEDURE field_update ( p_table      IN  varchar2,
                                           p_field      IN  varchar2,
                                           p_value      IN  varchar2,
                                           p_id_field   IN  varchar2,
                                           p_id_value   IN  varchar2)

AS
    v_statement      varchar2(500);

BEGIN
    -- this is a really UNSAFE way to do dynamic sql!!
    v_statement := 'UPDATE ' || p_table      || '
                   SET   ' || p_field      || ' = ''' || cp_value      || '''
                   WHERE ' || p_id_field   || ' = ''' || cp_id_value   || '''';

    execute immediate v_statement;

    commit;

END;
```

Procedure with Dynamic SQL

```
CREATE or REPLACE PROCEDURE field_update ( p_table      IN  varchar2,
                                           p_field      IN  varchar2,
                                           p_value      IN  varchar2,
                                           p_id_field   IN  varchar2,
                                           p_id_value   IN  varchar2)

AS
    v_statement      varchar2(500);

BEGIN
    -- make sure valid fields and tables are passed
    if ( is_valid_field(p_table, p_field) AND is_valid_field(p_table, p_id_field) ) then
        v_statement := 'UPDATE ' || p_table      || '
                        SET ' || p_field      || ' = :cp_value
                        WHERE ' || p_id_field || ' = :cp_id_value';
        execute immediate v_statement using p_value, p_id_value;
        commit;
    end if;
END;
```

Procedure with Dynamic SQL

```
CREATE or REPLACE FUNCTION is_valid_field ( p_table IN varchar2,  
                                           p_field IN  varchar2 )  
  
RETURN boolean IS  
    v_count  BINARY_INTEGER := 0;  
BEGIN  
    SELECT count(*) INTO v_count  
    FROM valid_table_fields  
    WHERE table_name = UPPER(p_table)  
    AND column_name = UPPER(p_field);  
    return ( v_count = 1 );  
END;
```

- We could easily implement this procedure and function in PHP, but by doing it in pl/sql we require only one round trip to the database instead of three.

Oracle and XML

- We can also reduce round trips to the database by returning query data in XML.

Oracle and XML

- With Oracle, you can write queries which return full XML docs or XML fragments. Simplest case:

```
SELECT XMLElement("last_name", last_name)
       FROM employee;
```

- Returns:

```
<last_name>Smith</last_name>
```

```
<last_name>Johnson</last_name>
```

etc...

- Note: We get a separate record returned for each item in the database (we'll come back to this).

Oracle and XML

- We can also add an attribute to the tag (which we can also alias):

```
SELECT XMLElement("last_name",  
                XMLAttributes(employee_id "employee_id"),  
                LAST_NAME)  
FROM employee  
WHERE employee_id = 71;
```

```
<last_name employee_id="71">Smith</last_name>
```

- What if we want more than one column?

Oracle and XML

- For multiple columns, we use XMLForest():

```
SELECT XMLForest(LAST_NAME "last_name",  
                FIRST_NAME "first_name",  
                PHONE      "phone")  
FROM employee  
WHERE employee_id = 71;
```

```
<last_name>Smith</last_name><first_name>Ellen</first_name>  
<phone>800-555-5555</phone>
```

Oracle and XML

- Or, we use XMLConcat():

```
SELECT XMLConcat(  
        XMLElement("last_name", LAST_NAME),  
        XMLElement("first_name", FIRST_NAME),  
        XMLElement("phone", PHONE) )  
FROM employee  
WHERE employee_id = 71;
```

```
<last_name>Smith</last_name><first_name>Ellen</first_name>  
<phone>800-555-5555</phone>
```

- Okay, okay, but what if we want data across multiple records?

Oracle and XML

- We use the XMLAgg() function to aggregate XML results from across several rows:

```
SELECT XMLAgg(XMLForest(LAST_NAME "last_name",  
                        FIRST_NAME "first_name"))  
FROM employee;
```

```
<last_name>Smith</last_name><first_name>Ellen</first_name>  
<last_name>Johnson</last_name><first_name>Ted</first_name>  
<last_name>Miller</last_name><first_name>Dana</first_name>
```

- Note: returns data as one record

Oracle and XML

- Now we use all these functions and nest them to get useful XML results.

```
SELECT XMLElement("employees",
                XMLAgg(XMLElement("employee",
                                XMLForest(LAST_NAME "last_name",
                                           FIRST_NAME "first_name"))))
FROM employee;
```

```
<employees>
  <employee>
    <last_name>Smith</last_name>
    <first_name>Ellen</first_name>
  </employee>
  <employee>
    ...
  </employee>
</employees>
```

Oracle and XML

- Caveat #1:
 - The XML functions return data as XMLType.
 - The OCI interface in PHP does not support the XMLType data type.
 - So we must convert from XMLType to STRING or CLOB to retrieve data into PHP.

```
SELECT XMLElement("employees",
                XMLAgg(XMLElement("employee",
                                XMLForest(LAST_NAME "last_name",
                                           FIRST_NAME "first_name")))).getClobVal()
FROM employee;
```

Oracle and XML

- Caveat #2:
 - As you can imagine, nesting the XML functions gets ugly fast.
 - For anything except simple XML extracts, you will probably want to write PL/SQL functions to make your query less complicated.

Oracle and XML

- Caveat #3:
 - Sorting elements in XMLAgg is supported in Oracle 10g.
 - Under 9i, use a sorted inline view if element order is important.

Example: xml_faculty()

```
CREATE or REPLACE FUNCTION xml_faculty (p_faculty_id IN NUMBER)
RETURN CLOB
AS
    v_xml CLOB;
BEGIN

    SELECT XMLElement("faculty",
                    XMLForest(f.FACULTY_ID           "faculty_id",
                              f.LAST_NAME           "last_name",
                              f.FACULTY_TYPE        "faculty_type",
                              f.DEPT_ORG            "dept_org",
                              f.JOB_TITLE           "job_title",
                              xml_research(f.faculty_id),
                              xml_committees(f.faculty_id)
                    ).getClobVal() INTO v_xml
    FROM fdb_faculty      f
    WHERE f.faculty_id = p_faculty_id;

    RETURN v_xml;

EXCEPTION
    WHEN NO_DATA_FOUND THEN RETURN null;

END;
```

Example: xml_committees()

```
CREATE OR REPLACE FUNCTION XML_COMMITTEES (p_faculty_id IN NUMBER)
RETURN XMLTYPE
AS
    x_xml XMLType := null;
BEGIN

    -- we use an inline view to overcome ordering limitations in XMLAgg under 9i
    SELECT XMLAgg(
        XMLElement("committee",
            XMLForest(v.COMMITTEE_MEMBERSHIP_ID "committee_membership_id",
                v.COMMITTEE_CODE "committee_code",
                v.COMMITTEE_DESCR "committee_descr")
            )
        ) INTO x_xml
    FROM ( SELECT cm.committee_membership_id,
        cm.committee_code,
        c.committee_descr
        FROM fdb_committee_membership cm, fdb_committee c
        WHERE cm.faculty_id = p_faculty_id
        AND cm.committee_code = c.committee_code
        ORDER BY c.committee_descr) v;

    RETURN x_xml;

EXCEPTION
    WHEN NO_DATA_FOUND THEN RETURN x_xml;
END;
```

Example:

- select xml_faculty(51) from dual;

```
<faculty>
  <faculty_id>51</faculty_id>
  <last_name>Smith</last_name>
  <first_name>Jacque</first_name>
  <faculty_type>ST</faculty_type>
  <dept_org>History</dept_org>
  <job_title>Professor</job_title>
  <research>Colonial America</research>
  <research>American Revolution</research>
  <committee>
    <committee_membership_id>782</committee_membership_id>
    <committee_code>GRAD</committee_code>
    <committee_descr>Graduate Admissions Committee</committee_descr>
  </committee>
  <committee>
    <committee_membership_id>981</committee_membership_id>
    <committee_code>RECR</committee_code>
    <committee_descr>Faculty Recruitment Committee</committee_descr>
  </committee>
</faculty>
```

Example: PHP code

- Class method / function to retrieve faculty data:

```
function retrieve($faculty_id) {  
  
    $faculty_id = intval($faculty_id);  
    if ( !$faculty_id ) return;  
    global $db;  
    $sql = "SELECT faculty.xml_faculty($faculty_id) AS xml FROM DUAL";  
    $cursor = oci_parse($db, $sql);  
    oci_execute($cursor, OCI_DEFAULT);  
  
    $result = oci_fetch_assoc($cursor, OCI_RETURN_LOBS);  
    $clob = $result['XML'];  
  
    oci_free_cursor($cursor);  
  
    $xml = simplexml_load_string($clob);  
    return $xml;  
}
```

print_r(\$xml);

SimpleXMLElement Object

```
(
  [faculty_id] => 51
  [last_name] => Smith
  [first_name] => Jacque
  [faculty_type] => ST
  [dept_org] => History
  [job_title] => Professor
  [research] => SimpleXMLElement Object
    (
      [0] => Colonial America
      [1] => American Revolution
    )
  [committee] => Array
    (
      [0] => SimpleXMLElement Object
        (
          [committee_membership_id] => 782
          [committee_code] => GRAD
          [committee_descr] => Graduate Admissions Committee
        )
      [1] => SimpleXMLElement Object (etc...)
```

SimpleXML Gotcha

- The SimpleXML object can return either strings or arrays for its child nodes.
- You may need to “cast” the xml object as string to return what you need if usage isn't obvious.
- E.g.,
 - `print $xml->tag;` (works)
 - `print $array[$xml->key];` (fails)
 - `print $array[(string) $xml->key];` (works!)
- No problems found in Smarty templates!

Resources

- Oracle 10g Documentation Master Index (registration required)
http://download-west.oracle.com/docs/cd/B14117_01/mix.101/b12039/toc.htm
- Oracle SQL Developer (java GUI interface for Oracle databases)
http://www.oracle.com/technology/products/database/sql_developer/index.html
- Using PHP 5 with Oracle XML DB
<http://www.oracle.com/technology/oramag/oracle/05-jul/o45php.html>
- Building Database-Driven PHP Applications on Oracle XML DB
http://www.oracle.com/technology/pub/articles/vasiliev_xmlldb_php.html
- Underground PHP Oracle Manual
<http://www.oracle.com/technology/tech/php/pdf/underground-php-oracle-manual.pdf>
- You'll find the link to download this presentation at:
<http://phpclimber.blogspot.com>
- Email me at:
robertom@sas.upenn.edu